

---

# DeePMD-kit

DeepModeling

Feb 04, 2024



# GETTING STARTED

1	Getting Started	3
1.1	Easy install	3
1.1.1	Install off-line packages	3
1.1.2	Install with conda	4
1.1.3	Install with docker	4
1.1.4	Install Python interface with pip	5
1.2	DeePMD-kit Quick Start Tutorial	5
1.2.1	Task	6
1.2.2	Table of contents	6
1.2.3	Get tutorial data via git	6
1.2.4	General Introduction	6
1.2.5	Data preparation	7
1.2.6	Prepare input script	9
1.2.7	Train a model	10
1.2.8	Freeze a model	16
1.2.9	Test a model	17
1.2.10	Run MD with LAMMPS	20
2	Installation	25
2.1	Easy install	25
2.1.1	Install off-line packages	25
2.1.2	Install with conda	26
2.1.3	Install with docker	26
2.1.4	Install Python interface with pip	27
2.2	Install from source code	27
2.2.1	Install the python interface	28
2.2.2	Install the C++ interface	31
2.3	Install from pre-compiled C library	33
2.3.1	Use Pre-compiled C Library to build the LAMMPS plugin and GROMACS patch	33
2.4	Install LAMMPS	33
2.4.1	Install LAMMPS's DeePMD-kit module (built-in mode)	33
2.4.2	Install LAMMPS (plugin mode)	35
2.5	Install i-PI	35
2.6	Install GROMACS with DeepMD	36
2.6.1	Patch source code of GROMACS	36
2.6.2	Compile GROMACS with deepmd-kit	36
2.7	Building conda packages	36
2.8	Install Node.js interface	37
2.8.1	Install from npm	37
2.8.2	Build from source	37

2.9	Easy install the latest development version . . . . .	38
2.9.1	Install with docker . . . . .	38
2.9.2	Install with pip . . . . .	38
2.9.3	Download pre-compiled C Library . . . . .	38
3	Data . . . . .	39
3.1	System . . . . .	39
3.2	Formats of a system . . . . .	41
3.2.1	NumPy format . . . . .	41
3.2.2	HDF5 format . . . . .	41
3.2.3	Raw format and data conversion . . . . .	42
3.3	Prepare data with dpdata . . . . .	42
4	Model . . . . .	45
4.1	Overall . . . . .	45
4.1.1	Theory . . . . .	45
4.1.2	Instructions . . . . .	45
4.2	Descriptor "se_e2_a" . . . . .	46
4.2.1	Theory . . . . .	47
4.2.2	Instructions . . . . .	47
4.3	Descriptor "se_e2_r" . . . . .	48
4.3.1	Theory . . . . .	48
4.3.2	Instructions . . . . .	49
4.4	Descriptor "se_e3" . . . . .	50
4.4.1	Theory . . . . .	50
4.4.2	Instructions . . . . .	50
4.5	Descriptor "se_atten" . . . . .	51
4.5.1	DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation	51
4.5.2	Theory . . . . .	51
4.5.3	Introduction to new features of DPA-1 . . . . .	52
4.5.4	Data format . . . . .	54
4.5.5	Training example . . . . .	55
4.6	Descriptor "hybrid" . . . . .	55
4.6.1	Theory . . . . .	55
4.6.2	Instructions . . . . .	55
4.7	Determine sel . . . . .	56
4.8	Fit energy . . . . .	56
4.8.1	Theory . . . . .	56
4.8.2	The fitting network . . . . .	57
4.8.3	Loss . . . . .	58
4.9	Fit spin energy . . . . .	58
4.9.1	Spin . . . . .	59
4.9.2	Spin Loss . . . . .	59
4.10	Fit tensor like Dipole and Polarizability . . . . .	60
4.10.1	Theory . . . . .	60
4.10.2	The fitting Network . . . . .	61
4.10.3	Loss . . . . .	61
4.10.4	Training Data Preparation . . . . .	62
4.10.5	Train the Model . . . . .	62
4.11	Fit electronic density of states (DOS) . . . . .	63
4.11.1	The fitting Network . . . . .	63
4.11.2	Loss . . . . .	64
4.11.3	Training Data Preparation . . . . .	64
4.11.4	Train the Model . . . . .	64

4.11.5	Test the Model . . . . .	65
4.12	Type embedding approach . . . . .	66
4.12.1	Theory . . . . .	66
4.12.2	Instructions . . . . .	67
4.13	Descriptor "se_a_mask" . . . . .	68
4.14	Deep potential long-range (DPLR) . . . . .	69
4.14.1	Theory . . . . .	70
4.14.2	Train a deep Wannier model for Wannier centroids . . . . .	70
4.14.3	Train the DPLR model . . . . .	71
4.14.4	Molecular dynamics simulation with DPLR . . . . .	71
4.15	Deep Potential - Range Correction (DPRc) . . . . .	74
4.15.1	Theory . . . . .	74
4.15.2	Training data . . . . .	75
4.15.3	Training the DPRc model . . . . .	75
4.15.4	Run MD simulations . . . . .	77
4.15.5	Pairwise DPRc . . . . .	77
4.16	Linear model . . . . .	80
4.17	Interpolation or combination with a pairwise potential . . . . .	80
4.17.1	Theory . . . . .	80
4.17.2	Table file . . . . .	81
4.17.3	Interpolation with a short-range pairwise potential . . . . .	81
4.17.4	Combination with a pairwise potential . . . . .	82
5	Training . . . . .	83
5.1	Train a model . . . . .	83
5.2	Advanced options . . . . .	84
5.2.1	Learning rate . . . . .	84
5.2.2	Training parameters . . . . .	85
5.2.3	Options and environment variables . . . . .	87
5.2.4	Adjust <code>se1</code> of a frozen model . . . . .	88
5.3	Training Parameters . . . . .	89
5.4	Parallel training . . . . .	131
5.4.1	Tuning learning rate . . . . .	131
5.4.2	Scaling test . . . . .	131
5.4.3	How to use . . . . .	132
5.4.4	Logging . . . . .	132
5.5	Multi-task training . . . . .	133
5.5.1	Theory . . . . .	133
5.5.2	Perform the multi-task training . . . . .	133
5.5.3	Initialization from pretrained multi-task model . . . . .	134
5.5.4	Share layers among energy fitting networks . . . . .	135
5.6	TensorBoard Usage . . . . .	136
5.6.1	Highlighted features . . . . .	136
5.6.2	How to use Tensorboard with DeePMD-kit . . . . .	136
5.6.3	Examples . . . . .	137
5.6.4	Attention . . . . .	141
5.7	Known limitations of using GPUs . . . . .	141
5.8	Finetune the pretrained model . . . . .	142
6	Freeze and Compress . . . . .	143
6.1	Freeze a model . . . . .	143
6.2	Compress a model . . . . .	143
6.2.1	Theory . . . . .	143
6.2.2	Instructions . . . . .	144

7	Test	147
7.1	Test a model	147
7.2	Calculate Model Deviation	148
7.2.1	Theory	148
7.2.2	Instructions	148
7.2.3	Relative model deviation	149
8	Inference	151
8.1	Python interface	151
8.1.1	External neighbor list algorithm	152
8.2	C/C++ interface	152
8.2.1	C++ interface	152
8.2.2	C interface	153
8.2.3	Header-only C++ library interface (recommended)	154
8.3	Node.js interface	155
9	Command line interface	157
9.1	Named Arguments	157
9.2	Valid subcommands	157
9.3	Sub-commands	157
9.3.1	transfer	157
9.3.2	train	158
9.3.3	freeze	159
9.3.4	test	160
9.3.5	compress	161
9.3.6	doc-train-input	162
9.3.7	model-devi	162
9.3.8	convert-from	163
9.3.9	neighbor-stat	164
9.3.10	train-nvnmd	165
9.3.11	gui	165
10	Integrate with third-party packages	167
10.1	Use deep potential with ASE	167
10.2	Run MD with LAMMPS	167
10.2.1	units	167
10.2.2	Enable DeePMD-kit plugin (plugin mode)	168
10.2.3	pair_style deepmd	168
10.2.4	Compute tensorial properties	170
10.2.5	Long-range interaction	170
10.2.6	Use of the centroid/stress/atom to get the full 3x3 “atomic-virial”	171
10.2.7	Computation of heat flux	171
10.3	Run path-integral MD with i-PI	172
10.4	Running MD with GROMACS	173
10.4.1	DP/MM Simulation	173
10.4.2	All-atom DP Simulation	175
10.5	Interfaces out of DeePMD-kit	176
10.5.1	dpdata	176
10.5.2	OpenMM plugin for DeePMD-kit	176
10.5.3	Amber interface to DeePMD-kit	176
10.5.4	CP2K interface to DeePMD-kit	176
10.5.5	DP-GEN	177
10.5.6	MAtom	177
10.5.7	ABACUS	177

11	Use NVNMD	179
11.1	Introduction	179
11.1.1	Training	179
11.1.2	Input script	180
11.1.3	Training	182
11.2	Testing	183
11.3	Running MD in Bohrium	183
11.3.1	Registration	183
11.3.2	Top-up and create a project	184
11.3.3	Run job	185
11.3.4	Check job status	186
11.3.5	Terminate and delete jobs	187
11.3.6	Download Results	187
11.4	Running MD in Nvnmd website	188
11.4.1	Account application	188
11.4.2	Adding task	188
11.4.3	Cancelling calculation	190
11.4.4	Downloading results	191
11.4.5	Deleting record	191
11.4.6	Clearing records	192
12	FAQs	193
12.1	How to tune Fitting/embedding-net size ?	193
12.1.1	Al2O3	193
12.1.2	Cu	194
12.1.3	Water	195
12.1.4	Mg-Al	196
12.2	How to control the parallelism of a job?	196
12.2.1	MPI (optional)	197
12.2.2	Parallelism between independent operators	197
12.2.3	Parallelism within an individual operators	197
12.2.4	Tune the performance	198
12.3	Do we need to set rcut < half boxsize?	198
12.4	How to set sel?	198
12.5	Installation	199
12.5.1	Inadequate versions of gcc/g++	199
12.5.2	Build files left in DeePMD-kit	199
12.6	The temperature undulates violently during the early stages of MD	199
12.7	MD: cannot run LAMMPS after installing a new version of DeePMD-kit	199
12.8	Model compatibility	200
12.9	Why does a model have low precision?	200
12.9.1	Data	200
12.9.2	Model	201
12.9.3	Training	201
13	Find DeePMD-kit C/C++ library from CMake	203
14	Create a model	205
14.1	Design a new component	205
14.2	Register new arguments	205
14.3	Package new codes	206
15	Atom Type Embedding	207
15.1	Overview	207
15.2	Preliminary	207

15.3	How to use . . . . .	208
15.4	Code Modification . . . . .	208
15.4.1	trainer (train/trainer.py) . . . . .	208
15.4.2	model (model/ener.py) . . . . .	208
15.4.3	embedding net (descriptor/se*.py) . . . . .	209
15.4.4	fitting net (fit/ener.py) . . . . .	209
16	Coding Conventions . . . . .	211
16.1	Preface . . . . .	211
16.2	Python . . . . .	211
16.2.1	Rules . . . . .	211
16.2.2	Whitespace . . . . .	212
16.2.3	General advice . . . . .	212
16.2.4	Writing documentation in the code . . . . .	212
16.3	C++ . . . . .	213
16.4	Run scripts to check the code . . . . .	213
17	CI/CD . . . . .	215
17.1	CI . . . . .	215
17.1.1	Test CUDA . . . . .	215
17.2	CD . . . . .	215
18	Python API . . . . .	217
18.1	backend package . . . . .	217
18.1.1	Submodules . . . . .	217
18.1.2	backend.dp_backend module . . . . .	217
18.1.3	backend.dynamic_metadata module . . . . .	217
18.1.4	backend.find_tensorflow module . . . . .	217
18.1.5	backend.read_env module . . . . .	218
18.2	deepmd package . . . . .	219
18.2.1	Subpackages . . . . .	225
18.2.2	Submodules . . . . .	552
18.2.3	deepmd.calculator module . . . . .	552
18.2.4	deepmd.common module . . . . .	554
18.2.5	deepmd.env module . . . . .	559
18.2.6	deepmd.lmp module . . . . .	559
18.3	deepmd_utils package . . . . .	560
18.3.1	Subpackages . . . . .	560
18.3.2	Submodules . . . . .	601
18.3.3	deepmd_utils.common module . . . . .	601
18.3.4	deepmd_utils.env module . . . . .	603
18.3.5	deepmd_utils.main module . . . . .	604
19	OP API . . . . .	607
19.1	op_module . . . . .	607
19.2	op_grads_module . . . . .	659
20	C++ API . . . . .	669
20.1	Class Hierarchy . . . . .	669
20.2	File Hierarchy . . . . .	669
20.3	Full API . . . . .	670
20.3.1	Namespaces . . . . .	670
20.3.2	Classes and Structs . . . . .	672
20.3.3	Enums . . . . .	707
20.3.4	Functions . . . . .	708



20.3.5	Typedefs . . . . .	717
21	C API . . . . .	719
21.1	Class Hierarchy . . . . .	719
21.2	File Hierarchy . . . . .	719
21.3	Full API . . . . .	720
21.3.1	Namespaces . . . . .	720
21.3.2	Classes and Structs . . . . .	721
21.3.3	Functions . . . . .	739
21.3.4	Defines . . . . .	786
21.3.5	Typedefs . . . . .	787
22	Core API . . . . .	789
22.1	Class Hierarchy . . . . .	789
22.2	File Hierarchy . . . . .	790
22.3	Full API . . . . .	791
22.3.1	Namespaces . . . . .	791
22.3.2	Classes and Structs . . . . .	795
22.3.3	Unions . . . . .	805
22.3.4	Functions . . . . .	805
22.3.5	Variables . . . . .	844
22.3.6	Defines . . . . .	845
22.3.7	Typedefs . . . . .	850
23	License . . . . .	853
24	Authors and Credits . . . . .	855
24.1	Cite DeePMD-kit and methods . . . . .	855
24.2	Package Contributors . . . . .	856
24.3	Other Credits . . . . .	858
25	Logo . . . . .	859
	Bibliography . . . . .	861
	Python Module Index . . . . .	865
	Index . . . . .	867



DeePMD-kit is a package written in Python/C++, designed to minimize the effort required to build deep learning-based models of interatomic potential energy and force field and to perform molecular dynamics (MD). This brings new hopes to addressing the accuracy-versus-efficiency dilemma in molecular simulations. Applications of DeePMD-kit span from finite molecules to extended systems and from metallic systems to chemically bonded systems.

---

Important: The project DeePMD-kit is licensed under [GNU LGPLv3.0](#). If you use this code in any future publications, please cite the following publications for general purpose:

- Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. “DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics.” *Computer Physics Communications* 228 (2018): 178-184.
- Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li’ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang. “DeePMD-kit v2: A software package for Deep Potential models.” *J. Chem. Phys.*, 159, 054801 (2023).

In addition, please follow [this page](#) to cite the methods you used.

---



## GETTING STARTED

In this text, we will call the deep neural network that is used to represent the interatomic interactions (Deep Potential) the model. The typical procedure of using DeePMD-kit is

### 1.1 Easy install

There are various easy methods to install DeePMD-kit. Choose one that you prefer. If you want to build by yourself, jump to the next two sections.

After your easy installation, DeePMD-kit (`dp`) and LAMMPS (`lmp`) will be available to execute. You can try `dp -h` and `lmp -h` to see the help. `mpirun` is also available considering you may want to train models or run LAMMPS in parallel.

---

Note: Note: The off-line packages and conda packages require the [GNU C Library 2.17](#) or above. The GPU version requires [compatible NVIDIA driver](#) to be installed in advance. It is possible to force conda to [override detection](#) when installation, but these requirements are still necessary during runtime.

---

- [Install off-line packages](#)
- [Install with conda](#)
- [Install with docker](#)
- [Install Python interface with pip](#)

#### 1.1.1 Install off-line packages

Both CPU and GPU version offline packages are available in [the Releases page](#).

Some packages are split into two files due to size limit of GitHub. One may merge them into one after downloading:

```
cat deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh.0 deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh.  
↩1 > deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh
```

One may enable the environment using

```
conda activate /path/to/deepmd-kit
```

### 1.1.2 Install with conda

DeePMD-kit is available with [conda](#). Install [Anaconda](#) or [Miniconda](#) first.

#### Official channel

One may create an environment that contains the CPU version of DeePMD-kit and LAMMPS:

```
conda create -n deepmd deepmd-kit==*cpu libdeepmd==*cpu lammps -c https://conda.deepmodeling.com
↪ -c defaults
```

Or one may want to create a GPU environment containing [CUDA Toolkit](#):

```
conda create -n deepmd deepmd-kit==*gpu libdeepmd==*gpu lammps cudatoolkit=11.6 horovod -c
↪ https://conda.deepmodeling.com -c defaults
```

One could change the CUDA Toolkit version from 10.2 or 11.6.

One may specify the DeePMD-kit version such as 2.1.1 using

```
conda create -n deepmd deepmd-kit=2.1.1=*cpu libdeepmd=2.1.1=*cpu lammps horovod -c https://conda.
↪ deepmodeling.com -c defaults
```

One may enable the environment using

```
conda activate deepmd
```

#### conda-forge channel

DeePMD-kit is also available on the [conda-forge](#) channel:

```
conda create -n deepmd deepmd-kit lammps horovod -c conda-forge
```

The supported platform includes Linux x86-64, macOS x86-64, and macOS arm64. Read [conda-forge FAQ](#) to learn how to install CUDA-enabled packages.

### 1.1.3 Install with docker

A docker for installing the DeePMD-kit is available [here](#).

To pull the CPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.1.1_cpu
```

To pull the GPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.1.1_cuda11.6_gpu
```

To pull the ROCm version:

```
docker pull deepmodeling/dpmdkit-rocm:dp2.0.3-rocm4.5.2-tf2.6-lmp29Sep2021
```

### 1.1.4 Install Python interface with pip

If you have no existing TensorFlow installed, you can use `pip` to install the pre-built package of the Python interface with CUDA 12 supported:

```
pip install deepmd-kit[gpu,cu12]
```

`cu12` is required only when CUDA Toolkit and cuDNN were not installed.

To install the package built against CUDA 11.8, use

```
pip install deepmd-kit-cu11[gpu,cu11]
```

Or install the CPU version without CUDA supported:

```
pip install deepmd-kit[cpu]
```

The [LAMMPS module](#) and [the i-Pi driver](#) are only provided on Linux and macOS. To install LAMMPS and/or i-Pi, add `lmp` and/or `ipi` to extras:

```
pip install deepmd-kit[gpu,cu12,lmp,ipi]
```

MPICH is required for parallel running. (The macOS arm64 package doesn't support MPI yet.)

It is suggested to install the package into an isolated environment. The supported platform includes Linux x86-64 and aarch64 with GNU C Library 2.28 or above, macOS x86-64 and arm64, and Windows x86-64. A specific version of TensorFlow which is compatible with DeePMD-kit will be also installed.

Warning: If your platform is not supported, or want to build against the installed TensorFlow, or want to enable ROCM support, please build from source.

## 1.2 DeePMD-kit Quick Start Tutorial

DeePMD-kit is a deep learning package for many-body potential energy representation and molecular dynamics.

This tutorial can be directly run on Bohrium Notebook, you can click the `Open in Bohrium` button above to quickly run this document in Bohrium.

After opening Bohrium Notebook, click the button `connect`, and choose `deepmd-kit:2.2.1-cuda11.6-notebook` as image and `c4_m8_cpu` as computing resources. Wait a minute and you can get started.

### 1.2.1 Task

Mastering the paradigm cycle of using DeePMD-kit to establish deep potential molecular dynamics models, and following a complete case to learn how to apply it to molecular dynamics tasks.

By the end of this tutorial, you will be able to:

- Prepare the formataive dataset and running scripts for training with DeePMD-kit;
- Train, freeze, and test DeePMD-kit models;
- Use DeePMD-kit in LAMMPS for calculations;

Work through this tutorial. It will take you 20 minutes, max!

### 1.2.2 Table of contents

- Get tutorial data via git
- General Introduction
- Data preparation
- Prepare input script
- Train a model
- Freeze a model
- Test a model
- Run MD with LAMMPS

### 1.2.3 Get tutorial data via git

```
! if ! [ -e colombo-academy-tutorials ];then git clone https://gitee.com/deepmodeling/colombo-  
↪academy-tutorials.git;fi;
```

```
Cloning into 'colombo-academy-tutorials'...  
remote: Enumerating objects: 7164, done.  
remote: Counting objects: 100% (174/174), done.  
remote: Compressing objects: 100% (138/138), done.  
remote: Total 7164 (delta 78), reused 71 (delta 32), pack-reused 6990  
Receiving objects: 100% (7164/7164), 45.31 MiB | 3.85 MiB/s, done.  
Resolving deltas: 100% (3378/3378), done.  
Updating files: 100% (185/185), done.
```

### 1.2.4 General Introduction

This tutorial will introduce you to the basic usage of the DeePMD-kit, taking a gas phase methane molecule as an example. [DeePMD-kit's documentation](#) is recommended as the complete reference.

The DP model is generated using the DeePMD-kit package (v2.1.5). The training data is converted into the format of DeePMD-kit using a tool named dpdata (v0.2.14).

Details of dpdata can be found in [the dpdata documentation](#).



We've prepared initial data for  $CH_4$  for you, and put them in the folder `colombo-academy-tutorials/DeePMD-kit/00.data`

```
import os

prefix_path = os.getcwd()
```

Folder `abacus_md` is obtained by performing ab-initio molecular dynamics with ABACUS. Detailed instructions on ABACUS can be found in its [document](#).

```
os.chdir(
    os.path.join(prefix_path, "colombo-academy-tutorials", "DeePMD-kit", "00.data")
)
os.listdir("abacus_md")
```

```
['C_ONCV_PBE-1.2.upf',
 'C_gga_6au_100Ry_2s2p1d.orb',
 'H_ONCV_PBE-1.2.upf',
 'H_gga_6au_100Ry_2s1p.orb',
 'INPUT',
 'KPT',
 'OUT.ABACUS',
 'STRU']
```

## 1.2.5 Data preparation

The training data utilized by DeePMD-kit comprises essential information such as atom type, simulation box, atom coordinate, atom force, system energy, and virial. A snapshot of a molecular system that includes this data is called a **frame**. Multiple frames with the same number of atoms and atom types make up a **system** of data. For instance, a molecular dynamics trajectory can be converted into a system of data, with each time step corresponding to a frame in the system.

To simplify the process of converting data generated by popular simulation software like CP2K, Gaussian, Quantum-Espresso, ABACUS, and LAMMPS into the compressed format of DeePMD-kit, we offer a convenient tool called `dpdata`.

Next, the data from AIMD is splitted randomly as training and validation data.

```
import dpdata
import numpy as np

# load data of abacus/md format
data = dpdata.LabeledSystem("abacus_md", fmt="abacus/md")
print("# the data contains %d frames" % len(data))

# random choose 40 index for validation_data
rng = np.random.default_rng()
index_validation = rng.choice(201, size=40, replace=False)

# other indexes are training_data
index_training = list(set(range(201)) - set(index_validation))
data_training = data.sub_system(index_training)
data_validation = data.sub_system(index_validation)

# all training data put into directory:"training_data"
data_training.to_deepmd_npy("training_data")
```

(continues on next page)

(continued from previous page)

```
# all validation data put into directory:"validation_data"
data_validation.to_deepmd_npy("validation_data")

print("# the training data contains %d frames" % len(data_training))
print("# the validation data contains %d frames" % len(data_validation))
```

```
# the data contains 201 frames
# the training data contains 161 frames
# the validation data contains 40 frames
```

As you can see, 161 frames are picked as training data, and the other 40 frames are validation data.

The DeePMD-kit adopts a compressed data format. All training data should first be converted into this format and can then be used by DeePMD-kit. The data format is explained in detail in the DeePMD-kit manual that can be found in [the DeePMD-kit Data Introduction](#).

```
! tree training_data
```

```
training_data
  set.000
    box.npy
    coord.npy
    energy.npy
    force.npy
    virial.npy
  type.raw
  type_map.raw

1 directory, 7 files
```

Let's have a look at `type.raw`:

```
! cat training_data/type.raw
```

```
0
0
0
0
1
```

This tells us there are 5 atoms in this example, 4 atoms represented by type “0”, and 1 atom represented by type “1”. Sometimes one needs to map the integer types to atom name. The mapping can be given by the file `type_map.raw`

```
! cat training_data/type_map.raw
```

```
H
C
```

This tells us the type “0” is named by “H”, and the type “1” is named by “C”.

More detailed doc about Data conversion can be found [here](#).

### 1.2.6 Prepare input script

Once the data preparation is done, we can go on with training. Now go to the training directory

```
os.chdir(
    os.path.join(prefix_path, "colombo-academy-tutorials", "DeePMD-kit", "01.train")
)
```

DeePMD-kit requires a json format file to specify parameters for training.

In the model section, the parameters of embedding and fitting networks are specified.

```
"model":{
  "type_map":    ["H", "C"],
  "descriptor":{
    "type":      "se_e2_a",
    "rcut":      6.00,
    "rcut_smth": 0.50,
    "sel":       "auto",
    "neuron":     [25, 50, 100],
    "resnet_dt":  false,
    "axis_neuron": 16,
    "seed":      1,
    "_comment":   "that's all"
  },
  "fitting_net":{
    "neuron":     [240, 240, 240],
    "resnet_dt":  true,
    "seed":      1,
    "_comment":   "that's all"
  },
  "_comment":    "that's all"
},
```

The explanation for some of the parameters is as follows:

Parameter	Expiation
type_map	the name of each type of atom
descriptor > type	the type of descriptor
descriptor > rcut	cut-off radius
descriptor > rcut_smth	where the smoothing starts
descriptor > sel	the maximum number of type i atoms in the cut-off radius
descriptor > neuron	size of the embedding neural network
descriptor > axis_neuron	the size of the submatrix of G (embedding matrix)
fitting_net > neuron	size of the fitting neural network

The `se_e2_a` descriptor is used to train the DP model. The item neurons set the size of the descriptors and fitting network to [25, 50, 100] and [240, 240, 240], respectively. The components in local environment to smoothly go to zero from 0.5 to 6 Å.

The following are the parameters that specify the learning rate and loss function.

```
"learning_rate" :{
  "type":      "exp",
  "decay_steps": 50,
  "start_lr":  0.001,
```

(continues on next page)

(continued from previous page)

```

    "stop_lr":          3.51e-8,
    "_comment":         "that's all"
  },
  "loss" :{
    "type":             "ener",
    "start_pref_e":     0.02,
    "limit_pref_e":     1,
    "start_pref_f":     1000,
    "limit_pref_f":     1,
    "start_pref_v":     0,
    "limit_pref_v":     0,
    "_comment":         "that's all"
  },

```

In the loss function, `pref_e` increases from 0.02 to 1, and `pref_f` decreases from 1000 to 1 progressively, which means that the force term dominates at the beginning, while energy and virial terms become important at the end. This strategy is very effective and reduces the total training time. `pref_v` is set to 0, indicating that no virial data are included in the training process. The starting learning rate, stop learning rate, and decay steps are set to 0.001, 3.51e-8, and 50, respectively. The model is trained for 10000 steps.

The training parameters are given in the following

```

"training" : {
  "training_data": {
    "systems":          ["../00.data/training_data"],
    "batch_size":       "auto",
    "_comment":         "that's all"
  },
  "validation_data":{
    "systems":          ["../00.data/validation_data/"],
    "batch_size":       "auto",
    "numb_btch":        1,
    "_comment":         "that's all"
  },
  "numb_steps":        10000,
  "seed":               10,
  "disp_file":         "lcurve.out",
  "disp_freq":          200,
  "save_freq":          10000,
},

```

More detailed docs about Data conversion can be found [here](#).

## 1.2.7 Train a model

After the training script is prepared, we can start the training with DeePMD-kit by simply running

```
! dp train input.json
```

```

WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↳ compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope)
↳ is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by

```

(continues on next page)

(continued from previous page)

```

↳setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↳OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
WARNING:root:Environment variable KMP_BLOCKTIME is empty. Use the default value 0
WARNING:root:Environment variable KMP_AFFINITY is empty. Use the default value granularity=fine,
↳verbose,compact,1,0
/opt/deepmd-kit-2.2.1/lib/python3.10/importlib/__init__.py:169: UserWarning: The NumPy module was
↳reloaded (imported a second time). This can in some cases result in small but subtle issues and
↳is discouraged.
_bootstrap._exec(spec, module)
DEEPMO INFO    Calculate neighbor statistics... (add --skip-neighbor-stat to skip this step)
2023-04-20 23:35:59.335932: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could
↳not load dynamic library 'libcuda.so.1'; dlderror: libcuda.so.1: cannot open shared object file:
↳No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:35:59.335979: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to
↳cuInit: UNKNOWN ERROR (303)
OMP: Info #155: KMP_AFFINITY: Initial OS proc set respected: 0-7
OMP: Info #216: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #216: KMP_AFFINITY: cpuid leaf 11 not supported.
OMP: Info #216: KMP_AFFINITY: decoding legacy APIC ids.
OMP: Info #157: KMP_AFFINITY: 8 available OS procs
OMP: Info #158: KMP_AFFINITY: Uniform topology
OMP: Info #287: KMP_AFFINITY: topology layer "LL cache" is equivalent to "socket".
OMP: Info #192: KMP_AFFINITY: 1 socket x 4 cores/socket x 2 threads/core (4 total cores)
OMP: Info #218: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #172: KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 1 maps to socket 0 core 0 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 2 maps to socket 0 core 1 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 3 maps to socket 0 core 1 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 4 maps to socket 0 core 2 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 5 maps to socket 0 core 2 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 6 maps to socket 0 core 3 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 7 maps to socket 0 core 3 thread 1
OMP: Info #254: KMP_AFFINITY: pid 118 tid 140 thread 1 bound to OS proc set 2
OMP: Info #254: KMP_AFFINITY: pid 118 tid 142 thread 2 bound to OS proc set 4
OMP: Info #254: KMP_AFFINITY: pid 118 tid 144 thread 4 bound to OS proc set 1
OMP: Info #254: KMP_AFFINITY: pid 118 tid 143 thread 3 bound to OS proc set 6
OMP: Info #254: KMP_AFFINITY: pid 118 tid 145 thread 5 bound to OS proc set 3
OMP: Info #254: KMP_AFFINITY: pid 118 tid 146 thread 6 bound to OS proc set 5
OMP: Info #254: KMP_AFFINITY: pid 118 tid 147 thread 7 bound to OS proc set 7
OMP: Info #254: KMP_AFFINITY: pid 118 tid 148 thread 8 bound to OS proc set 0
OMP: Info #254: KMP_AFFINITY: pid 118 tid 139 thread 9 bound to OS proc set 2
OMP: Info #254: KMP_AFFINITY: pid 118 tid 149 thread 10 bound to OS proc set 4
OMP: Info #254: KMP_AFFINITY: pid 118 tid 150 thread 11 bound to OS proc set 6
OMP: Info #254: KMP_AFFINITY: pid 118 tid 151 thread 12 bound to OS proc set 1
OMP: Info #254: KMP_AFFINITY: pid 118 tid 152 thread 13 bound to OS proc set 3
OMP: Info #254: KMP_AFFINITY: pid 118 tid 153 thread 14 bound to OS proc set 5
OMP: Info #254: KMP_AFFINITY: pid 118 tid 154 thread 15 bound to OS proc set 7
OMP: Info #254: KMP_AFFINITY: pid 118 tid 155 thread 16 bound to OS proc set 0
DEEPMO INFO    training data with min nbor dist: 1.045920568611028
DEEPMO INFO    training data with max nbor size: [4 1]
DEEPMO INFO
DEEPMO INFO    |  _ _ \      |  _ _ \  |  \  |  _ _ \      |  _ _ \  |  _ _ \
DEEPMO INFO    |  |  |  |  _ _ \  |  _ _ \  |  \  |  _ _ \  |  _ _ \  |  _ _ \
DEEPMO INFO    |  |  |  |  _ _ \  |  _ _ \  |  \  |  _ _ \  |  _ _ \  |  _ _ \
DEEPMO INFO    |  |  |  |  _ _ \  |  _ _ \  |  \  |  _ _ \  |  _ _ \  |  _ _ \
DEEPMO INFO    |  |  |  |  _ _ \  |  _ _ \  |  \  |  _ _ \  |  _ _ \  |  _ _ \
DEEPMO INFO    |  |  |  |  _ _ \  |  _ _ \  |  \  |  _ _ \  |  _ _ \  |  _ _ \

```

(continues on next page)

(continued from previous page)

```

DEEPMO INFO Please read and cite:
DEEPMO INFO Wang, Zhang, Han and E, Comput.Phys.Comm. 228, 178-184 (2018)
DEEPMO INFO installed to: /home/conda/feedstock_root/build_artifacts/deepmd-kit_
↳1678943793317/work/_skbuild/linux-x86_64-3.10/cmake-install
DEEPMO INFO source : v2.2.1
DEEPMO INFO source brach: HEAD
DEEPMO INFO source commit: 3ac8c4c7
DEEPMO INFO source commit at: 2023-03-16 12:33:24 +0800
DEEPMO INFO build float prec: double
DEEPMO INFO build variant: cuda
DEEPMO INFO build with tf inc: /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/
↳include;/opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/../../../../include
DEEPMO INFO build with tf lib:
DEEPMO INFO ---Summary of the training-----
DEEPMO INFO running on: bohrium-14076-1013950
DEEPMO INFO computing device: cpu:0
DEEPMO INFO CUDA_VISIBLE_DEVICES: unset
DEEPMO INFO Count of visible GPU: 0
DEEPMO INFO num_intra_threads: 0
DEEPMO INFO num_inter_threads: 0
DEEPMO INFO -----
DEEPMO INFO ---Summary of DataSystem: training -----
↳--
DEEPMO INFO found 1 system(s):
DEEPMO INFO system natoms bch_sz n_bch prob pbc
DEEPMO INFO ../00.data/training_data 5 7 23 1.000 T
DEEPMO INFO -----
↳--
DEEPMO INFO ---Summary of DataSystem: validation -----
↳--
DEEPMO INFO found 1 system(s):
DEEPMO INFO system natoms bch_sz n_bch prob pbc
DEEPMO INFO ../00.data/validation_data 5 7 5 1.000 T
DEEPMO INFO -----
↳--
DEEPMO INFO training without frame parameter
DEEPMO INFO data stating... (this step may take long time)
OMP: Info #254: KMP_AFFINITY: pid 118 tid 118 thread 0 bound to OS proc set 0
DEEPMO INFO built lr
DEEPMO INFO built network
DEEPMO INFO built training
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by
↳setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↳OP_PARALLELISM_THREADS. See https://deepmd.rtd.io/parallelism/ for more information.
DEEPMO INFO initialize model from scratch
DEEPMO INFO start training at lr 1.00e-03 (== 1.00e-03), decay_step 50, decay_rate 0.950006,
↳final lr will be 3.51e-08
DEEPMO INFO batch 200 training time 6.10 s, testing time 0.02 s
DEEPMO INFO batch 400 training time 4.83 s, testing time 0.02 s
DEEPMO INFO batch 600 training time 4.84 s, testing time 0.02 s
DEEPMO INFO batch 800 training time 4.85 s, testing time 0.02 s
DEEPMO INFO batch 1000 training time 4.85 s, testing time 0.02 s
DEEPMO INFO saved checkpoint model.ckpt
DEEPMO INFO batch 1200 training time 4.86 s, testing time 0.02 s
DEEPMO INFO batch 1400 training time 5.39 s, testing time 0.02 s
DEEPMO INFO batch 1600 training time 4.84 s, testing time 0.02 s

```

(continues on next page)

(continued from previous page)

```

DEEPMd INFO    batch    1800 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    2000 training time 4.84 s, testing time 0.02 s
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    2200 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    2400 training time 4.90 s, testing time 0.02 s
DEEPMd INFO    batch    2600 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    2800 training time 4.84 s, testing time 0.02 s
DEEPMd INFO    batch    3000 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    3200 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    3400 training time 4.99 s, testing time 0.02 s
DEEPMd INFO    batch    3600 training time 4.88 s, testing time 0.02 s
DEEPMd INFO    batch    3800 training time 4.85 s, testing time 0.02 s
DEEPMd INFO    batch    4000 training time 4.88 s, testing time 0.02 s
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    4200 training time 4.88 s, testing time 0.02 s
DEEPMd INFO    batch    4400 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    4600 training time 4.92 s, testing time 0.02 s
DEEPMd INFO    batch    4800 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    5000 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    5200 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    5400 training time 4.88 s, testing time 0.02 s
DEEPMd INFO    batch    5600 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    5800 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    6000 training time 4.90 s, testing time 0.02 s
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↳ training/saver.py:1066: remove_checkpoint (from tensorflow.python.training.checkpoint_
↳ management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↳ training/saver.py:1066: remove_checkpoint (from tensorflow.python.training.checkpoint_
↳ management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to delete files with this prefix.
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    6200 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    6400 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    6600 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    6800 training time 4.84 s, testing time 0.02 s
DEEPMd INFO    batch    7000 training time 4.93 s, testing time 0.02 s
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    7200 training time 4.89 s, testing time 0.02 s
DEEPMd INFO    batch    7400 training time 4.88 s, testing time 0.02 s
DEEPMd INFO    batch    7600 training time 4.88 s, testing time 0.02 s
DEEPMd INFO    batch    7800 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    8000 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    8200 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    8400 training time 4.85 s, testing time 0.02 s
DEEPMd INFO    batch    8600 training time 4.86 s, testing time 0.02 s
DEEPMd INFO    batch    8800 training time 4.87 s, testing time 0.02 s
DEEPMd INFO    batch    9000 training time 4.83 s, testing time 0.02 s
DEEPMd INFO    saved checkpoint model.ckpt
DEEPMd INFO    batch    9200 training time 4.87 s, testing time 0.02 s

```

(continues on next page)

(continued from previous page)

```

DEEPMD INFO  batch    9400 training time 4.95 s, testing time 0.02 s
DEEPMD INFO  batch    9600 training time 4.89 s, testing time 0.02 s
DEEPMD INFO  batch    9800 training time 4.87 s, testing time 0.02 s
DEEPMD INFO  batch   10000 training time 4.88 s, testing time 0.02 s
DEEPMD INFO  saved checkpoint model.ckpt
DEEPMD INFO  average training time: 0.0244 s/batch (exclude first 200 batches)
DEEPMD INFO  finished training
DEEPMD INFO  wall time: 256.669 s

```

On the screen, you will see the information of the data system(s)

```

DEEPMD INFO  -----
DEEPMD INFO  ---Summary of DataSystem: training -----
DEEPMD INFO  found 1 system(s):
DEEPMD INFO                system  natoms  bch_sz  n_bch  prob  pbc
DEEPMD INFO  ../00.data/training_data    5      7    23  1.000  T
DEEPMD INFO  -----
DEEPMD INFO  ---Summary of DataSystem: validation -----
DEEPMD INFO  found 1 system(s):
DEEPMD INFO                system  natoms  bch_sz  n_bch  prob  pbc
DEEPMD INFO  ../00.data/validation_data    5      7     5  1.000  T
DEEPMD INFO  -----

```

and the starting and final learning rate of this training

```

DEEPMD INFO  start training at lr 1.00e-03 (== 1.00e-03), decay_step 50, decay_rate 0.950006,
↪ final lr will be 3.51e-08

```

If everything works fine, you will see, on the screen, information printed every 1000 steps, like

```

DEEPMD INFO  batch    200 training time 6.04 s, testing time 0.02 s
DEEPMD INFO  batch    400 training time 4.80 s, testing time 0.02 s
DEEPMD INFO  batch    600 training time 4.80 s, testing time 0.02 s
DEEPMD INFO  batch    800 training time 4.78 s, testing time 0.02 s
DEEPMD INFO  batch   1000 training time 4.77 s, testing time 0.02 s
DEEPMD INFO  saved checkpoint model.ckpt
DEEPMD INFO  batch   1200 training time 4.47 s, testing time 0.02 s
DEEPMD INFO  batch   1400 training time 4.49 s, testing time 0.02 s
DEEPMD INFO  batch   1600 training time 4.45 s, testing time 0.02 s
DEEPMD INFO  batch   1800 training time 4.44 s, testing time 0.02 s
DEEPMD INFO  batch   2000 training time 4.46 s, testing time 0.02 s
DEEPMD INFO  saved checkpoint model.ckpt

```

They present the training and testing time counts. At the end of the 1000th batch, the model is saved in Tensorflow's checkpoint file `model.ckpt`. At the same time, the training and testing errors are presented in file `lcurve.out`.

The file contains 8 columns, from left to right, are the training step, the validation loss, training loss, root mean square (RMS) validation error of energy, RMS training error of energy, RMS validation error of force, RMS training error of force and the learning rate. The RMS error (RMSE) of the energy is normalized by number of atoms in the system.

```

head -n 2 lcurve.out
#  step      rmse_val  rmse_trn  rmse_e_val  rmse_e_trn  rmse_f_val  rmse_f_trn  lr
    0      2.02e+01  1.51e+01  1.37e-01  1.41e-01  6.40e-01  4.79e-01  1.0e-03

```

and



```
$ tail -n 2 lcurve.out
9800      2.45e-02    4.02e-02    3.20e-04    3.88e-04    2.40e-02    3.94e-02    4.3e-08
10000     4.60e-02    3.76e-02    8.65e-04    5.35e-04    4.52e-02    3.69e-02    3.5e-08
```

Volumes 4, 5 and 6, 7 present energy and force training and testing errors, respectively.

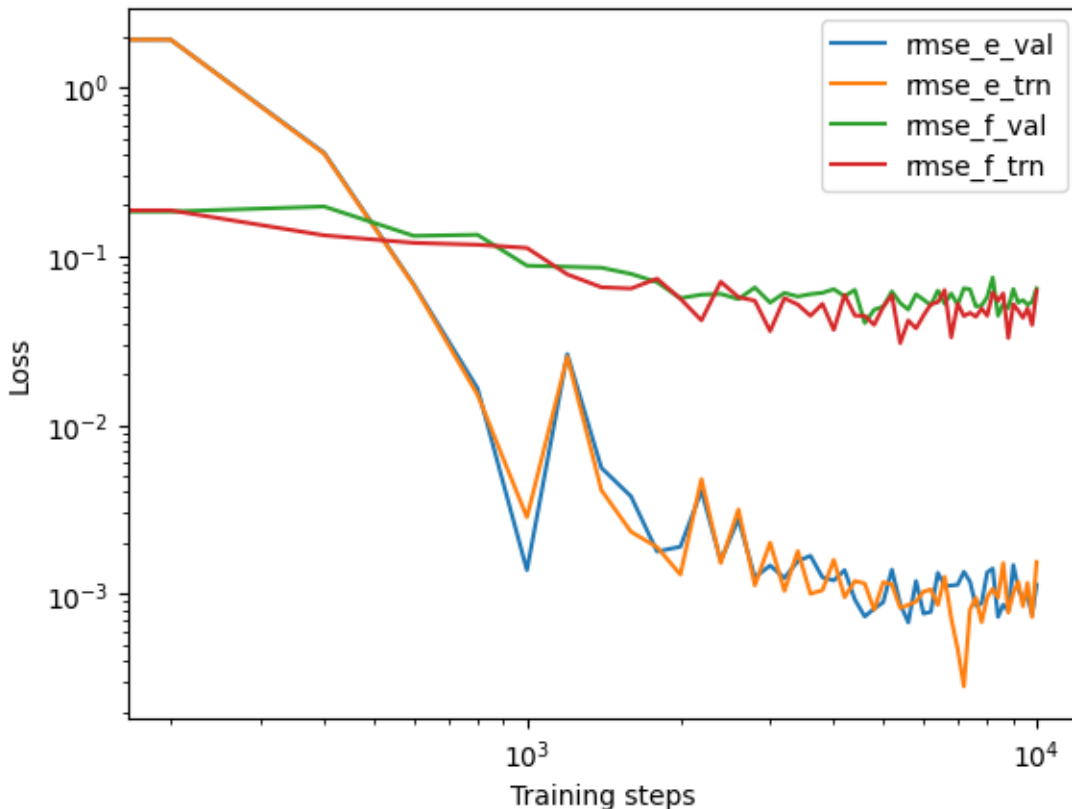
```
! head -n 2 lcurve.out && tail -n 2 lcurve.out
```

#	step	rmse_val	rmse_trn	rmse_e_val	rmse_e_trn	rmse_f_val	rmse_f_trn	lr
	0	2.06e+01	1.94e+01	1.34e-01	1.35e-01	6.51e-01	6.14e-01	1.0e-03
	9800	5.49e-02	4.00e-02	7.55e-04	7.28e-04	5.37e-02	3.91e-02	4.3e-08
	10000	6.56e-02	6.37e-02	1.13e-03	1.54e-03	6.44e-02	6.25e-02	3.5e-08

The loss function can be visualized to monitor the training process.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

with open("lcurve.out") as f:
    headers = f.readline().split()[1:]
lcurve = pd.DataFrame(np.loadtxt("lcurve.out"), columns=headers)
legends = ["rmse_e_val", "rmse_e_trn", "rmse_f_val", "rmse_f_trn"]
for legend in legends:
    plt.loglog(lcurve["step"], lcurve[legend], label=legend)
plt.legend()
plt.xlabel("Training steps")
plt.ylabel("Loss")
plt.show()
```



## 1.2.8 Freeze a model

At the end of the training, the model parameters saved in TensorFlow's checkpoint file should be frozen as a model file that is usually ended with extension `.pb`. Simply execute

```
! dp freeze -o graph.pb
```

```
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↳ compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope)
↳ is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by
↳ setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↳ OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
WARNING:root:Environment variable KMP_BLOCKTIME is empty. Use the default value 0
WARNING:root:Environment variable KMP_AFFINITY is empty. Use the default value granularity=fine,
↳ verbose,compact,1,0
/opt/deepmd-kit-2.2.1/lib/python3.10/importlib/__init__.py:169: UserWarning: The NumPy module was
↳ reloaded (imported a second time). This can in some cases result in small but subtle issues and
↳ is discouraged.
  _bootstrap._exec(spec, module)
2023-04-20 23:40:25.666203: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could
↳ not load dynamic library 'libcuda.so.1'; dLError: libcuda.so.1: cannot open shared object file:
↳ No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:25.666257: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to
```

(continues on next page)

(continued from previous page)

```

↪cuInit: UNKNOWN ERROR (303)
DEEPMO INFO    The following nodes will be frozen: ['model_type', 'descript_attr/rcut', 'descript_
↪attr/ntypes', 'model_attr/tmap', 'model_attr/model_type', 'model_attr/model_version', 'train_
↪attr/min_nbor_dist', 'train_attr/training_script', 'o_energy', 'o_force', 'o_virial', 'o_atom_
↪energy', 'o_atom_virial', 'fitting_attr/dfparam', 'fitting_attr/daparam']
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/deepmd/entrypoints/
↪freeze.py:354: convert_variables_to_constants (from tensorflow.python.framework.graph_util_impl)
↪is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.convert_variables_to_constants`
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/deepmd/entrypoints/
↪freeze.py:354: convert_variables_to_constants (from tensorflow.python.framework.graph_util_impl)
↪is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.convert_variables_to_constants`
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↪framework/convert_to_constants.py:925: extract_sub_graph (from tensorflow.python.framework.graph_
↪util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↪framework/convert_to_constants.py:925: extract_sub_graph (from tensorflow.python.framework.graph_
↪util_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use `tf.compat.v1.graph_util.extract_sub_graph`
DEEPMO INFO    1211 ops in the final graph.

```

and it will output a model file named `graph.pb` in the current directory.

## 1.2.9 Test a model

We can check the quality of the trained model by running

```
! dp test -m graph.pb -s ../00.data/validation_data
```

```

WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/tensorflow/python/
↪compat/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope)
↪is deprecated and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by
↪setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↪OP_PARALLELISM_THREADS. See https://deepmd.rtfd.io/parallelism/ for more information.
WARNING:root:Environment variable KMP_BLOCKTIME is empty. Use the default value 0
WARNING:root:Environment variable KMP_AFFINITY is empty. Use the default value granularity=fine,
↪verbose,compact,1,0
/opt/deepmd-kit-2.2.1/lib/python3.10/importlib/__init__.py:169: UserWarning: The NumPy module was
↪reloaded (imported a second time). This can in some cases result in small but subtle issues and
↪is discouraged.
  _bootstrap._exec(spec, module)
2023-04-20 23:40:30.102300: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could
↪not load dynamic library 'libcuda.so.1'; dLError: libcuda.so.1: cannot open shared object file:
↪No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:30.102346: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to
↪cuInit: UNKNOWN ERROR (303)

```

(continues on next page)

(continued from previous page)

```

WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/deepmd/utils/batch_size.
↳py:61: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be
↳removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
WARNING:tensorflow:From /opt/deepmd-kit-2.2.1/lib/python3.10/site-packages/deepmd/utils/batch_size.
↳py:61: is_gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be
↳removed in a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
DEEPMD WARNING You can use the environment variable DP_INFER_BATCH_SIZE to control the inference
↳batch size (nframes * natoms). The default value is 1024.
DEEPMD INFO      # -----output of dp test-----
DEEPMD INFO      # testing system : ../00.data/validation_data
OMP: Info #155: KMP_AFFINITY: Initial OS proc set respected: 0-7
OMP: Info #216: KMP_AFFINITY: decoding x2APIC ids.
OMP: Info #216: KMP_AFFINITY: cpuid leaf 11 not supported.
OMP: Info #216: KMP_AFFINITY: decoding legacy APIC ids.
OMP: Info #157: KMP_AFFINITY: 8 available OS procs
OMP: Info #158: KMP_AFFINITY: Uniform topology
OMP: Info #287: KMP_AFFINITY: topology layer "LL cache" is equivalent to "socket".
OMP: Info #192: KMP_AFFINITY: 1 socket x 4 cores/socket x 2 threads/core (4 total cores)
OMP: Info #218: KMP_AFFINITY: OS proc to physical thread map:
OMP: Info #172: KMP_AFFINITY: OS proc 0 maps to socket 0 core 0 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 1 maps to socket 0 core 0 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 2 maps to socket 0 core 1 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 3 maps to socket 0 core 1 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 4 maps to socket 0 core 2 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 5 maps to socket 0 core 2 thread 1
OMP: Info #172: KMP_AFFINITY: OS proc 6 maps to socket 0 core 3 thread 0
OMP: Info #172: KMP_AFFINITY: OS proc 7 maps to socket 0 core 3 thread 1
OMP: Info #254: KMP_AFFINITY: pid 254 tid 265 thread 1 bound to OS proc set 2
OMP: Info #254: KMP_AFFINITY: pid 254 tid 267 thread 2 bound to OS proc set 4
OMP: Info #254: KMP_AFFINITY: pid 254 tid 268 thread 3 bound to OS proc set 6
OMP: Info #254: KMP_AFFINITY: pid 254 tid 269 thread 4 bound to OS proc set 1
OMP: Info #254: KMP_AFFINITY: pid 254 tid 270 thread 5 bound to OS proc set 3
OMP: Info #254: KMP_AFFINITY: pid 254 tid 271 thread 6 bound to OS proc set 5
OMP: Info #254: KMP_AFFINITY: pid 254 tid 272 thread 7 bound to OS proc set 7
OMP: Info #254: KMP_AFFINITY: pid 254 tid 273 thread 8 bound to OS proc set 0
DEEPMD INFO      # number of test data : 40
DEEPMD INFO      Energy MAE           : 4.400922e-03 eV
DEEPMD INFO      Energy RMSE          : 5.258026e-03 eV
DEEPMD INFO      Energy MAE/Natoms    : 8.801843e-04 eV
DEEPMD INFO      Energy RMSE/Natoms   : 1.051605e-03 eV
DEEPMD INFO      Force  MAE           : 4.277741e-02 eV/A
DEEPMD INFO      Force  RMSE          : 5.514855e-02 eV/A
DEEPMD INFO      Virial  MAE           : 6.080471e-02 eV
DEEPMD INFO      Virial  RMSE          : 7.882116e-02 eV
DEEPMD INFO      Virial  MAE/Natoms   : 1.216094e-02 eV
DEEPMD INFO      Virial  RMSE/Natoms  : 1.576423e-02 eV
DEEPMD INFO      # -----

```

The correlation between predicted data and original data can also be calculated.

```

import dpdata

training_systems = dpdata.LabeledSystem("../00.data/training_data", fmt="deepmd/npz")

```

```
predict = training_systems.predict("graph.pb")
```

```
2023-04-20 23:40:32.104716: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow
↳ binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU
↳ instructions in performance-critical operations: AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-04-20 23:40:34.426193: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.
↳ cc:64] Could not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot open
↳ shared object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/
↳ nvidia/lib64
2023-04-20 23:40:34.427318: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.
↳ cc:64] Could not load dynamic library 'libnvinfer_plugin.so.7'; dlerror: libnvinfer_plugin.so.7:
↳ cannot open shared object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/
↳ lib:/usr/local/nvidia/lib64
2023-04-20 23:40:34.427332: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT
↳ Warning: Cannot dlopen some TensorRT libraries. If you would like to use Nvidia GPU with
↳ TensorRT, please make sure the missing libraries mentioned above are installed properly.
```

```
WARNING:tensorflow:From /opt/mamba/lib/python3.10/site-packages/tensorflow/python/compat/v2_compat.
↳ py:107: disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and
↳ will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

```
WARNING:root:To get the best performance, it is recommended to adjust the number of threads by
↳ setting the environment variables OMP_NUM_THREADS, TF_INTRA_OP_PARALLELISM_THREADS, and TF_INTER_
↳ OP_PARALLELISM_THREADS. See https://deepmd.rtd.io/parallelism/ for more information.
```

```
WARNING:tensorflow:From /opt/mamba/lib/python3.10/site-packages/deepmd/utils/batch_size.py:61: is_
↳ gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be removed in
↳ a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
```

```
2023-04-20 23:40:36.161142: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow
↳ binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU
↳ instructions in performance-critical operations: AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-04-20 23:40:36.165078: W tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.
↳ cc:64] Could not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared
↳ object file: No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/
↳ lib64
2023-04-20 23:40:36.165119: W tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:265]
↳ failed call to cuInit: UNKNOWN ERROR (303)
2023-04-20 23:40:36.165142: I tensorflow/compiler/xla/stream_executor/cuda/cuda_diagnostics.
↳ cc:156] kernel driver does not appear to be running on this host (bohrium-14076-1013950): /proc/
↳ driver/nvidia/version does not exist
2023-04-20 23:40:36.181810: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:357] MLIR
↳ V1 optimization pass is not enabled
WARNING:tensorflow:From /opt/mamba/lib/python3.10/site-packages/deepmd/utils/batch_size.py:61: is_
↳ gpu_available (from tensorflow.python.framework.test_util) is deprecated and will be removed in
↳ a future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
WARNING:deepmd.utils.batch_size:You can use the environment variable DP_INFER_BATCH_SIZE to control
↳ the inference batch size (nframes * natoms). The default value is 1024.
```

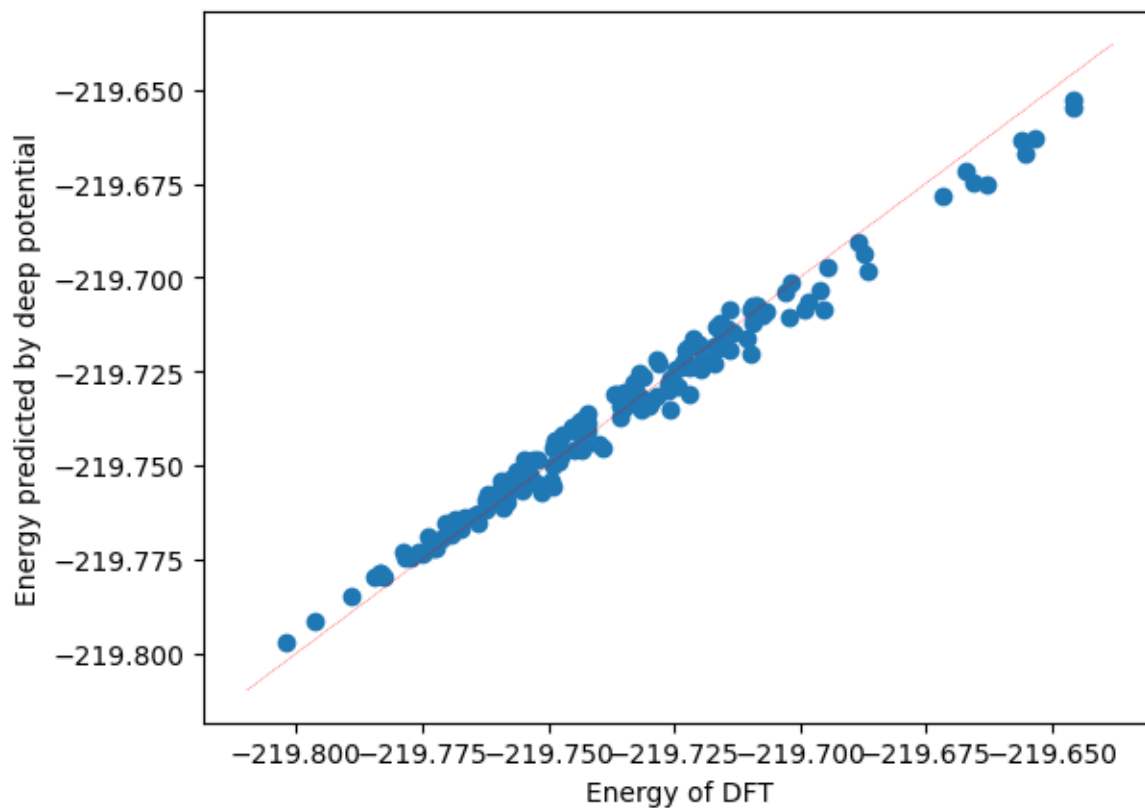
```
import matplotlib.pyplot as plt
import numpy as np

plt.scatter(training_systems["energies"], predict["energies"])

x_range = np.linspace(plt.xlim()[0], plt.xlim()[1])

plt.plot(x_range, x_range, "r--", linewidth=0.25)
plt.xlabel("Energy of DFT")
plt.ylabel("Energy predicted by deep potential")
plt.plot()
```

```
[]
```



### 1.2.10 Run MD with LAMMPS

The model can drive molecular dynamics in LAMMPS.

```
! cd ../02.lmp && cp ../01.train/graph.pb ./ && ls
```

```
conf.lmp graph.pb in.lammps
```

Here `conf.lmp` gives the initial configuration of a gas phase methane MD simulation, and the file `in.lammps` is the LAMMPS input script. One may check `in.lammps` and finds that it is a rather standard LAMMPS input file for a MD simulation, with only two exception lines:

```
pair_style deepmd graph.pb
pair_coeff * *
```

where the pair style deepmd is invoked and the model file `graph.pb` is provided, which means the atomic interaction will be computed by the DP model that is stored in the file `graph.pb`.

In an environment with a compatible version of LAMMPS, the deep potential molecular dynamics can be performed via

```
lmp -i input.lammps
```

```
! cd ../02.lmp && cp ../01.train/graph.pb ./ && lmp -i in.lammps
```

```
Warning:
This LAMMPS executable is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation.
LAMMPS (23 Jun 2022 - Update 1)
OMP_NUM_THREADS environment is not set. Defaulting to 1 thread. (src/comm.cpp:98)
  using 1 OpenMP thread(s) per MPI task
Loaded 1 plugins from /opt/deepmd-kit-2.2.1/lib/deepmd_lmp
Reading data file ...
  triclinic box = (0 0 0) to (10.114259 10.263124 10.216793) with tilt (0.036749877 0.13833062 -0.
↳ 056322169)
  1 by 1 by 1 MPI processor grid
  reading atoms ...
  5 atoms
  read_data CPU = 0.004 seconds
DeePMD-kit WARNING: Environmental variable OMP_NUM_THREADS is not set. Tune OMP_NUM_THREADS for
↳ the best performance. See https://deepmd.rtfd.io/parallelism/ for more information.
Summary of lammps deepmd module ...
  >>> Info of deepmd-kit:
    installed to:      /opt/deepmd-kit-2.2.1
    source:           v2.2.1
    source branch:    HEAD
    source commit:    3ac8c4c7
    source commit at: 2023-03-16 12:33:24 +0800
    support model ver.: 1.1
    build variant:    cuda
    build with tf inc: /opt/deepmd-kit-2.2.1/include;/opt/deepmd-kit-2.2.1/include
    build with tf lib: /opt/deepmd-kit-2.2.1/lib/libtensorflow_cc.so
    set tf intra_op_parallelism_threads: 0
    set tf inter_op_parallelism_threads: 0
  >>> Info of lammps module:
    use deepmd-kit at: /opt/deepmd-kit-2.2.1DeePMD-kit WARNING: Environmental variable OMP_NUM_
↳ THREADS is not set. Tune OMP_NUM_THREADS for the best performance. See https://deepmd.rtfd.io/
↳ parallelism/ for more information.
DeePMD-kit: Successfully load libcudart.so
2023-04-20 23:40:39.637091: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow
↳ binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU
↳ instructions in performance-critical operations: SSE4.1 SSE4.2 AVX AVX2 AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-04-20 23:40:39.643206: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could
↳ not load dynamic library 'libcudart.so.1'; dLError: libcudart.so.1: cannot open shared object file:
↳ No such file or directory; LD_LIBRARY_PATH: /usr/local/nvidia/lib:/usr/local/nvidia/lib64
2023-04-20 23:40:39.643234: W tensorflow/stream_executor/cuda/cuda_driver.cc:269] failed call to
↳ cuInit: UNKNOWN ERROR (303)
```

(continues on next page)

(continued from previous page)

```

2023-04-20 23:40:39.643257: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel_
↳driver does not appear to be running on this host (bohrium-14076-1013950): /proc/driver/nvidia/
↳version does not exist
2023-04-20 23:40:39.645305: I tensorflow/core/common_runtime/process_util.cc:146] Creating new_
↳thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best_
↳performance.
2023-04-20 23:40:39.700559: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:354] MLIR_
↳V1 optimization pass is not enabled
>>> Info of model(s):
using 1 model(s): graph.pb
rcut in model: 6
ntypes in model: 2

```

CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE

Your simulation uses code contributions which should be cited:

- USER-DEEPMO package:

The log file lists these citations in BibTeX format.

CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE-CITE

Generated 0 of 1 mixed pair\_coeff terms from geometric mixing rule

Neighbor list info ...

```

update every 10 steps, delay 0 steps, check no
max neighbors/atom: 2000, page size: 100000
master list distance cutoff = 7
ghost atom cutoff = 7
binsize = 3.5, bins = 3 3 3
1 neighbor lists, perpetual/occasional/extra = 1 0 0
(1) pair deepmd, perpetual
    attributes: full, newton on
    pair build: full/bin/atomonly
    stencil: full/bin/3d
    bin: standard

```

Setting up Verlet run ...

```

Unit style      : metal
Current step    : 0
Time step       : 0.001

```

Per MPI rank memory allocation (min/avg/max) = 3.809 | 3.809 | 3.809 Mbytes

Step	PotEng	KinEng	TotEng	Temp	Press	Volume
0	-219.77011	0.025852029	-219.74426	50	-810.10259	1060.5429
100	-219.76784	0.023303362	-219.74454	45.070664	-605.50113	1060.5429
200	-219.77863	0.032400378	-219.74622	62.665059	-53.929107	1060.5429
300	-219.77403	0.027115352	-219.74692	52.443373	642.24342	1060.5429
400	-219.77126	0.023079501	-219.74818	44.637697	861.365	1060.5429
500	-219.786	0.034433001	-219.75156	66.596322	256.47994	1060.5429
600	-219.78295	0.029039598	-219.75391	56.165027	-527.21506	1060.5429
700	-219.777	0.020227709	-219.75677	39.122091	-696.11258	1060.5429
800	-219.78394	0.022893217	-219.76105	44.277408	-77.227892	1060.5429
900	-219.77998	0.015506508	-219.76447	29.990893	663.84491	1060.5429
1000	-219.78328	0.015178419	-219.7681	29.356341	482.8228	1060.5429
1100	-219.7903	0.018763273	-219.77154	36.28975	-273.19351	1060.5429
1200	-219.78639	0.012922048	-219.77347	24.992328	-577.90459	1060.5429
1300	-219.79131	0.015848131	-219.77546	30.65162	-129.85247	1060.5429
1400	-219.78829	0.011969602	-219.77632	23.150218	545.58517	1060.5429
1500	-219.78735	0.010610097	-219.77674	20.520821	356.36805	1060.5429

(continues on next page)



(continued from previous page)

1600	-219.78834	0.011547453	-219.77679	22.333746	-386.08305	1060.5429
1700	-219.78549	0.0095297364	-219.77596	18.431312	-522.29867	1060.5429
1800	-219.78787	0.01302987	-219.77484	25.200865	120.83085	1060.5429
1900	-219.7845	0.012341623	-219.77216	23.869737	643.66442	1060.5429
2000	-219.7857	0.017597987	-219.76811	34.035987	255.57892	1060.5429
2100	-219.7853	0.023253088	-219.76205	44.973429	-465.61243	1060.5429
2200	-219.77987	0.024650089	-219.75522	47.675348	-708.62743	1060.5429
2300	-219.78134	0.030690759	-219.75065	59.358512	-221.82549	1060.5429
2400	-219.77737	0.029446857	-219.74792	56.952699	635.02431	1060.5429
2500	-219.768	0.022122766	-219.74587	42.787292	826.89652	1060.5429
2600	-219.77246	0.02691536	-219.74554	52.056572	168.88834	1060.5429
2700	-219.77746	0.031963987	-219.7455	61.821042	-497.33107	1060.5429
2800	-219.7733	0.02814671	-219.74515	54.438107	-792.71093	1060.5429
2900	-219.77498	0.029131114	-219.74585	56.342026	-685.23164	1060.5429
3000	-219.78212	0.034326288	-219.74779	66.38993	-20.441816	1060.5429
3100	-219.77222	0.02366469	-219.74856	45.769502	708.42782	1060.5429
3200	-219.77252	0.022334468	-219.75019	43.196742	753.3138	1060.5429
3300	-219.78538	0.032458098	-219.75292	62.776693	36.172647	1060.5429
3400	-219.78047	0.026131264	-219.75434	50.540064	-661.25487	1060.5429
3500	-219.77926	0.022926821	-219.75633	44.342401	-623.5037	1060.5429
3600	-219.78369	0.024854728	-219.75884	48.071137	74.821258	1060.5429
3700	-219.7768	0.016731114	-219.76006	32.359382	709.57785	1060.5429
3800	-219.77927	0.017595175	-219.76168	34.03055	543.56168	1060.5429
3900	-219.7864	0.023003584	-219.7634	44.490867	-230.55364	1060.5429
4000	-219.78098	0.017102387	-219.76388	33.077456	-677.85161	1060.5429
4100	-219.78581	0.020907229	-219.7649	40.436341	-343.9622	1060.5429
4200	-219.78717	0.021708329	-219.76546	41.985736	491.95578	1060.5429
4300	-219.78328	0.018229256	-219.76505	35.256916	680.5279	1060.5429
4400	-219.79007	0.024931071	-219.76514	48.21879	-26.785455	1060.5429
4500	-219.78331	0.019795452	-219.76352	38.286071	-624.98799	1060.5429
4600	-219.78094	0.0196038	-219.76134	37.915399	-584.8297	1060.5429
4700	-219.78608	0.027516802	-219.75857	53.219812	74.218844	1060.5429
4800	-219.77656	0.023488867	-219.75307	45.429446	827.4406	1060.5429
4900	-219.78039	0.032832529	-219.74755	63.500874	634.64896	1060.5429
5000	-219.78237	0.040761952	-219.7416	78.837046	-224.81626	1060.5429

Loop time of 12.1251 on 1 procs for 5000 steps with 5 atoms

Performance: 35.629 ns/day, 0.674 hours/ns, 412.369 timesteps/s  
242.0% CPU use with 1 MPI tasks x 1 OpenMP threads

MPI task timing breakdown:

Section	min time	avg time	max time	varavg	%total
Pair	12.072	12.072	12.072	0.0	99.56
Neigh	0.0066181	0.0066181	0.0066181	0.0	0.05
Comm	0.012792	0.012792	0.012792	0.0	0.11
Output	0.0044695	0.0044695	0.0044695	0.0	0.04
Modify	0.022737	0.022737	0.022737	0.0	0.19
Other		0.006263			0.05

Nlocal:                    5 ave                    5 max                    5 min  
Histogram: 1 0 0 0 0 0 0 0 0  
Nghost:                    130 ave                    130 max                    130 min  
Histogram: 1 0 0 0 0 0 0 0 0  
Neighs:                    0 ave                    0 max                    0 min  
Histogram: 1 0 0 0 0 0 0 0 0

(continues on next page)

(continued from previous page)

```
FullNghs:          20 ave          20 max          20 min
Histogram: 1 0 0 0 0 0 0 0 0 0
```

```
Total # of neighbors = 20
Ave neighs/atom = 4
Neighbor list builds = 500
Dangerous builds not checked
Total wall time: 0:00:13
```

## INSTALLATION

### 2.1 Easy install

There are various easy methods to install DeePMD-kit. Choose one that you prefer. If you want to build by yourself, jump to the next two sections.

After your easy installation, DeePMD-kit (`dp`) and LAMMPS (`lmp`) will be available to execute. You can try `dp -h` and `lmp -h` to see the help. `mpirun` is also available considering you may want to train models or run LAMMPS in parallel.

---

Note: Note: The off-line packages and conda packages require the [GNU C Library 2.17](#) or above. The GPU version requires [compatible NVIDIA driver](#) to be installed in advance. It is possible to force conda to [override detection](#) when installation, but these requirements are still necessary during runtime.

---

- [Install off-line packages](#)
- [Install with conda](#)
- [Install with docker](#)
- [Install Python interface with pip](#)

#### 2.1.1 Install off-line packages

Both CPU and GPU version offline packages are available in [the Releases page](#).

Some packages are splited into two files due to size limit of GitHub. One may merge them into one after downloading:

```
cat deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh.0 deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh.  
↵1 > deepmd-kit-2.1.1-cuda11.6_gpu-Linux-x86_64.sh
```

One may enable the environment using

```
conda activate /path/to/deepmd-kit
```

### 2.1.2 Install with conda

DeePMD-kit is available with [conda](#). Install [Anaconda](#) or [Miniconda](#) first.

#### Official channel

One may create an environment that contains the CPU version of DeePMD-kit and LAMMPS:

```
conda create -n deepmd deepmd-kit==*cpu libdeepmd==*cpu lammps -c https://conda.deepmodeling.com
↪ -c defaults
```

Or one may want to create a GPU environment containing [CUDA Toolkit](#):

```
conda create -n deepmd deepmd-kit==*gpu libdeepmd==*gpu lammps cudatoolkit=11.6 horovod -c
↪ https://conda.deepmodeling.com -c defaults
```

One could change the CUDA Toolkit version from 10.2 or 11.6.

One may specify the DeePMD-kit version such as 2.1.1 using

```
conda create -n deepmd deepmd-kit=2.1.1=*cpu libdeepmd=2.1.1=*cpu lammps horovod -c https://conda.
↪ deepmodeling.com -c defaults
```

One may enable the environment using

```
conda activate deepmd
```

#### conda-forge channel

DeePMD-kit is also available on the [conda-forge](#) channel:

```
conda create -n deepmd deepmd-kit lammps horovod -c conda-forge
```

The supported platform includes Linux x86-64, macOS x86-64, and macOS arm64. Read [conda-forge FAQ](#) to learn how to install CUDA-enabled packages.

### 2.1.3 Install with docker

A docker for installing the DeePMD-kit is available [here](#).

To pull the CPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.1.1_cpu
```

To pull the GPU version:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:2.1.1_cuda11.6_gpu
```

To pull the ROCm version:

```
docker pull deepmodeling/dpmdkit-rocm:dp2.0.3-rocm4.5.2-tf2.6-lmp29Sep2021
```

### 2.1.4 Install Python interface with pip

If you have no existing TensorFlow installed, you can use `pip` to install the pre-built package of the Python interface with CUDA 12 supported:

```
pip install deepmd-kit[gpu,cu12]
```

`cu12` is required only when CUDA Toolkit and cuDNN were not installed.

To install the package built against CUDA 11.8, use

```
pip install deepmd-kit-cu11[gpu,cu11]
```

Or install the CPU version without CUDA supported:

```
pip install deepmd-kit[cpu]
```

The [LAMMPS module](#) and the [i-Pi driver](#) are only provided on Linux and macOS. To install LAMMPS and/or i-Pi, add `lmp` and/or `ipi` to extras:

```
pip install deepmd-kit[gpu,cu12,lmp,ipi]
```

MPICH is required for parallel running. (The macOS arm64 package doesn't support MPI yet.)

It is suggested to install the package into an isolated environment. The supported platform includes Linux x86-64 and aarch64 with GNU C Library 2.28 or above, macOS x86-64 and arm64, and Windows x86-64. A specific version of TensorFlow which is compatible with DeePMD-kit will be also installed.

Warning: If your platform is not supported, or want to build against the installed TensorFlow, or want to enable ROCM support, please [build from source](#).

## 2.2 Install from source code

Please follow our [GitHub](#) webpage to download the [latest released version](#) and [development version](#).

Or get the DeePMD-kit source code by `git clone`

```
cd /some/workspace
git clone https://github.com/deepmodeling/deepmd-kit.git deepmd-kit
```

For convenience, you may want to record the location of the source to a variable, saying `deepmd_source_dir` by

```
cd deepmd-kit
deepmd_source_dir=`pwd`
```

## 2.2.1 Install the python interface

### Install Tensorflow's python interface

First, check the python version on your machine

```
python --version
```

We follow the virtual environment approach to install TensorFlow's Python interface. The full instruction can be found on the official [TensorFlow website](#). TensorFlow 1.8 or later is supported. Now we assume that the Python interface will be installed to the virtual environment directory `$tensorflow_venv`

```
virtualenv -p python3 $tensorflow_venv
source $tensorflow_venv/bin/activate
pip install --upgrade pip
pip install --upgrade tensorflow
```

It is important that every time a new shell is started and one wants to use DeePMD-kit, the virtual environment should be activated by

```
source $tensorflow_venv/bin/activate
```

if one wants to skip out of the virtual environment, he/she can do

```
deactivate
```

If one has multiple python interpreters named something like `python3.x`, it can be specified by, for example

```
virtualenv -p python3.8 $tensorflow_venv
```

If one does not need the GPU support of DeePMD-kit and is concerned about package size, the CPU-only version of TensorFlow should be installed by

```
pip install --upgrade tensorflow-cpu
```

To verify the installation, run

```
python -c "import tensorflow as tf;print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

One should remember to activate the virtual environment every time he/she uses DeePMD-kit.

One can also [build the TensorFlow Python interface from source](#) for custom hardware optimization, such as CUDA, ROCM, or OneDNN support.

### Install the DeePMD-kit's python interface

Check the compiler version on your machine

```
gcc --version
```

The compiler GCC 4.8 or later is supported in the DeePMD-kit. Note that TensorFlow may have specific requirements for the compiler version to support the C++ standard version and `_GLIBCXX_USE_CXX11_ABI` used by TensorFlow. It is recommended to use [the same compiler version as TensorFlow](#), which can be printed by `python -c "import tensorflow;print(tensorflow.version.COMPILER_VERSION)"`.

Execute

```
cd $deepmd_source_dir
pip install .
```

One may set the following environment variables before executing `pip`:

Environment variables	Allowed value	Default value	Usage
DP_VARIANT	cpu, cuda, rocm	cpu	Build CPU variant or GPU variant with CUDA or ROCM support.
CUDA-Toolkit_ROOT	Path	Detected automatically	The path to the CUDA toolkit directory. CUDA 9.0 or later is supported. NVCC is required.
ROCM_ROOT	Path	Detected automatically	The path to the ROCM toolkit directory.
TENSORFLOW_ROOT	Path	Detected automatically	The path to TensorFlow Python library. By default the installer only finds TensorFlow under user site-package directory ( <code>site.getusersitepackages()</code> ) or system site-package directory ( <code>sysconfig.get_path("purelib")</code> ) due to limitation of <a href="#">PEP-517</a> . If not found, the latest TensorFlow (or the environment variable <code>TENSORFLOW_VERSION</code> if given) from PyPI will be built against.
DP_ENABLE_NATIVE_OPTIMIZATION	YES/NO	YES	Enable compilation optimization for the native machine's CPU type. Do not enable it if generated code will run on different CPUs.
CMAKE_ARGS	Flags	-	Additional CMake arguments
<LANG>FLAGS (<LANG>=CXX, CUDA or HIP)	Flags	-	Default compilation flags to be used when compiling <LANG> files. See <a href="#">CMake documentation</a> .

To test the installation, one should first jump out of the source directory

```
cd /some/other/workspace
```

then execute

```
dp -h
```

It will print the help information like

```
usage: dp [-h] {train,freeze,test} ...
```

```
DeePMD-kit: A deep learning package for many-body potential energy
```

(continues on next page)

(continued from previous page)

```

representation and molecular dynamics

optional arguments:
  -h, --help            show this help message and exit

Valid subcommands:
  {train,freeze,test}
    train                train a model
    freeze               freeze the model
    test                 test the model

```

## Install horovod and mpi4py

Horovod and mpi4py are used for parallel training. For better performance on GPU, please follow the tuning steps in [Horovod on GPU](#).

```

# With GPU, prefer NCCL as a communicator.
HOROVOD_WITHOUT_GLOO=1 HOROVOD_WITH_TENSORFLOW=1 HOROVOD_GPU_OPERATIONS=NCCL HOROVOD_NCCL_HOME=/
↪path/to/nccl pip install horovod mpi4py

```

If your work in a CPU environment, please prepare runtime as below:

```

# By default, MPI is used as communicator.
HOROVOD_WITHOUT_GLOO=1 HOROVOD_WITH_TENSORFLOW=1 pip install horovod mpi4py

```

To ensure Horovod has been built with proper framework support enabled, one can invoke the `horovodrun --check-build` command, e.g.,

```

$ horovodrun --check-build

Horovod v0.22.1:

Available Frameworks:
  [X] TensorFlow
  [X] PyTorch
  [ ] MXNet

Available Controllers:
  [X] MPI
  [X] Gloo

Available Tensor Operations:
  [X] NCCL
  [ ] DDL
  [ ] CCL
  [X] MPI
  [X] Gloo

```

Since version 2.0.1, Horovod and mpi4py with MPICH support are shipped with the installer.

If you don't install Horovod, DeePMD-kit will fall back to serial mode.



## 2.2.2 Install the C++ interface

If one does not need to use DeePMD-kit with Lammmps or I-Pi, then the python interface installed in the previous section does everything and he/she can safely skip this section.

### Install Tensorflow's C++ interface (optional)

Since TensorFlow 2.12, TensorFlow C++ library (`libtensorflow_cc`) is packaged inside the Python library. Thus, you can skip building TensorFlow C++ library manually. If that does not work for you, you can still build it manually.

The C++ interface of DeePMD-kit was tested with compiler GCC  $\geq 4.8$ . It is noticed that the I-Pi support is only compiled with GCC  $\geq 4.8$ . Note that TensorFlow may have specific requirements for the compiler version.

First, the C++ interface of Tensorflow should be installed. It is noted that the version of Tensorflow should be consistent with the python interface. You may follow the instruction or run the script `$deepmd_source_dir/source/install/build_tf.py` to install the corresponding C++ interface.

### Install DeePMD-kit's C++ interface

Now go to the source code directory of DeePMD-kit and make a building place.

```
cd $deepmd_source_dir/source
mkdir build
cd build
```

The installation requires CMake 3.16 or later for the CPU version, CMake 3.23 or later for the CUDA support, and CMake 3.21 or later for the ROCm support. One can install CMake via `pip` if it is not installed or the installed version does not satisfy the requirement:

```
pip install -U cmake
```

I assume you have activated the TensorFlow Python environment and want to install DeePMD-kit into path `$deepmd_root`, then execute CMake

```
cmake -DUSE_TF_PYTHON_LIBS=TRUE -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
```

If you specify `-DUSE_TF_PYTHON_LIBS=FALSE`, you need to give the location where TensorFlow's C++ interface is installed to `-DTENSORFLOW_ROOT=${tensorflow_root}`.

One may add the following arguments to `cmake`:

CMake Arguments	Allowed value	Default value	Usage
- DTENSORFLOW_ROOT	Path =<value>	-	The Path to TensorFlow's C++ interface.
- DCMAKE_INSTALL_PREFIX	Path =<value>	-	The Path where DeePMD-kit will be installed.
- DUSE_CUDA_TOOLKIT	TRUE or FALSE =<value>	FALSE	If TRUE, Build GPU support with CUDA toolkit.
- DCUDAToolkit_ROOT	Path =<value>	De- tected auto- mati- cally	The path to the CUDA toolkit directory. CUDA 9.0 or later is supported. NVCC is required.
- DUSE_ROCM_TOOLKIT	TRUE or FALSE =<value>	FALSE	If TRUE, Build GPU support with ROCM toolkit.
- DCMAKE_HIP_COMPILER_ROOT	Path =<value>	De- tected auto- mati- cally	The path to the ROCM toolkit directory.
- DLAMMPS_SOURCE_ROOT	Path =<value>	-	Only necessary for LAMMPS plugin mode. The path to the <a href="#">LAMMPS source code</a> . LAMMPS 8Apr2021 or later is supported. If not assigned, the plugin mode will not be enabled.
- DUSE_TF_PYTHON_LIBS	TRUE or FALSE =<value>	FALSE	If TRUE, Build C++ interface with TensorFlow's Python libraries(TensorFlow's Python Interface is required). And there's no need for building TensorFlow's C++ interface.
- DENABLE_NATIVE_OPTIMIZATION	TRUE or FALSE =<value>	FALSE	Enable compilation optimization for the native machine's CPU type. Do not enable it if generated code will run on different CPUs.
- DCMAKE_<LANG>_FLAGS (<LANG>=CXX, CUDA or HIP)	str =<value>	-	Default compilation flags to be used when compiling <LANG> files. See <a href="#">CMake documentation</a> .

If the CMake has been executed successfully, then run the following make commands to build the package:

```
make -j4
make install
```

Option -j4 means using 4 processes in parallel. You may want to use a different number according to your hardware.

If everything works fine, you will have the executable and libraries installed in \$deepmd\_root/bin and \$deepmd\_root/lib

```
$ ls $deepmd_root/bin
$ ls $deepmd_root/lib
```

## 2.3 Install from pre-compiled C library

DeePMD-kit provides pre-compiled C library package (`libdeepmd_c.tar.gz`) in each [release](#). It can be used to build the [LAMMPS plugin](#) and [GROMACS patch](#), as well as many [third-party software packages](#), without building TensorFlow and DeePMD-kit on one's own. It can be downloaded via the shell command:

```
wget https://github.com/deepmodeling/deepmd-kit/releases/latest/download/libdeepmd_c.tar.gz
tar xzf libdeepmd_c.tar.gz
```

The library is built in Linux (GLIBC 2.17) with CUDA 12.2 (`libdeepmd_c.tar.gz`) or 11.8 (`libdeepmd_c_cu11.tar.gz`). It's noted that this package does not contain CUDA Toolkit and cuDNN, so one needs to download them from the NVIDIA website.

### 2.3.1 Use Pre-compiled C Library to build the LAMMPS plugin and GROMACS patch

When one [installs DeePMD-kit's C++ interface](#), one can use the CMake argument `DEEPMDC_ROOT` to the path `libdeepmd_c`.

```
cd $deepmd_source_dir/source
mkdir build
cd build
cmake -DDEEPMDC_ROOT=/path/to/libdeepmd_c -DCMAKE_INSTALL_PREFIX=$deepmd_root ..
make -j8
make install
```

Then one can follow the manual [Install LAMMPS](#) and/or [Install GROMACS](#).

## 2.4 Install LAMMPS

There are two ways to install LAMMPS: the built-in mode and the plugin mode. The built-in mode builds LAMMPS along with the DeePMD-kit and DeePMD-kit will be loaded automatically when running LAMMPS. The plugin mode builds LAMMPS and a plugin separately, so one needs to use `plugin load` command to load the DeePMD-kit's LAMMPS plugin library.

### 2.4.1 Install LAMMPS's DeePMD-kit module (built-in mode)

Before following this section, [DeePMD-kit C++ interface](#) should have been installed.

DeePMD-kit provides a module for running MD simulations with LAMMPS. Now make the DeePMD-kit module for LAMMPS.

```
cd $deepmd_source_dir/source/build
make lammps
```

DeePMD-kit will generate a module called `USER-DEEPMDC` in the `build` directory, which supports either double or single float precision interface. Now download the LAMMPS code, and uncompress it.

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_2Aug2023_update2.tar.gz
tar xf stable_2Aug2023_update2.tar.gz
```

The source code of LAMMPS is stored in the directory `lammps-stable_2Aug2023_update2`.

Then, you can build LAMMPS with either `make` or `CMake`.

### With `make`

Now go into the LAMMPS code and copy the DeePMD-kit module like this

```
cd lammps-stable_2Aug2023_update2/src/  
cp -r $deepmd_source_dir/source/build/USER-DEEPMD .  
make yes-kSPACE  
make yes-extra-fix  
make yes-user-deepmd
```

You can enable any other package you want. Now build LAMMPS

```
make mpi -j4
```

If everything works fine, you will end up with an executable `lmp_mpi`.

```
./lmp_mpi -h
```

The DeePMD-kit module can be removed from the LAMMPS source code by

```
make no-user-deepmd
```

### With `CMake`

Now go into the LAMMPS directory and create a directory called `build`:

```
mkdir -p lammps-stable_2Aug2023_update2/build/  
cd lammps-stable_2Aug2023_update2/build/
```

Patch the LAMMPS `CMakeLists.txt` file:

```
echo "include($deepmd_source_dir/source/lmp/builtin.cmake)" >> ../cmake/CMakeLists.txt
```

It's expected to see one extra line in the end of `CMakeLists.txt`.

Now build LAMMPS. You can install any other package you want.

```
cmake -D LAMMPS_INSTALL_RPATH=ON -D BUILD_SHARED_LIBS=yes -D CMAKE_INSTALL_PREFIX=$deepmd_root -  
↪DCMAKE_PREFIX_PATH=$deepmd_root ../cmake  
make -j4  
make install
```

If everything works fine, you will end up with an executable `$deepmd_root/bin/lmp`.

```
$deepmd_root/bin/lmp -h
```

## 2.4.2 Install LAMMPS (plugin mode)

Starting from 8Apr2021, LAMMPS also provides a plugin mode, allowing one to build LAMMPS and a plugin separately.

Now download the LAMMPS code (8Apr2021 or later), and uncompress it:

```
cd /some/workspace
wget https://github.com/lammps/lammps/archive/stable_2Aug2023_update2.tar.gz
tar xf stable_2Aug2023_update2.tar.gz
```

The source code of LAMMPS is stored in the directory `lammps-stable_2Aug2023_update2`. The directory of the source code should be specified as the CMAKE argument `LAMMPS_SOURCE_ROOT` during installation of the DeePMD-kit C++ interface. Now go into the LAMMPS directory and create a directory called `build`

```
mkdir -p lammps-stable_2Aug2023_update2/build/
cd lammps-stable_2Aug2023_update2/build/
```

Now build LAMMPS. Note that `PLUGIN` must be enabled, and `BUILD_SHARED_LIBS` must be set to `yes`. You can install any other package you want.

```
cmake -D PKG_PLUGIN=ON -D LAMMPS_INSTALL_RPATH=ON -D BUILD_SHARED_LIBS=yes -D CMAKE_INSTALL_PREFIX=
↳ ${deepmd_root} -D CMAKE_INSTALL_LIBDIR=lib -D CMAKE_INSTALL_FULL_LIBDIR=${deepmd_root}/lib ../
↳ cmake
make -j4
make install
```

If everything works fine, you will end up with an executable `${deepmd_root}/bin/lmp`.

```
${deepmd_root}/bin/lmp -h
```

Note: If `${tensorflow_root}`, `${deepmd_root}`, or the path to TensorFlow Python package if applicable is different from the prefix of LAMMPS, you need to append the library path to `RUNPATH` of `liblammps.so`. For example, use `patchelf >= 0.13`

```
patchelf --add-rpath "${tensorflow_root}/lib" liblammps.so
```

## 2.5 Install i-PI

The i-PI works in a client-server model. The i-PI provides the server for integrating the replica positions of atoms, while the DeePMD-kit provides a client named `dp_ipi` that computes the interactions (including energy, forces and virials). The server and client communicate via the Unix domain socket or the Internet socket. Full documentation for i-PI can be found [here](#). The source code and a complete installation guide for i-PI can be found [here](#). To use i-PI with already existing drivers, install and update using Pip:

```
pip install -U i-PI
```

Test with Pytest:

```
pip install pytest
pytest --pyargs ipi.tests
```

## 2.6 Install GROMACS with DeepMD

Before following this section, DeePMD-kit C++ interface should have be installed.

### 2.6.1 Patch source code of GROMACS

Download the source code of a supported GROMACS version (2020.2) from <https://manual.gromacs.org/2020.2/download.html>. Run the following command:

```
export PATH=$PATH:$deepmd_kit_root/bin
dp_gmx_patch -d $gromacs_root -v $version -p
```

where `deepmd_kit_root` is the directory where the latest version of DeePMD-kit is installed, and `gromacs_root` refers to the source code directory of GROMACS. And `version` represents the version of GROMACS, where only 2020.2 is supported now. If attempting to patch another version of GROMACS you will still need to set `version` to 2020.2 as this is the only supported version, we cannot guarantee that patching other versions of GROMACS will work.

### 2.6.2 Compile GROMACS with deepmd-kit

The C++ interface of Deepmd-kit 2.x and TensorFlow 2.x are required. And be aware that only DeePMD-kit with high precision is supported now since we cannot ensure single precision is enough for a GROMACS simulation. Here is a sample compile script:

```
#!/bin/bash
export CC=/usr/bin/gcc
export CXX=/usr/bin/g++
export CMAKE_PREFIX_PATH="/path/to/fftw-3.3.9" # fftw libraries
mkdir build
cd build

cmake3 .. -DCMAKE_CXX_STANDARD=14 \ # not required, but c++14 seems to be more compatible with
↪ higher version of tensorflow
    -DGMX_MPI=ON \
    -DGMX_GPU=CUDA \ # Gromacs on ROCm has not been fully developed yet
    -DCUDAToolkit_ROOT=/path/to/cuda \
    -DCMAKE_INSTALL_PREFIX=/path/to/gromacs-2020.2-deepmd
make -j
make install
```

## 2.7 Building conda packages

One may want to keep both convenience and personalization of the DeePMD-kit. To achieve this goal, one can consider building conda packages. We provide building scripts in [deepmd-kit-recipes](#) organization. These building tools are driven by [conda-build](#) and [conda-smithy](#).

For example, if one wants to turn on MPIIO package in LAMMPS, go to [lammps-feedstock](#) repository and modify `recipe/build.sh`. `-D PKG_MPIIO=OFF` should be changed to `-D PKG_MPIIO=ON`. Then go to the main directory and execute

```
./build-locally.py
```

This requires that Docker has been installed. After the building, the packages will be generated in `build_artifacts/linux-64` and `build_artifacts/noarch`, and then one can install then executing

```
conda create -n deepmd lammps -c file:///path/to/build_artifacts -c https://conda.deepmodeling.com
↪ -c nvidia
```

One may also upload packages to one's Anaconda channel, so they can be installed on other machines:

```
anaconda upload /path/to/build_artifacts/linux-64/*.tar.bz2 /path/to/build_artifacts/noarch/*.tar.
↪ bz2
```

## 2.8 Install Node.js interface

DeePMD-kit has an inference interface for Node.js, the most common programming language in the world, via a wrapper of the header-only C++ interface created by SWIG.

### 2.8.1 Install from npm

```
npm i deepmd-kit
# Or if you want to install globally
npm i -g deepmd-kit
```

### 2.8.2 Build from source

Before building DeePMD-kit, install [Node.js](#), [SWIG](#) (v4.1.0 for Node.js v12-v18 support), and [node-gyp](#) globally.

When using CMake to [build DeePMD-kit from source](#), set argument `BUILD_NODEJS_IF=ON` and `NODEJS_INCLUDE_DIRS=/path/to/nodejs/include` (the path to the include directory of Node.js):

```
cmake -D BUILD_NODEJS_IF=ON \
      -D NODEJS_INCLUDE_DIRS=/path/to/nodejs/include \
      .. # and other arguments
make
make install
```

After installing DeePMD-kit, two files, `bind.gyp` and `deepmdJAVASCRIPT_wrap.cxx` will be generated in `$deepmd_source_dir/source/nodejs`.

Go to this directory, and install the Node.js package globally:

```
cd $deepmd_source_dir/source/nodejs
npm i
npm link
```

The `deepmd-kit` package should be globally available in Node.js environments:

```
const deepmd = require("deepmd-kit");
```

## 2.9 Easy install the latest development version

DeePMD-kit is actively developed in the `devel` branch. The documentation of the `latest` version matches the `devel` branch.

The following is the way to install the pre-compiled packages without [building from source](#). All of them are built with [GitHub Actions](#).

### 2.9.1 Install with docker

The `devel` tag is used to mark the latest development version with CUDA 12.2 support:

```
docker pull ghcr.io/deepmodeling/deepmd-kit:devel
```

For CUDA 11.8 support, use the `devel_cu11` tag.

### 2.9.2 Install with pip

Below is an one-line shell command to download the [artifact](#) containing wheels and install it with `pip`:

```
pip install -U --pre deepmd-kit[gpu,cu12,lmf] --extra-index-url https://deepmodeling.github.io/  
↳deepmd-kit/simple
```

`cu12` and `lmf` are optional, which is the same as the stable version.

### 2.9.3 Download pre-compiled C Library

The [pre-compiled C library](#) can be downloaded from [here](#), or via a shell command:

```
wget https://nightly.link/deepmodeling/deepmd-kit/workflows/package_c/devel/libdeepmd_c-0-  
↳libdeepmd_c.tar.gz.zip && unzip libdeepmd_c-0-libdeepmd_c.tar.gz.zip
```



## DATA

In this section, we will introduce how to convert the DFT-labeled data into the data format used by DeePMD-kit.

The DeePMD-kit organizes data in **systems**. Each **system** is composed of a number of **frames**. One may roughly view a **frame** as a snapshot of an MD trajectory, but it does not necessarily come from an MD simulation. A **frame** records the coordinates and types of atoms, cell vectors if the periodic boundary condition is assumed, energy, atomic forces and virials. It is noted that the **frames** in one **system** share the same number of atoms with the same type.

### 3.1 System

DeePMD-kit takes a system as the data structure. A snapshot of a system is called a frame. A system may contain multiple frames with the same atom types and numbers, i.e. the same formula (like H<sub>2</sub>O). To contains data with different formulas, one usually needs to divide data into multiple systems, which may sometimes result in sparse-frame systems. See a [new system format](#) to further combine different systems with the same atom numbers, when training with descriptor `se_atten`.

A system should contain system properties, input frame properties, and labeled frame properties. The system property contains the following property:

ID	Property	Raw file	Required/Optional	Shape	Description
type	Atom type indexes	type.raw	Required	Natoms	Integers that start with 0. If both the training parameter <code>type_map</code> is set and <code>type_map.raw</code> is provided, the system atom type should be mapped to <code>type_map.raw</code> in <code>type.raw</code> and will be mapped to the model atom type when training; otherwise, the system atom type will be always mapped to the model atom type (whether <code>type_map</code> is set or not)
type_map	Atom type names	type_map.raw	Optional	Ntypes	Atom names that map to atom type, which is unnecessary to be contained in the periodic table. Only works when the training parameter <code>type_map</code> is set
nopbc	Non-periodic system	nopbc	Optional	1	If True, this system is non-periodic; otherwise it's periodic

The input frame properties contain the following property, the first axis of which is the number of frames:

ID	Property	Raw file	Unit	Re-quired/Optional	Shape	Description
coord	Atomic coordinates	co-ord.raw	Å	Required	Nframes * Natoms * 3	
box	Boxes	box.raw	Å	Required if periodic	Nframes * 3 * 3	in the order XX XY XZ YX YY YZ ZX ZY ZZ
fparam	Extra frame parameters	fparam.raw	Any	Optional	Nframes * Any	
aparam	Extra atomic parameters	aparam.raw	Any	Optional	Nframes * aparam * Any	
numb_copy	Each frame is copied by the numb_copy (int) times	prob.raw	1	Optional	Nframes	Integer; Default is 1 for all frames

The labeled frame properties are listed as follows, all of which will be used for training if and only if the loss function contains such property:

ID	Property	Raw file	Unit	Shape	Description
energy	Frame energies	en-ergy.raw	eV	Nframes	
force	Atomic forces	force.raw	eV/Å	Nframes * Natoms * 3	
virial	Frame virial	virial.raw	eV	Nframes * 9	in the order XX XY XZ YX YY YZ ZX ZY ZZ
atom_ener	Atomic energies	atom_ener.raw	eV	Nframes * Natoms	
atom_pref	Weights of atomic forces	atom_pref.raw	1	Nframes * Natoms	
dipole	Frame dipole	dipole.raw	Any	Nframes * 3	
atomic_dipole	Atomic dipole	atomic_dipole.raw	Any	Nframes * Natoms * 3	
polarizability	Frame polarizability	polar-izabil-ity.raw	Any	Nframes * 9	in the order XX XY XZ YX YY YZ ZX ZY ZZ
atomic_polarizability	Atomic polarizability	atomic_polarizability.raw	Any	Nframes * Natoms * 9	in the order XX XY XZ YX YY YZ ZX ZY ZZ
drdq	Partial derivative of atomic coordinates with respect to generalized coordinates	drdq.raw	1	Nframes * Natoms * 3 * Ngen_coords	

In general, we always use the following convention of units:

Property	Unit
Time	ps
Length	Å
Energy	eV
Force	eV/Å
Virial	eV
Pressure	Bar

## 3.2 Formats of a system

Two binary formats, NumPy and HDF5, are supported for training. The raw format is not directly supported, but a tool is provided to convert data from the raw format to the NumPy format.

### 3.2.1 NumPy format

In a system with the Numpy format, the system properties are stored as text files ending with `.raw`, such as `type.raw` and `type_map.raw`, under the system directory. If one needs to train a non-periodic system, an empty `nopbc` file should be put under the system directory. Both input and labeled frame properties are saved as the [NumPy binary data \(NPY\) files](#) ending with `.numpy` in each of the `set.*` directories. Take an example, a system may contain the following files:

```
type.raw
type_map.raw
nopbc
set.000/coord.npy
set.000/energy.npy
set.000/force.npy
set.001/coord.npy
set.001/energy.npy
set.001/force.npy
```

We assume that the atom types do not change in all frames. It is provided by `type.raw`, which has one line with the types of atoms written one by one. The atom types should be integers. For example the `type.raw` of a system that has 2 atoms with 0 and 1:

```
$ cat type.raw
0 1
```

Sometimes one needs to map the integer types to atom names. The mapping can be given by the file `type_map.raw`. For example

```
$ cat type_map.raw
0 H
```

The type 0 is named by "0" and the type 1 is named by "H".

For training models with descriptor `se_atten`, a [new system format](#) is supported to put together the frame-sparse systems with the same atom number.

### 3.2.2 HDF5 format

A system with the HDF5 format has the same structure as the Numpy format, but in an HDF5 file, a system is organized as an [HDF5 group](#). The file name of a Numpy file is the key in an HDF5 file, and the data is the value of the key. One needs to use `#` in a DP path to divide the path to the HDF5 file and the HDF5 path:

```
/path/to/data.hdf5#/H20
```

Here, `/path/to/data.hdf5` is the file path and `/H20` is the HDF5 path. All HDF5 paths should start with `/`. There should be some data in the H20 group, such as `/H20/type.raw` and `/H20/set.000/force.npy`.

An HDF5 file with a large number of systems has better performance than multiple NumPy files in a large cluster.

### 3.2.3 Raw format and data conversion

A raw file is a plain text file with each information item written in one file and one frame written on one line. It's not directly supported, but we provide a tool to convert them.

In the raw format, the property of one frame is provided per line, ending with `.raw`. Take an example, the default files that provide box, coordinate, force, energy and virial are `box.raw`, `coord.raw`, `force.raw`, `energy.raw` and `virial.raw`, respectively. Here is an example of `force.raw`:

```
$ cat force.raw
-0.724  2.039 -0.951  0.841 -0.464  0.363
 6.737  1.554 -5.587 -2.803  0.062  2.222
-1.968 -0.163  1.020 -0.225 -0.789  0.343
```

This `force.raw` contains 3 frames with each frame having the forces of 2 atoms, thus it has 3 lines and 6 columns. Each line provides all the 3 force components of 2 atoms in 1 frame. The first three numbers are the 3 force components of the first atom, while the second three numbers are the 3 force components of the second atom. Other files are organized similarly. The number of lines of all raw files should be identical.

One can use the script `$deepmd_source_dir/data/raw/raw_to_set.sh` to convert the prepared raw files to the NumPy format. For example, if we have a raw file that contains 6000 frames,

```
$ ls
box.raw coord.raw energy.raw force.raw type.raw virial.raw
$ $deepmd_source_dir/data/raw/raw_to_set.sh 2000
nframe is 6000
nline per set is 2000
will make 3 sets
making set 0 ...
making set 1 ...
making set 2 ...
$ ls
box.raw coord.raw energy.raw force.raw set.000 set.001 set.002 type.raw virial.raw
```

It generates three sets `set.000`, `set.001` and `set.002`, with each set containing 2000 frames in the Numpy format.

## 3.3 Prepare data with dpdata

One can use a convenient tool `dpdata` to convert data directly from the output of first principle packages to the DeePMD-kit format.

To install one can execute

```
pip install dpdata
```

An example of converting data [VASP](#) data in OUTCAR format to DeePMD-kit data can be found at

```
$deepmd_source_dir/examples/data_conv
```

Switch to that directory, then one can convert data by using the following python script

```
import dpdata

dsys = dpdata.LabeledSystem("OUTCAR")
dsys.to("deepmd/npz", "deepmd_data", set_size=dsys.get_nframes())
```

`get_nframes()` method gets the number of frames in the OUTCAR, and the argument `set_size` enforces that the set size is equal to the number of frames in the system, viz. only one `set` is created in the `system`.

The data in DeePMD-kit format is stored in the folder `deepmd_data`.

A list of all [supported data format](#) and more nice features of `dpdata` can be found on the [official website](#).



## 4.1 Overall

### 4.1.1 Theory

A Deep Potential (DP) model, denoted by  $\mathcal{M}$ , can be generally represented as

$$\mathbf{y}_i = \mathcal{M}(\mathbf{x}_i, \{\mathbf{x}_j\}_{j \in n(i)}; \boldsymbol{\theta}) = \mathcal{F}(\mathcal{D}(\mathbf{x}_i, \{\mathbf{x}_j\}_{j \in n(i)}; \boldsymbol{\theta}_d); \boldsymbol{\theta}_f),$$

where  $\mathbf{y}_i$  is the fitting properties,  $\mathcal{F}$  is the fitting network,  $\mathcal{D}$  is the descriptor.  $\mathbf{x} = (\mathbf{r}_i, \alpha_i)$ , with  $\mathbf{r}_i$  being the Cartesian coordinates and  $\alpha_i$  being the chemical species, denotes the degrees of freedom of the atom  $i$ .

The indices of the neighboring atoms (i.e. atoms within a certain cutoff radius) of atom  $i$  are given by the notation  $n(i)$ . Note that the Cartesian coordinates can be either under the periodic boundary condition (PBC) or in vacuum (under the open boundary condition). The network parameters are denoted by  $\boldsymbol{\theta} = \{\boldsymbol{\theta}_d, \boldsymbol{\theta}_f\}$ , where  $\boldsymbol{\theta}_d$  and  $\boldsymbol{\theta}_f$  yield the network parameters of the descriptor (if any) and those of the fitting network, respectively. From the above equation, one may compute the global property of the system by

$$\mathbf{y} = \sum_{i=1}^N \mathbf{y}_i,$$

where  $N$  is the number of atoms in a frame. For example, if  $y_i$  represents the potential energy contribution of atom  $i$ , then  $y$  gives the total potential energy of the frame.<sup>1</sup>

### 4.1.2 Instructions

A model has two parts, a descriptor that maps atomic configuration to a set of symmetry invariant features, and a fitting net that takes descriptor as input and predicts the atomic contribution to the target physical property. It's defined in the `model` section of the `input.json`, for example,

```
"model": {
  "type_map":      ["O", "H"],
  "descriptor": {
    "...": "..."
  },
}
```

(continues on next page)

---

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

(continued from previous page)

```

    "fitting_net" : {
        "...": "..."
    }
}

```

The two subsections, `descriptor` and `fitting_net`, define the descriptor and the fitting net, respectively.

The `type_map` is optional, which provides the element names (but not necessarily same as the actual name of the element) of the corresponding atom types. A water model, as in this example, has two kinds of atoms. The atom types are internally recorded as integers, e.g., 0 for oxygen and 1 for hydrogen here. A mapping from the atom type to their names is provided by `type_map`.

DeePMD-kit implements the following descriptors:

1. `se_e2_a`: DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.
2. `se_e2_r`: DeepPot-SE constructed from radial information of atomic configurations. The embedding takes the distance between atoms as input.
3. `se_e3`: DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes angles between two neighboring atoms as input.
4. `se_a_mask`: DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The input frames in one system can have a varied number of atoms. Input particles are padded with virtual particles of the same length.
5. `loc_frame`: Defines a local frame at each atom and compute the descriptor as local coordinates under this frame.
6. `hybrid`: Concatenate a list of descriptors to form a new descriptor.

The fitting of the following physical properties is supported

1. `ener`: Fit the energy of the system. The force (derivative with atom positions) and the virial (derivative with the box tensor) can also be trained.
2. `dipole`: The dipole moment.
3. `polar`: The polarizability.

## 4.2 Descriptor "se\_e2\_a"

The notation of `se_e2_a` is short for the Deep Potential Smooth Edition (DeepPot-SE) constructed from all information (both angular and radial) of atomic configurations. The `e2` stands for the embedding with two-atoms information. This descriptor was described in detail in [the DeepPot-SE paper](#).

Note that it is sometimes called a “two-atom embedding descriptor” which means the input of the embedding net is atomic distances. The descriptor does encode multi-body information (both angular and radial information of neighboring atoms).



### 4.2.1 Theory

The two-body embedding smooth edition of the DP descriptor  $\mathcal{D}^i \in \mathbb{R}^{M \times M_{<}}$ , is usually named DeepPot-SE descriptor. It is noted that the descriptor is a multi-body representation of the local environment of the atom  $i$ . We call it two-body embedding because the embedding network takes only the distance between atoms  $i$  and  $j$  (see below), but it is not implied that the descriptor takes only the pairwise information between  $i$  and its neighbors. The descriptor, using full information, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2} (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i,$$

where  $N_c$  is the expected maximum number of neighboring atoms, which is the same constant for all atoms over all frames. A matrix with a dimension of  $N_c$  will be padded if the number of neighboring atoms is less than  $N_c$ .  $\mathcal{R}^i \in \mathbb{R}^{N_c \times 4}$  is the coordinate matrix, and each row of  $\mathcal{R}^i$  can be constructed as

$$(\mathcal{R}^i)_j = \left\{ s(r_{ij}) \quad \frac{s(r_{ij})x_{ij}}{r_{ij}} \quad \frac{s(r_{ij})y_{ij}}{r_{ij}} \quad \frac{s(r_{ij})z_{ij}}{r_{ij}} \right\},$$

where  $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i = (x_{ij}, y_{ij}, z_{ij})$  is the relative coordinate and  $r_{ij} = \|\mathbf{r}_{ij}\|$  is its norm. The switching function  $s(r)$  is defined as

$$s(r) = \begin{cases} \frac{1}{r}, & rr_s, \\ \frac{1}{r} [x^3(-6x^2 + 15x - 10) + 1], & r_s \leq rr_c, \\ 0, & r \geq r_c, \end{cases}$$

where  $x = \frac{r-r_s}{r_c-r_s}$  switches from 1 at  $r_s$  to 0 at the cutoff radius  $r_c$ . The switching function  $s(r)$  is smooth in the sense that the second-order derivative is continuous.

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N_c \times M}$  consists of  $M$  nodes from the output layer of an NN function  $\mathcal{N}_g$  of  $s(r_{ij})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(s(r_{ij})),$$

where the subscript  $e,2$  is used to distinguish the NN from other NNs used in the DP model. In the above equation, the network parameters are not explicitly written.  $\mathcal{G}_{<}^i \in \mathbb{R}^{N_c \times M_{<}}$  only takes first  $M_{<}$  columns of  $\mathcal{G}^i$  to reduce the size of  $\mathcal{D}^i$ .  $r_s, r_c, M$  and  $M_{<}$  are hyperparameters provided by the user. The DeepPot-SE is continuous up to the second-order derivative in its domain.<sup>1</sup>

### 4.2.2 Instructions

In this example, we will train a DeepPot-SE model for a water system. A complete training input script of this example can be found in the directory.

```
$deepmd_source_dir/examples/water/se_e2_a/input.json
```

With the training input script, data are also provided in the example directory. One may train the model with the DeePMD-kit from the directory.

The construction of the descriptor is given by section descriptor. An example of the descriptor is provided as follows

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

```

"descriptor" :{
  "type":          "se_e2_a",
  "rcut_smth":     0.50,
  "rcut":          6.00,
  "sel":           [46, 92],
  "neuron":        [25, 50, 100],
  "type_one_side": true,
  "axis_neuron":   16,
  "resnet_dt":     false,
  "seed":          1
}

```

- The type of the descriptor is set to "se\_e2\_a".
- rcut is the cut-off radius for neighbor searching, and the rcut\_smth gives where the smoothing starts.
- sel gives the maximum possible number of neighbors in the cut-off radius. It is a list, the length of which is the same as the number of atom types in the system, and sel[i] denotes the maximum possible number of neighbors with type i.
- The neuron specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a [ResNet architecture](#) is built between them.
- If the option type\_one\_side is set to true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{types}}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{types}}^2$  sets of embedding network parameters.
- The axis\_neuron specifies the size of the submatrix of the embedding matrix, the axis matrix as explained in the [DeepPot-SE paper](#)
- If the option resnet\_dt is set to true, then a timestep is used in the ResNet.
- seed gives the random seed that is used to generate random numbers when initializing the model parameters.

## 4.3 Descriptor "se\_e2\_r"

The notation of se\_e2\_r is short for the Deep Potential Smooth Edition (DeepPot-SE) constructed from the radial information of atomic configurations. The e2 stands for the embedding with two-atom information.

### 4.3.1 Theory

The descriptor, using either radial-only information, is given by

$$\mathcal{D}^i = \frac{1}{N_c} \sum_j (\mathcal{G}^i)_{jk},$$

where  $N_c$  is the expected maximum number of neighboring atoms, which is the same constant for all atoms over all frames. A matrix with a dimension of  $N_c$  will be padded if the number of neighboring atoms is less than  $N_c$ .

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N_c \times M}$  consists of  $M$  nodes from the output layer of an NN function  $\mathcal{N}_g$  of  $s(r_{ij})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(s(r_{ij})),$$

where  $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i = (x_{ij}, y_{ij}, z_{ij})$  is the relative coordinate and  $r_{ij} = \|\mathbf{r}_{ij}\|$  is its norm. The switching function  $s(r)$  is defined as

$$s(r) = \begin{cases} \frac{1}{r}, & rr_s, \\ \frac{1}{r} [x^3(-6x^2 + 15x - 10) + 1], & r_s \leq rr_c, \\ 0, & r \geq r_c, \end{cases}$$

where  $x = \frac{r-r_s}{r_c-r_s}$  switches from 1 at  $r_s$  to 0 at the cutoff radius  $r_c$ . The switching function  $s(r)$  is smooth in the sense that the second-order derivative is continuous.

In the above equations, the network parameters are not explicitly written.  $r_s, r_c$  and  $M$  are hyperparameters provided by the user. The DeepPot-SE is continuous up to the second-order derivative in its domain.<sup>1</sup>

### 4.3.2 Instructions

A complete training input script of this example can be found in the directory

```
$deepmd_source_dir/examples/water/se_e2_r/input.json
```

The training input script is very similar to that of `se_e2_a`. The only difference lies in the descriptor section

```
"descriptor": {
  "type":          "se_e2_r",
  "sel":           [46, 92],
  "rcut_smth":     0.50,
  "rcut":          6.00,
  "neuron":        [5, 10, 20],
  "resnet_dt":     false,
  "seed":          1,
  "_comment": " that's all"
},
```

The type of the descriptor is set by the key type.

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

## 4.4 Descriptor "se\_e3"

The notation of `se_e3` is short for the Deep Potential Smooth Edition (DeepPot-SE) constructed from all information (both angular and radial) of atomic configurations. The embedding takes bond angles between a central atom and its two neighboring atoms as input (denoted by `e3`).

### 4.4.1 Theory

The three-body embedding DeepPot-SE descriptor incorporates bond-angle information, making the model more accurate. The descriptor  $\mathcal{D}^i$  can be represented as

$$\mathcal{D}^i = \frac{1}{N_c^2} (\mathcal{R}^i (\mathcal{R}^i)^T) : \mathcal{G}^i,$$

where  $N_c$  is the expected maximum number of neighboring atoms, which is the same constant for all atoms over all frames.  $\mathcal{R}^i$  is constructed as

$$(\mathcal{R}^i)_j = \left\{ s(r_{ij}) \quad \frac{s(r_{ij})x_{ij}}{r_{ij}} \quad \frac{s(r_{ij})y_{ij}}{r_{ij}} \quad \frac{s(r_{ij})z_{ij}}{r_{ij}} \right\},$$

Currently, only the full information case of  $\mathcal{R}^i$  is supported by the three-body embedding. Each element of  $\mathcal{G}^i \in \mathbb{R}^{N_c \times N_c \times M}$  comes from  $M$  nodes from the output layer of an NN  $\mathcal{N}_{e,3}$  function:

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}((\theta_i)_{jk}),$$

where  $(\theta_i)_{jk} = (\mathcal{R}^i)_{j,2,3,4} \cdot (\mathcal{R}^i)_{k,2,3,4}$  considers the angle form of two neighbours ( $j$  and  $k$ ). The notation  $:$  in the equation indicates the contraction between matrix  $\mathcal{R}^i (\mathcal{R}^i)^T$  and the first two dimensions of tensor  $\mathcal{G}^i$ .<sup>1</sup>

### 4.4.2 Instructions

A complete training input script of this example can be found in the directory

```
$deepmd_source_dir/examples/water/se_e3/input.json
```

The training input script is very similar to that of `se_e2_a`. The only difference lies in the `descriptor` <model/descriptor> section

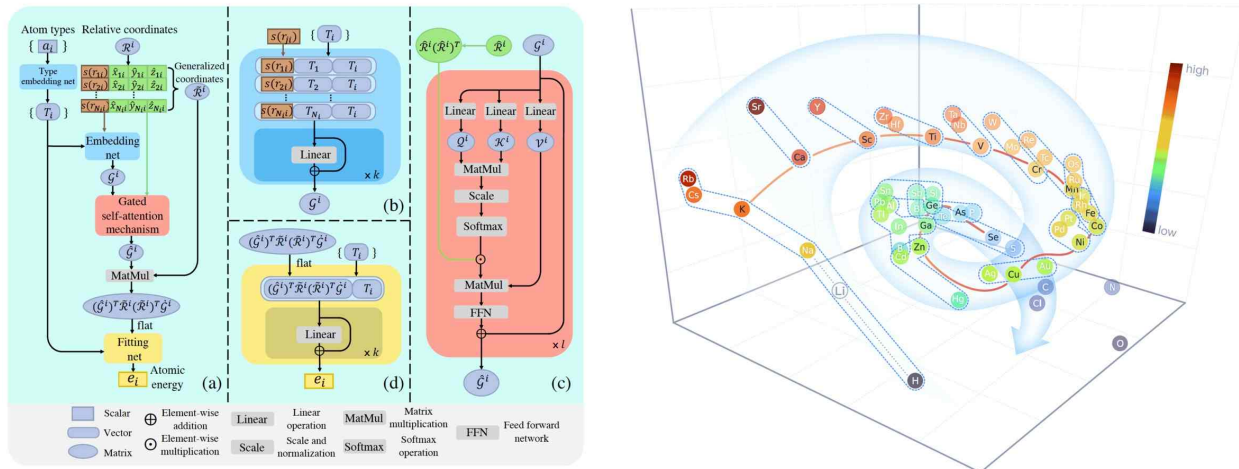
```
"descriptor": {
  "type":          "se_e3",
  "sel":           [40, 80],
  "rcut_smth":     0.50,
  "rcut":          6.00,
  "neuron":        [2, 4, 8],
  "resnet_dt":     false,
  "seed":          1,
  "_comment":      " that's all"
},
```

The type of the descriptor is set by the key type.

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

## 4.5 Descriptor "se\_atten"

### 4.5.1 DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation



Here we propose DPA-1, a Deep Potential model with a novel attention mechanism, which is highly effective for representing the conformation and chemical spaces of atomic systems and learning the PES.

See [this paper](#) for more information. DPA-1 is implemented as a new descriptor "se\_atten" for model training, which can be used after simply editing the input.json.

### 4.5.2 Theory

Attention-based descriptor  $\mathcal{D}^i \in \mathbb{R}^{M \times M_c}$ , which is proposed in pretrainable DPA-1 model, is given by

$$\mathcal{D}^i = \frac{1}{N_c^2} (\hat{\mathcal{G}}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \hat{\mathcal{G}}^i,$$

where  $\hat{\mathcal{G}}^i$  represents the embedding matrix  $\mathcal{G}^i$  after additional self-attention mechanism and  $\mathcal{R}^i$  is defined by the full case in the `se_e2_a`. Note that we obtain  $\mathcal{G}^i$  using the type embedding method by default in this descriptor.

To perform the self-attention mechanism, the queries  $\mathcal{Q}^{i,l} \in \mathbb{R}^{N_c \times d_k}$ , keys  $\mathcal{K}^{i,l} \in \mathbb{R}^{N_c \times d_k}$ , and values  $\mathcal{V}^{i,l} \in \mathbb{R}^{N_c \times d_v}$  are first obtained:

$$\begin{aligned} (\mathcal{Q}^{i,l})_j &= Q_l \left( (\mathcal{G}^{i,l-1})_j \right), \\ (\mathcal{K}^{i,l})_j &= K_l \left( (\mathcal{G}^{i,l-1})_j \right), \\ (\mathcal{V}^{i,l})_j &= V_l \left( (\mathcal{G}^{i,l-1})_j \right), \end{aligned}$$

where  $Q_l, K_l, V_l$  represent three trainable linear transformations that output the queries and keys of dimension  $d_k$  and values of dimension  $d_v$ , and  $l$  is the index of the attention layer. The input embedding matrix to the attention layers, denoted by  $\mathcal{G}^{i,0}$ , is chosen as the two-body embedding matrix.

Then the scaled dot-product attention method is adopted:

$$A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l}) = \varphi(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}) \mathcal{V}^{i,l},$$

where  $\varphi(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}) \in \mathbb{R}^{N_c \times N_c}$  is attention weights. In the original attention method, one typically has  $\varphi(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}) = \text{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right)$ , with  $\sqrt{d_k}$  being the normalization temperature. This is slightly modified to incorporate the angular information:

$$\varphi(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{R}^{i,l}) = \text{softmax}\left(\frac{\mathcal{Q}^{i,l}(\mathcal{K}^{i,l})^T}{\sqrt{d_k}}\right) \odot \hat{\mathcal{R}}^i (\hat{\mathcal{R}}^i)^T,$$

where  $\hat{\mathcal{R}}^i \in \mathbb{R}^{N_c \times 3}$  denotes normalized relative coordinates,  $\hat{\mathcal{R}}_j^i = \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}$  and  $\odot$  means element-wise multiplication.

Then layer normalization is added in a residual way to finally obtain the self-attention local embedding matrix  $\hat{\mathcal{G}}^i = \mathcal{G}^{i,L_a}$  after  $L_a$  attention layers:<sup>1</sup>

$$\mathcal{G}^{i,l} = \mathcal{G}^{i,l-1} + \text{LayerNorm}(A(\mathcal{Q}^{i,l}, \mathcal{K}^{i,l}, \mathcal{V}^{i,l}, \mathcal{R}^{i,l})).$$

### 4.5.3 Introduction to new features of DPA-1

Next, we will list the detailed settings in input.json and the data format, especially for large systems with dozens of elements. An example of DPA-1 input can be found here.

#### Descriptor "se\_atten"

The notation of `se_atten` is short for the smooth edition of Deep Potential with an attention mechanism. This descriptor was described in detail in [the DPA-1 paper](#) and the images above.

In this example, we will train a DPA-1 model for a water system. A complete training input script of this example can be found in the directory:

```
$deepmd_source_dir/examples/water/se_atten/input.json
```

With the training input script, data are also provided in the example directory. One may train the model with the DeePMD-kit from the directory.

An example of the DPA-1 descriptor is provided as follows

```
"descriptor" :{
  "type":          "se_atten",
  "rcut_smth":     0.50,
  "rcut":          6.00,
  "sel":           120,
  "neuron":        [25, 50, 100],
  "axis_neuron":   16,
  "resnet_dt":     false,
  "attn":          128,
  "attn_layer":    2,
  "attn_mask":     false,
  "attn_dotr":     true,
  "seed":          1
}
```

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a [Creative Commons Attribution \(CC BY\) license](#).

- The type of the descriptor is set to "se\_attn", which will use DPA-1 structures.
- rcut is the cut-off radius for neighbor searching, and the rcut\_smth gives where the smoothing starts.
- sel gives the maximum possible number of neighbors in the cut-off radius. It is an int. Note that this number highly affects the efficiency of training, which we usually use less than 200. (We use 120 for training 56 elements in [OC2M dataset](#))
- The neuron specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a [ResNet architecture](#) is built between them.
- The axis\_neuron specifies the size of the submatrix of the embedding matrix, the axis matrix as explained in the [DeepPot-SE paper](#)
- If the option resnet\_dt is set to true, then a timestep is used in the ResNet.
- seed gives the random seed that is used to generate random numbers when initializing the model parameters.
- attn sets the length of a hidden vector during scale-dot attention computation.
- attn\_layer sets the number of layers in attention mechanism.
- attn\_mask determines whether to mask the diagonal in the attention weights and False is recommended.
- attn\_dotr determines whether to dot the relative coordinates on the attention weights as a gated scheme, True is recommended.

### Descriptor "se\_attn\_v2"

We highly recommend using the version 2.0 of the attention-based descriptor "se\_attn\_v2", which is inherited from "se\_attn" but with the following parameter modifications:

```
"stripped_type_embedding": true,
"smooth_type_embedding": true,
"set_davg_zero": false
```

Practical evidence demonstrates that "se\_attn\_v2" offers better and more stable performance compared to "se\_attn".

### Fitting "ener"

DPA-1 only supports "ener" fitting type, and you can refer [here](#) for detailed information.

### Type embedding

DPA-1 only supports models with type embeddings. And the default setting is as follows:

```
"type_embedding": {
  "neuron":      [8],
  "resnet_dt":   false,
  "seed":        1
}
```

You can add these settings in `input.json` if you want to change the default ones, see [here](#) for detailed information.

### Type map

For training large systems, especially those with dozens of elements, the `type` determines the element index of training data:

```
"type_map": [
  "Mg",
  "Al",
  "Cu"
]
```

which should include all the elements in the dataset you want to train on.

### 4.5.4 Data format

DPA-1 supports the standard data format, which is detailed in [data-conv.md](#) and [system.md](#). Note that in this format, only those frames with the same fingerprint (i.e. the number of atoms of different elements) can be put together as a unified system. This may lead to sparse frame numbers in those rare systems.

An ideal way is to put systems with the same total number of atoms together, which is the way we trained DPA-1 on [OC2M](#). This system format, which is called `mixed_type`, is proper to put frame-sparse systems together and is slightly different from the standard one. Take an example, a `mixed_type` may contain the following files:

```
type.raw
type_map.raw
set.*/box.npy
set.*/coord.npy
set.*/energy.npy
set.*/force.npy
set.*/real_atom_types.npy
```

This system contains `Nframes` frames with the same atom number `Natoms`, the total number of element types contained in all frames is `Ntypes`. Most files are the same as those in [standard formats](#), here we only list the distinct ones:

ID	Property	File	Re- quired/Optional	Shape	Description
/	Atom type indexes (place holder)	type.raw	Re- quired	Natoms	All zeros to fake the type input
type_map	Atom type names	type_map.raw	Re- quired	Ntypes	Atom names that map to atom type contained in all the frames, which is unnecessary to be contained in the periodic table
type	Atom type indexes of each frame	real_atom_types.npy	Re- quired	Nframes * Natoms	Integers that describe atom types in each frame, corresponding to indexes in type_map. -1 means virtual atoms.

With these edited files, one can put together frames with the same `Natoms`, instead of the same formula (like H2O). Note that this `mixed_type` format only supports `se_atten` descriptor.



To put frames with different `Natoms` into the same system, one can pad systems by adding virtual atoms whose type is `-1`. Virtual atoms do not contribute to any fitting property, so the atomic property of virtual atoms (e.g. forces) should be given zero.

The API to generate or transfer to `mixed_type` format is available on [dpdata](#) for a more convenient experience.

### 4.5.5 Training example

Here we upload the AlMgCu example shown in the paper, you can download it here: [Baidu disk](#); [Google disk](#).

## 4.6 Descriptor "hybrid"

This descriptor hybridizes multiple descriptors to form a new descriptor. For example, we have a list of descriptors denoted by  $\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_N$ , the hybrid descriptor is the concatenation of the list, i.e.  $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_N)$ .

### 4.6.1 Theory

A hybrid descriptor  $\mathcal{D}_{\text{hyb}}^i$  concatenates multiple kinds of descriptors into one descriptor:

$$\mathcal{D}_{\text{hyb}}^i = \{ \mathcal{D}_1^i \quad \mathcal{D}_2^i \quad \dots \quad \mathcal{D}_n^i \}.$$

The list of descriptors can be different types or the same descriptors with different parameters. This way, one can set the different cutoff radii for different descriptors.<sup>1</sup>

### 4.6.2 Instructions

To use the descriptor in DeePMD-kit, one firstly set the type to `hybrid`, then provide the definitions of the descriptors by the items in the `list`,

```
"descriptor" : {
  "type": "hybrid",
  "list" : [
    {
      "type" : "se_e2_a",
      ...
    },
    {
      "type" : "se_e2_r",
      ...
    }
  ]
},
```

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, *J. Chem. Phys.* 159, 054801 (2023) licensed under a [Creative Commons Attribution \(CC BY\)](#) license.

A complete training input script of this example can be found in the directory

```
$deepmd_source_dir/examples/water/hybrid/input.json
```

## 4.7 Determine sel

All descriptors require to set `sel`, which means the expected maximum number of type-*i* neighbors of an atom. DeePMD-kit will allocate memory according to `sel`.

`sel` should not be too large or too small. If `sel` is too large, the computing will become much slower and cost more memory. If `sel` is not enough, the energy will be not conserved, making the accuracy of the model worse.

To determine a proper `sel`, one can calculate the neighbor stat of the training data before training:

```
dp neighbor-stat -s data -r 6.0 -t O H
```

where `data` is the directory of data, `6.0` is the cutoff radius, and `O` and `H` is the type map. The program will give the `max_nbor_size`. For example, `max_nbor_size` of the water example is `[38, 72]`, meaning an atom may have 38 O neighbors and 72 H neighbors in the training data.

The `sel` should be set to a higher value than that of the training data, considering there may be some extreme geometries during MD simulations. As a result, we set `sel` to `[46, 92]` in the water example.

## 4.8 Fit energy

In this section, we will take `$deepmd_source_dir/examples/water/se_e2_a/input.json` as an example of the input file.

### 4.8.1 Theory

In the DP model, we let the fitting network  $\mathcal{F}_0$  maps the descriptor  $\mathcal{D}^i$  to a scalar, where the subscript 0 means that the output is a zero-order tensor (i.e. scalar). The model can then be used to predict the total potential energy of the system by

$$E = \sum_i E_i = \sum_i \mathcal{F}_0(\mathcal{D}^i),$$

where the output of the fitting network is treated as the atomic potential energy contribution, i.e.  $E_i$ . The output scalar can also be treated as other scalar properties defined on an atom, for example, the partial charge of atom  $i$ .

In some cases, atomic-specific or frame-specific parameters, such as electron temperature, may be treated as extra input to the fitting network. We denote the atomic and frame-specific parameters by  $\mathbf{P}^i \in \mathbb{R}^{N_p}$  (with  $N_p$  being the dimension) and  $\mathbf{Q} \in \mathbb{R}^{N_q}$  (with  $N_q$  being the dimension), respectively.

$$E_i = \mathcal{F}_0(\{\mathcal{D}^i, \mathbf{P}^i, \mathbf{Q}\}).$$

The atomic force  $\mathbf{F}_i$  and the virial tensor  $\Xi = (\Xi_{\alpha\beta})$  (if PBC is applied) can be derived from the potential energy  $E$ :

$$F_{i,\alpha} = -\frac{\partial E}{\partial r_{i,\alpha}},$$

$$\Xi_{\alpha\beta} = - \sum_{\gamma} \frac{\partial E}{\partial h_{\gamma\alpha}} h_{\gamma\beta},$$

where  $r_{i,\alpha}$  and  $F_{i,\alpha}$  denotes the  $\alpha$ -th component of the coordinate and force of atom  $i$ .  $h_{\alpha\beta}$  is the  $\beta$ -th component of the  $\alpha$ -th basis vector of the simulation region.

The properties  $\eta$  of the energy loss function could be energy  $E$ , force  $\mathbf{F}$ , virial  $\Xi$ , relative energy  $\Delta E$ , or any combination among them, and the loss functions of them are

$$\begin{aligned} L_E(\mathbf{x}; \boldsymbol{\theta}) &= \frac{1}{N} (E(\mathbf{x}; \boldsymbol{\theta}) - E^*)^2, \\ L_F(\mathbf{x}; \boldsymbol{\theta}) &= \frac{1}{3N} \sum_{k=1}^N \sum_{\alpha=1}^3 (F_{k,\alpha}(\mathbf{x}; \boldsymbol{\theta}) - F_{k,\alpha}^*)^2, \\ L_{\Xi}(\mathbf{x}; \boldsymbol{\theta}) &= \frac{1}{9N} \sum_{\alpha,\beta=1}^3 (\Xi_{\alpha\beta}(\mathbf{x}; \boldsymbol{\theta}) - \Xi_{\alpha\beta}^*)^2, \\ L_{\Delta E}(\mathbf{x}; \boldsymbol{\theta}) &= \frac{1}{N} (\Delta E(\mathbf{x}; \boldsymbol{\theta}) - \Delta E^*)^2, \end{aligned}$$

where  $F_{k,\alpha}$  is the  $\alpha$ -th component of the force on atom  $k$ , and the superscript  $*$  indicates the label of the property that should be provided in advance. Using  $N$  ensures that each loss of fitting property is averaged over atomic contributions before they contribute to the total loss by weight.

If part of atoms is more important than others, for example, certain atoms play an essential role when calculating free energy profiles or kinetic isotope effects, the MSE of atomic forces with prefactors  $q_k$  can also be used as the loss function:

$$L_F^p(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{3N} \sum_{k=1}^N \sum_{\alpha} q_k (F_{k,\alpha}(\mathbf{x}; \boldsymbol{\theta}) - F_{k,\alpha}^*)^2.$$

The atomic forces with larger prefactors will be fitted more accurately than those in other atoms.

If some forces are quite large, for example, forces can be greater than 60 eV/Å in high-temperature reactive simulations, one may also prefer the force loss is relative to the magnitude:

$$L_F^r(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{3N} \sum_{k=1}^N \sum_{\alpha} \left( \frac{F_{k,\alpha}(\mathbf{x}; \boldsymbol{\theta}) - F_{k,\alpha}^*}{|\mathbf{F}_k^*| + \nu} \right)^2.$$

where  $\nu$  is a small constant used to protect an atom where the magnitude of  $\mathbf{F}_k^*$  is small from having a large  $L_F^r$ . Benefiting from the relative force loss, small forces can be fitted more accurately.<sup>1</sup>

## 4.8.2 The fitting network

The construction of the fitting net is given by section fitting\_net

```
"fitting_net" : {
  "neuron":           [240, 240, 240],
  "resnet_dt":        true,
  "seed":             1
},
```

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

- neuron specifies the size of the fitting net. If two neighboring layers are of the same size, then a ResNet architecture is built between them.
- If the option `resnet_dt` is set to `true`, then a timestep is used in the ResNet.
- `seed` gives the random seed that is used to generate random numbers when initializing the model parameters.

### 4.8.3 Loss

The loss function  $L$  for training energy is given by

$$L = p_e L_e + p_f L_f + p_v L_v$$

where  $L_e$ ,  $L_f$ , and  $L_v$  denote the loss in energy, forces and virials, respectively.  $p_e$ ,  $p_f$ , and  $p_v$  give the prefactors of the energy, force and virial losses. The prefactors may not be a constant, rather it changes linearly with the learning rate. Taking the force prefactor for example, at training step  $t$ , it is given by

$$p_f(t) = p_f^0 \frac{\alpha(t)}{\alpha(0)} + p_f^\infty (1 - \frac{\alpha(t)}{\alpha(0)})$$

where  $\alpha(t)$  denotes the learning rate at step  $t$ .  $p_f^0$  and  $p_f^\infty$  specifies the  $p_f$  at the start of the training and the limit of  $t \rightarrow \infty$  (set by `start_pref_f` and `limit_pref_f`, respectively), i.e.

$$pref_f(t) = start\_pref_f * (lr(t)/start\_lr) + limit\_pref_f * (1 - lr(t)/start\_lr)$$

The `loss` section in the `input.json` is

```
"loss" : {
  "start_pref_e":    0.02,
  "limit_pref_e":    1,
  "start_pref_f":    1000,
  "limit_pref_f":    1,
  "start_pref_v":    0,
  "limit_pref_v":    0
}
```

The options `start_pref_e`, `limit_pref_e`, `start_pref_f`, `limit_pref_f`, `start_pref_v` and `limit_pref_v` determine the start and limit prefactors of energy, force and virial, respectively.

If one does not want to train with virial, then he/she may set the virial prefactors `start_pref_v` and `limit_pref_v` to 0.

## 4.9 Fit spin energy

In this section, we will take `$deepmd_source_dir/examples/NiO/se_e2_a/input.json` as an example of the input file.

### 4.9.1 Spin

The construction of the fitting net is give by section [spin](#)

```
"spin" : {
  "use_spin":      [true, false],
  "virtual_len":    [0.4],
  "spin_norm":     [1.2737],
},
```

- use\_spin determines whether to turn on the magnetism of the atoms. The index of this option matches option type\_map <model/type\_map>.
- virtual\_len specifies the distance between virtual atom and the belonging real atom.
- spin\_norm gives the magnitude of the magnetic moment for each magnatic atom.

### 4.9.2 Spin Loss

The spin loss function  $L$  for training energy is given by

$$L = p_e L_e + p_{fr} L_{fr} + p_{fm} L_{fm} + p_v L_v$$

where  $L_e$ ,  $L_{fr}$ ,  $L_{fm}$  and  $L_v$  denote the loss in energy, atomic force, magnatic force and virial, respectively.  $p_e$ ,  $p_{fr}$ ,  $p_{fm}$  and  $p_v$  give the prefactors of the energy, atomic force, magnatic force and virial losses.

The prefectors may not be a constant, rather it changes linearly with the learning rate. Taking the atomic force prefactor for example, at training step  $t$ , it is given by

$$p_{fr}(t) = p_{fr}^0 \frac{\alpha(t)}{\alpha(0)} + p_{fr}^\infty (1 - \frac{\alpha(t)}{\alpha(0)})$$

where  $\alpha(t)$  denotes the learning rate at step  $t$ .  $p_{fr}^0$  and  $p_{fr}^\infty$  specifies the  $p_f$  at the start of the training and at the limit of  $t \rightarrow \infty$  (set by [start\\_pref\\_fr](#) and [limit\\_pref\\_f](#), respectively), i.e.

$$pref_{fr}(t) = start_{pref_{fr}} * (lr(t)/start_{lr}) + limit_{pref_{fr}} * (1 - lr(t)/start_{lr})$$

The [loss](#) section in the `input.json` is

```
"loss" :{
  "type":              "ener_spin",
  "start_pref_e":      0.02,
  "limit_pref_e":      1,
  "start_pref_fr":     1000,
  "limit_pref_fr":     1.0,
  "start_pref_fm":     10000,
  "limit_pref_fm":     10.0,
  "start_pref_v":      0,
  "limit_pref_v":      0,
},
```

The options [start\\_pref\\_e](#), [limit\\_pref\\_e](#), [start\\_pref\\_fr](#), [limit\\_pref\\_fm](#), [start\\_pref\\_v](#) and [limit\\_pref\\_v](#) determine the start and limit prefactors of energy, atomic force, magnatic force and virial, respectively.

If one does not want to train with virial, then he/she may set the virial prefactors [start\\_pref\\_v](#) and [limit\\_pref\\_v](#) to 0.

## 4.10 Fit tensor like Dipole and Polarizability

Unlike energy, which is a scalar, one may want to fit some high dimensional physical quantity, like dipole (vector) and polarizability (matrix, shorted as polar). Deep Potential has provided different APIs to do this. In this example, we will show you how to train a model to fit a water system. A complete training input script of the examples can be found in

```
$deepmd_source_dir/examples/water_tensor/dipole/dipole_input.json
$deepmd_source_dir/examples/water_tensor/polar/polar_input.json
```

The training and validation data are also provided our examples. But note that the data provided along with the examples are of limited amount, and should not be used to train a production model.

Similar to the `input.json` used in `ener` mode, training JSON is also divided into `model`, `learning_rate`, `loss` and `training`. Most keywords remain the same as `ener` mode, and their meaning can be found [here](#). To fit a tensor, one needs to modify `model/fitting_net` and `loss`.

### 4.10.1 Theory

To represent the first-order tensorial properties (i.e. vector properties), we let the fitting network, denoted by  $\mathcal{F}_1$ , output an  $M$ -dimensional vector; then we have the representation,

$$(T_i^{(1)})_\alpha = \frac{1}{N_c} \sum_{j=1}^{N_c} \sum_{m=1}^M (\mathcal{G}^i)_{jm} (\mathcal{R}^i)_{j,\alpha+1} (\mathcal{F}_1(\mathcal{D}^i))_m, \alpha = 1, 2, 3.$$

We let the fitting network  $\mathcal{F}_2$  output an  $M$ -dimensional vector, and the second-order tensorial properties (matrix properties) are formulated as

$$(T_i^{(2)})_{\alpha\beta} = \frac{1}{N_c^2} \sum_{j=1}^{N_c} \sum_{k=1}^{N_c} \sum_{m=1}^M (\mathcal{G}^i)_{jm} (\mathcal{R}^i)_{j,\alpha+1} (\mathcal{R}^i)_{k,\beta+1} (\mathcal{G}^i)_{km} (\mathcal{F}_2(\mathcal{D}^i))_m, \alpha, \beta = 1, 2, 3,$$

where  $\mathcal{G}^i$  and  $\mathcal{R}^i$  can be found in [se\\_e2\\_a](#). Thus, the tensor fitting network requires the descriptor to have the same or similar form as the DeepPot-SE descriptor.  $\mathcal{F}_1$  and  $\mathcal{F}_2$  are the neural network functions. The total tensor  $\mathbf{T}$  (total dipole  $\mathbf{T}^{(1)}$  or total polarizability  $\mathbf{T}^{(2)}$ ) is the sum of the atomic tensor:

$$\mathbf{T} = \sum_i \mathbf{T}_i.$$

The tensorial models can be used to calculate IR spectrum and Raman spectrum.<sup>1</sup>

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

### 4.10.2 The fitting Network

The `fitting_net` section tells DP which fitting net to use.

The JSON of `dipole` type should be provided like

```
"fitting_net" : {
  "type": "dipole",
  "sel_type": [0],
  "neuron": [100,100,100],
  "resnet_dt": true,
  "seed": 1,
},
```

The JSON of `polar` type should be provided like

```
"fitting_net" : {
  "type": "polar",
  "sel_type": [0],
  "neuron": [100,100,100],
  "resnet_dt": true,
  "seed": 1,
},
```

- `type` specifies which type of fitting net should be used. It should be either `dipole` or `polar`. Note that `global_polar` mode in version 1.x is already deprecated and is merged into `polar`. To specify whether a system is global or atomic, please see [here](#).
- `sel_type` is a list specifying which type of atoms have the quantity you want to fit. For example, in the water system, `sel_type` is `[0]` since 0 represents atom O. If left unset, all types of atoms will be fitted.
- The rest arguments have the same meaning as they do in `ener` mode.

### 4.10.3 Loss

DP supports a combinational training of the global system (only a global `tensor` label, i.e. `dipole` or `polar`, is provided in a frame) and atomic system (labels for each atom included in `sel_type` are provided). In a global system, each frame has just one `tensor` label. For example, when fitting `polar`, each frame will just provide a `1 x 9` vector which gives the elements of the polarizability tensor of that frame in order `XX, XY, XZ, YX, YY, YZ, XZ, ZY, ZZ`. By contrast, in an atomic system, each atom in `sel_type` has a `tensor` label. For example, when fitting a `dipole`, each frame will provide a `#sel_atom x 3` matrices, where `#sel_atom` is the number of atoms whose type are in `sel_type`.

The `loss` section tells DP the weight of these two kinds of loss, i.e.

```
loss = pref * global_loss + pref_atomic * atomic_loss
```

The loss section should be provided like

```
"loss" : {
  "type": "tensor",
  "pref": 1.0,
  "pref_atomic": 1.0
},
```

- `type` should be written as `tensor` as a distinction from `ener` mode.

- `pref` and `pref_atomic` respectively specify the weight of global loss and atomic loss. It can not be left unset. If set to 0, the corresponding label will NOT be included in the training process.

#### 4.10.4 Training Data Preparation

In tensor mode, the identification of the label's type (global or atomic) is derived from the file name. The global label should be named `dipole.npy/raw` or `polarizability.npy/raw`, while the atomic label should be named `atomic_dipole.npy/raw` or `atomic_polarizability.npy/raw`. If wrongly named, DP will report an error

```
ValueError: cannot reshape array of size xxx into shape (xx,xx). This error may occur when your
↪label mismatch it's name, i.e. you might store global tensor in `atomic_tensor.npy` or atomic
↪tensor in `tensor.npy`.
```

In this case, please check the file name of the label.

#### 4.10.5 Train the Model

The training command is the same as `ener` mode, i.e.

```
dp train input.json
```

The detailed loss can be found in `lcurve.out`:

#	step	rmse_val	rmse_trn	rmse_lc_val	rmse_lc_trn	rmse_gl_val	rmse_gl_trn	lr
	0	8.34e+00	8.26e+00	8.34e+00	8.26e+00	0.00e+00	0.00e+00	1.0e-02
	100	3.51e-02	8.55e-02	0.00e+00	8.55e-02	4.38e-03	0.00e+00	5.0e-03
	200	4.77e-02	5.61e-02	0.00e+00	5.61e-02	5.96e-03	0.00e+00	2.5e-03
	300	5.68e-02	1.47e-02	0.00e+00	0.00e+00	7.10e-03	1.84e-03	1.3e-03
	400	3.73e-02	3.48e-02	1.99e-02	0.00e+00	2.18e-03	4.35e-03	6.3e-04
	500	2.77e-02	5.82e-02	1.08e-02	5.82e-02	2.11e-03	0.00e+00	3.2e-04
	600	2.81e-02	5.43e-02	2.01e-02	0.00e+00	1.01e-03	6.79e-03	1.6e-04
	700	2.97e-02	3.28e-02	2.03e-02	0.00e+00	1.17e-03	4.10e-03	7.9e-05
	800	2.25e-02	6.19e-02	9.05e-03	0.00e+00	1.68e-03	7.74e-03	4.0e-05
	900	3.18e-02	5.54e-02	9.93e-03	5.54e-02	2.74e-03	0.00e+00	2.0e-05
	1000	2.63e-02	5.02e-02	1.02e-02	5.02e-02	2.01e-03	0.00e+00	1.0e-05
	1100	3.27e-02	5.89e-02	2.13e-02	5.89e-02	1.43e-03	0.00e+00	5.0e-06
	1200	2.85e-02	2.42e-02	2.85e-02	0.00e+00	0.00e+00	3.02e-03	2.5e-06
	1300	3.47e-02	5.71e-02	1.07e-02	5.71e-02	3.00e-03	0.00e+00	1.3e-06
	1400	3.13e-02	5.76e-02	3.13e-02	5.76e-02	0.00e+00	0.00e+00	6.3e-07
	1500	3.34e-02	1.11e-02	2.09e-02	0.00e+00	1.57e-03	1.39e-03	3.2e-07
	1600	3.11e-02	5.64e-02	3.11e-02	5.64e-02	0.00e+00	0.00e+00	1.6e-07
	1700	2.97e-02	5.05e-02	2.97e-02	5.05e-02	0.00e+00	0.00e+00	7.9e-08
	1800	2.64e-02	7.70e-02	1.09e-02	0.00e+00	1.94e-03	9.62e-03	4.0e-08
	1900	3.28e-02	2.56e-02	3.28e-02	0.00e+00	0.00e+00	3.20e-03	2.0e-08
	2000	2.59e-02	5.71e-02	1.03e-02	5.71e-02	1.94e-03	0.00e+00	1.0e-08

One may notice that in each step, some of the local loss and global loss will be 0.0. This is because our training data and validation data consist of the global system and atomic system, i.e.

```
--training_data
    >atomic_system
    >global_system
--validation_data
```

(continues on next page)



(continued from previous page)

```
>atomic_system
>global_system
```

During training, at each step when the `lcurve.out` is printed, the system used for evaluating the training (validation) error may be either with only global or only atomic labels, thus the corresponding atomic or global errors are missing and are printed as zeros.

## 4.11 Fit electronic density of states (DOS)

Here we present an API to DeepDOS model, which can be used to fit electronic density of state (DOS) (which is a vector).

See the [PRB paper](#) for details.

In this example, we will show you how to train a model to fit a silicon system. A complete training input script of the examples can be found in

```
$deepmd_source_dir/examples/dos/input.json
```

The training and validation data are also provided our examples. But note that the data provided along with the examples are of limited amount, and should not be used to train a production model.

Similar to the `input.json` used in `ener` mode, training JSON is also divided into `model`, `learning_rate`, `loss` and `training`. Most keywords remain the same as `ener` mode, and their meaning can be found [here](#). To fit the `dos`, one needs to modify `model/fitting_net` and `loss`.

### 4.11.1 The fitting Network

The `fitting_net` section tells DP which fitting net to use.

The JSON of `dos` type should be provided like

```
"fitting_net" : {
  "type": "dos",
  "numb_dos": 250,
  "sel_type": [0],
  "neuron": [120,120,120],
  "resnet_dt": true,
  "fparam": 0,
  "seed": 1,
},
```

- `type` specifies which type of fitting net should be used. It should be `dos`.
- `numb_dos` specifies the length of output vector (density of states), which the same as the `NEDOS` set in VASP software, this argument defines the output length of the neural network. We note that the length of `dos` provided in training set should be the same.
- The rest arguments have the same meaning as they do in `ener` mode.

### 4.11.2 Loss

DeepDOS supports trainings of the global system (a global `dos` label is provided in a frame) or atomic system (atomic labels `atom_dos` is provided for each atom in a frame). In a global system, each frame has just one `dos` label. For example, when fitting `dos`, each frame will just provide a `1 x numb_dos` vector which gives the total electronic density of states. By contrast, in an atomic system, each atom in has a `atom_dos` label. For example, when fitting the site-projected electronic density of states, each frame will provide a `natom x numb_dos` matrices,

The `loss` section tells DP the weight of these two kinds of loss, i.e.

```
loss = pref * global_loss + pref_atomic * atomic_loss
```

The loss section should be provided like

```
"loss" : {
    "type": "dos",
    "start_pref_dos": 0.0,
    "limit_pref_dos": 0.0,
    "start_pref_cdf": 0.0,
    "limit_pref_cdf": 0.0,
    "start_pref_ados": 1.0,
    "limit_pref_ados": 1.0,
    "start_pref_acdf": 0.0,
    "limit_pref_acdf": 0.0
},
```

- `type` should be written as `dos` as a distinction from `ener` mode.
- `pref_dos` and `pref_ados`, respectively specify the weight of global and atomic loss. If set to 0, the corresponding label will not be included in the training process.
- We also provides a combination training of vector and its cumulative distribution function `cdf`, which can be defined as

$$D(\epsilon) = \int_{\epsilon_{min}}^{\epsilon} g(\epsilon') d\epsilon'$$

### 4.11.3 Training Data Preparation

The global label should be named `dos.npy/raw`, while the atomic label should be named `atomic_dos.npy/raw`. If wrongly named, DP will report an error.

To prepare the data, we recommend shifting the DOS data by the Fermi level.

### 4.11.4 Train the Model

The training command is the same as `ener` mode, i.e.

```
dp train input.json
```

The detailed loss can be found in `lcurve.out`:

#	step	rmse_trn	rmse_ados_trn	rmse_ados_lr
	0	1.11e+00	1.11e+00	1.0e-03
	100	5.00e-02	5.00e-02	1.0e-03

(continues on next page)

(continued from previous page)

200	4.70e-02	4.70e-02	1.0e-03
300	6.45e-02	6.45e-02	1.0e-03
400	3.39e-02	3.39e-02	1.0e-03
500	4.60e-02	4.60e-02	1.0e-03
600	3.98e-02	3.98e-02	1.0e-03
700	9.50e-02	9.50e-02	1.0e-03
800	5.49e-02	5.49e-02	1.0e-03
900	5.57e-02	5.57e-02	1.0e-03
1000	3.73e-02	3.73e-02	1.0e-03
1100	4.33e-02	4.33e-02	1.0e-03
1200	3.27e-02	3.27e-02	1.0e-03
1300	3.68e-02	3.68e-02	1.0e-03
1400	3.09e-02	3.09e-02	1.0e-03
1500	3.42e-02	3.42e-02	1.0e-03
1600	5.62e-02	5.62e-02	1.0e-03
1700	6.12e-02	6.12e-02	1.0e-03
1800	4.10e-02	4.10e-02	1.0e-03
1900	5.30e-02	5.30e-02	1.0e-03
2000	3.85e-02	3.85e-02	1.0e-03

#### 4.11.5 Test the Model

In this earlier version, we can use `dp test` to infer the electronic density of state for given frames.

```
$DP freeze -o frozen_model.pb
```

```
$DP test -m frozen_model.pb -s ../data/111/$k -d ${output_prefix} -a -n 100
```

if `dp test -d ${output_prefix} -a` is specified, the predicted DOS and atomic DOS for each frame is output in the working directory

```
${output_prefix}.ados.out.0  ${output_prefix}.ados.out.1  ${output_prefix}.ados.out.2  ${output_
↪prefix}.ados.out.3
${output_prefix}.dos.out.0   ${output_prefix}.dos.out.1  ${output_prefix}.dos.out.2  ${output_
↪prefix}.dos.out.3
```

for `*.dos.out.*`, it contains matrix with shape of (2, numb\_dos), for `*.ados.out.*`, it contains matrix with shape of (2, natom x numb\_dos),

```
# frame - 0: data_dos pred_dos
0.0000000000000000e+00 1.963193264917645342e-03
0.0000000000000000e+00 1.178440836781313727e-03
0.0000000000000000e+00 1.441258071790407769e-04
0.0000000000000000e+00 1.787297933314058174e-03
0.0000000000000000e+00 1.901603280243024940e-03
0.0000000000000000e+00 2.279848925571981155e-03
0.0000000000000000e+00 2.149355854688561607e-03
0.0000000000000000e+00 1.829848459515726056e-03
0.0000000000000000e+00 1.905156512419792225e-03
```

## 4.12 Type embedding approach

We generate specific a type embedding vector for each atom type so that we can share one descriptor embedding net and one fitting net in total, which decline training complexity largely.

The training input script is similar to that of `se_e2_a`, but different by adding the `type_embedding` section.

### 4.12.1 Theory

Usually, when the type embedding approach is not enabled, for a system with multiple chemical species ( $|\{\alpha_i\}| > 1$ ), parameters of the embedding network  $\mathcal{N}_{e,\{2,3\}}$  are as follows chemical-species-wise:

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}^{\alpha_i,\alpha_j}(s(r_{ij})) \quad \text{or} \quad (\mathcal{G}^i)_j = \mathcal{N}_{e,2}^{\alpha_j}(s(r_{ij})),$$

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}^{\alpha_j,\alpha_k}((\theta_i)_{jk}).$$

Thus, there will be  $N_t^2$  or  $N_t$  embedding networks where  $N_t$  is the number of chemical species. To improve the performance of matrix operations,  $n(i)$  is divided into blocks of different chemical species. Each matrix with a dimension of  $N_e$  is divided into corresponding blocks, and each block is padded to  $N_e^{\alpha_j}$  separately. The limitation of this approach is that when there are large numbers of chemical species, the number of embedding networks will increase, requiring large memory and decreasing computing efficiency.

Similar to the embedding networks, if the type embedding approach is not used, the fitting network parameters are chemical-species-wise, and there are  $N_t$  sets of fitting network parameters. For performance, atoms are sorted by their chemical species  $\alpha_i$  in advance. Take an example, the atomic energy  $E_i$  is represented as follows:

$$E_i = \mathcal{F}_0^{\alpha_i}(\mathcal{D}^i).$$

To reduce the number of NN parameters and improve computing efficiency when there are large numbers of chemical species, the type embedding  $\mathcal{A}$  is introduced, represented as a NN function  $\mathcal{N}_t$  of the atomic type  $\alpha$ :

$$\mathcal{A}^i = \mathcal{N}_t(\text{one hot}(\alpha_i)),$$

where  $\alpha_i$  is converted to a one-hot vector representing the chemical species before feeding to the NN. The type embeddings of central and neighboring atoms  $\mathcal{A}^i$  and  $\mathcal{A}^j$  are added as an extra input of the embedding network  $\mathcal{N}_{e,\{2,3\}}$ :

$$(\mathcal{G}^i)_j = \mathcal{N}_{e,2}(\{s(r_{ij}), \mathcal{A}^i, \mathcal{A}^j\}) \quad \text{or} \quad (\mathcal{G}^i)_j = \mathcal{N}_{e,2}(\{s(r_{ij}), \mathcal{A}^j\}),$$

$$(\mathcal{G}^i)_{jk} = \mathcal{N}_{e,3}(\{(\theta_i)_{jk}, \mathcal{A}^j, \mathcal{A}^k\}).$$

In fitting networks, the type embedding is inserted into the input of the fitting networks:

$$E_i = \mathcal{F}_0(\{\mathcal{D}^i, \mathcal{A}^i\}).$$

In this way, all chemical species share the same network parameters through the type embedding.<sup>1</sup>

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

### 4.12.2 Instructions

The `model` defines how the model is constructed, adding a section of type embedding net:

```
"model": {
  "type_map":      ["O", "H"],
  "type_embedding":{
    ...
  },
  "descriptor" :{
    ...
  },
  "fitting_net" : {
    ...
  }
}
```

The model will automatically apply the type embedding approach and generate type embedding vectors. If the type embedding vector is detected, the descriptor and fitting net would take it as a part of the input.

The construction of type embedding net is given by `type_embedding`. An example of `type_embedding` is provided as follows

```
"type_embedding":{
  "neuron":      [2, 4, 8],
  "resnet_dt":    false,
  "seed":        1
}
```

- The `neuron` specifies the size of the type embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. It takes a one-hot vector as input and output dimension equals to the last dimension of the `neuron` list. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a ResNet architecture is built between them.
- If the option `resnet_dt` is set to `true`, then a timestep is used in the ResNet.
- `seed` gives the random seed that is used to generate random numbers when initializing the model parameters.

A complete training input script of this example can be found in the directory.

```
$deepmd_source_dir/examples/water/se_e2_a_tebd/input.json
```

See [here](#) for further explanation of type embedding.

---

Note: You can't apply the compression method while using the atom type embedding.

---

## 4.13 Descriptor "se\_a\_mask"

Descriptor `se_a_mask` is a concise implementation of the descriptor `se_e2_a`, but functions slightly differently. `se_a_mask` is specially designed for DP/MM simulations where the number of atoms in DP regions is dynamically changed in simulations.

Therefore, the descriptor `se_a_mask` is not supported for training with PBC systems for simplicity. Besides, to make the output shape of the descriptor matrix consistent, the input coordinates are padded with virtual particle coordinates to the maximum number of atoms (specified with `sel` in the descriptor setting) in the system. The real/virtual sign of the atoms is specified with the `aparam.npy` (`[ nframes * natoms ]`) file in the input systems set directory. The `aparam.npy` can also be seen as the mask of the atoms in the system, which is also the origin of the name `se_a_mask`.

In this example, we will train a DP Mask model for zinc protein interactions. The input systems are the collection of zinc and its coordinates residues. A sample input system that contains 2 frames is included in the directory.

```
$deepmd_source_dir/examples/zinc_protein/data_dp_mask
```

A complete training input script of this example can be found in the directory.

```
$deepmd_source_dir/examples/zinc_protein/zinc_se_a_mask.json
```

The construction of the descriptor is given by section descriptor. An example of the descriptor is provided as follows

```
"descriptor" :{
  "type":      "se_a_mask",
  "sel":       [36, 16, 24, 64, 6, 1],
  "neuron":    [25, 50, 100],
  "axis_neuron": 16,
  "type_one_side": false,
  "resnet_dt":  false,
  "seed":      1
}
```

- The type of the descriptor is set to `"se_a_mask"`.
- `sel` gives the maximum number of atoms in input coordinates. It is a list, the length of which is the same as the number of atom types in the system, and `sel[i]` denotes the maximum number of atoms with type `i`.
- The neuron specifies the size of the embedding net. From left to right the members denote the sizes of each hidden layer from the input end to the output end, respectively. If the outer layer is twice the size of the inner layer, then the inner layer is copied and concatenated, then a [ResNet architecture](#) is built between them.
- The `axis_neuron` specifies the size of the submatrix of the embedding matrix, the axis matrix as explained in the [DeepPot-SE paper](#)
- If the option `type_one_side` is set to `true`, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{types}}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{types}}^2$  sets of embedding network parameters.
- If the option `resnet_dt` is set to `true`, then a timestep is used in the ResNet.
- `seed` gives the random seed that is used to generate random numbers when initializing the model parameters.

To make the `aparam.npy` used for descriptor `se_a_mask`, two variables in `fitting_net` section are needed.

```
"fitting_net" :{
  "neuron": [240, 240, 240],
  "resnet_dt": true,
  "seed": 1,
  "numb_aparam": 1,
  "use_aparam_as_mask": true
}
```

- `neuron`, `resnet_dt` and `seed` are the same as the `fitting_net` section for fitting energy.
- `numb_aparam` gives the dimension of the `aparam.npy` file. In this example, it is set to 1 and stores the real/virtual sign of the atoms. For real/virtual atoms, the corresponding sign in `aparam.npy` is set to 1/0.
- `use_aparam_as_mask` is set to `true` to use the `aparam.npy` as the mask of the atoms in the descriptor `se_a_mask`.

Finally, to make a reasonable fitting task with `se_a_mask` descriptor for DP/MM simulations, the loss function with `se_a_mask` is designed to include the atomic forces difference in specific atoms of the input particles only. More details about the selection of the specific atoms can be found in paper [DP/MM](left to be filled). Thus, `atom_pref.npy` (`[ nframes * natoms ]`) is required as the indicator of the specific atoms in the input particles. And the `loss` section in the training input script should be set as follows.

```
"loss": {
  "type": "ener",
  "start_pref_e": 0.0,
  "limit_pref_e": 0.0,
  "start_pref_f": 0.0,
  "limit_pref_f": 0.0,
  "start_pref_pf": 1.0,
  "limit_pref_pf": 1.0,
  "_comment": " that's all"
}
```

## 4.14 Deep potential long-range (DPLR)

Notice: The interfaces of DPLR are not stable and subject to change

The method of DPLR is described in [this paper](#). One is recommended to read the paper before using the DPLR.

In the following, we take the DPLR model for example to introduce the training and LAMMPS simulation with the DPLR model. The DPLR model is trained in two steps.

### 4.14.1 Theory

The Deep Potential Long Range (DPLR) model adds the electrostatic energy to the total energy:

$$E = E_{\text{DP}} + E_{\text{ele}},$$

where  $E_{\text{DP}}$  is the short-range contribution constructed as the [standard energy model](#) that is fitted against  $(E^* - E_{\text{ele}})$ .  $E_{\text{ele}}$  is the electrostatic energy introduced by a group of Gaussian distributions that is an approximation of the electronic structure of the system, and is calculated in Fourier space by

$$E_{\text{ele}} = \frac{1}{2\pi V} \sum_{m \neq 0, \|m\| \leq L} \frac{\exp(-\pi^2 m^2 / \beta^2)}{m^2} S^2(m),$$

where  $\beta$  is a freely tunable parameter that controls the spread of the Gaussians.  $L$  is the cutoff in Fourier space and  $S(m)$ , the structure factor, is given by

$$S(m) = \sum_i q_i e^{-2\pi i m \mathbf{r}_i} + \sum_n q_n e^{-2\pi i m \mathbf{W}_n},$$

where  $i = \sqrt{-1}$  denotes the imaginary unit,  $\mathbf{r}_i$  indicates ion coordinates,  $q_i$  is the charge of the ion  $i$ , and  $\mathbf{W}_n$  is the  $n$ -th Wannier centroid (WC) which can be obtained from a separated [dipole model](#). It can be proved that the error in the electrostatic energy introduced by the Gaussian approximations is dominated by a summation of dipole-quadrupole interactions that decay as  $r^{-4}$ , where  $r$  is the distance between the dipole and quadrupole.<sup>1</sup>

### 4.14.2 Train a deep Wannier model for Wannier centroids

We use the deep Wannier model (DW) to represent the relative position of the Wannier centroid (WC) with the atom with which it is associated. One may consult the introduction of the [dipole model](#) for a detailed introduction. An example input `wc.json` and a small dataset `data` for tutorial purposes can be found in

```
$deepmd_source_dir/examples/water/dplr/train/
```

It is noted that the tutorial dataset is not enough for training a productive model. Two settings make the training input script different from an energy training input:

```
"fitting_net": {
  "type": "dipole",
  "dipole_type": [0],
  "neuron": [128, 128, 128],
  "seed": 1
},
```

The type of fitting is set to dipole. The dipole is associated with type 0 atoms (oxygen), by the setting `"dipole_type": [0]`. What we trained is the displacement of the WC from the corresponding oxygen atom. It shares the same training input as the atomic dipole because both are 3-dimensional vectors defined on atoms. The loss section is provided as follows

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a [Creative Commons Attribution \(CC BY\)](#) license.



```
"loss": {
  "type": "tensor",
  "pref": 0.0,
  "pref_atomic": 1.0
},
```

so that the atomic dipole is trained as labels. Note that the NumPy compressed file `atomic_dipole.npy` should be provided in each dataset.

The training and freezing can be started from the example directory by

```
dp train dw.json && dp freeze -o dw.pb
```

### 4.14.3 Train the DPLR model

The training of the DPLR model is very similar to the standard short-range DP models. An example input script can be found in the example directory. The following section is introduced to compute the long-range energy contribution of the DPLR model, and modify the short-range DP model by this part.

```
"modifier": {
  "type": "dipole_charge",
  "model_name": "dw.pb",
  "model_charge_map": [-8],
  "sys_charge_map": [6, 1],
  "ewald_h": 1.00,
  "ewald_beta": 0.40
},
```

The `model_name` specifies which DW model is used to predict the position of WCs. `model_charge_map` gives the amount of charge assigned to WCs. `sys_charge_map` provides the nuclear charge of oxygen (type 0) and hydrogen (type 1) atoms. `ewald_beta` (unit  $\text{\AA}^{-1}$ ) gives the spread parameter controls the spread of Gaussian charges, and `ewald_h` (unit  $\text{\AA}$ ) assigns the grid size of Fourier transformation. The DPLR model can be trained and frozen by (from the example directory)

```
dp train ener.json && dp freeze -o ener.pb
```

### 4.14.4 Molecular dynamics simulation with DPLR

In MD simulations, the long-range part of the DPLR is calculated by the LAMMPS `kpace` support. Then the long-range interaction is back-propagated to atoms by DeePMD-kit. This setup is commonly used in classical molecular dynamics simulations as the “virtual site”. Unfortunately, LAMMPS does not natively support virtual sites, so we have to hack the LAMMPS code, which makes the input configuration and script a little wired.

An example of an input configuration file and script can be found in

```
$deepmd_source_dir/examples/water/dplr/lmp/
```

We use `atom_style full` for DPLR simulations. the coordinates of the WCs are explicitly written in the configuration file. Moreover, a virtual bond is established between the oxygens and the WCs to indicate they are associated together. The configuration file containing 128 H2O molecules is thus written as

```

512 atoms
3 atom types
128 bonds
1 bond types

0 16.421037674 xlo xhi
0 16.421037674 ylo yhi
0 16.421037674 zlo zhi
0 0 0 xy xz yz

Masses

1 16
2 2
3 16

Atoms

      1      1 1 6 8.4960699081e+00 7.5073699951e+00 9.6371297836e+00
      2      2 1 6 4.0597701073e+00 6.8156299591e+00 1.2051420212e+01
...
      385     1 3 -8 8.4960699081e+00 7.5073699951e+00 9.6371297836e+00
      386     2 3 -8 4.0597701073e+00 6.8156299591e+00 1.2051420212e+01
...

Bonds

1 1 1 385
2 1 2 386
...
```

The oxygens and hydrogens are assigned with atom types 1 and 2 (corresponding to training atom types 0 and 1), respectively. The WCs are assigned with atom type 3. We want to simulate heavy water so the mass of hydrogens is set to 2.

An example input script is provided in

```
$deepmd_source_dir/examples/water/dplr/lmp/in.lammps
```

Here are some explanations

```

# groups of real and virtual atoms
group          real_atom type 1 2
group          virtual_atom type 3

# bond between real and its corresponding virtual site should be given
# to setup a map between real and virtual atoms. However, no real
# bonded interaction is applied, thus bond_style "zero" is used.
pair_style      deepmd ener.pb
pair_coeff      * *
bond_style      zero
bond_coeff      *
special_bonds   lj/coul 1 1 1 angle no
```

Type 1 and 2 (O and H) are `real_atoms`, while type 3 (WCs) are `virtual_atoms`. The model file `ener.pb` stores both the DW and DPLR models, so the position of WCs and the energy can be inferred from it. A

virtual bond type is specified by `bond_style zero`. The `special_bonds` command switches off the exclusion of intramolecular interactions.

```
# kspace_style "pppm/dplr" should be used. in addition the
# gewald(1/distance) should be set the same as that used in
# training. Currently only ik differentiation is supported.
kspace_style      pppm/dplr 1e-5
kspace_modify     gewald ${BETA} diff ik mesh ${KMESH} ${KMESH} ${KMESH}
```

The long-range part is calculated by the `kspace` support of LAMMPS. The `kspace_style pppm/dplr` is required. The spread parameter set by variable `BETA` should be set the same as that used in training. The `KMESH` should be set dense enough so the long-range calculation is converged.

### fix dplr command

Syntax

```
fix ID group-ID style_name keyword value ...
```

- ID, group-ID are documented in :doc:fix <fix> command
- style\_name = dplr
- three or more keyword/value pairs may be appended

```
keyword = *model* or *type_associate* or *bond_type* or *efield*
*model* value = name
name = name of DPLR model file (e.g. frozen_model.pb) (not DW model)
*type_associate* values = NR1 NW1 NR2 NW2 ...
NRi = type of real atom in i-th (real atom, Wannier centroid) pair
NWj = type of Wannier in i-th (real atom, Wannier centroid) pair
*bond_type* values = NB1 NB2 ...
NBi = bond type of i-th (real atom, Wannier centroid) pair
*efield* (optional) values = Ex Ey Ez
Ex/Ey/Ez = electric field along x/y/z direction
```

Examples

```
# "fix dplr" set the position of the virtual atom, and spread the
# electrostatic interaction asserting on the virtual atom to the real
# atoms. "type_associate" associates the real atom type its
# corresponding virtual atom type. "bond_type" gives the type of the
# bond between the real and virtual atoms.
fix          0 all dplr model ener.pb type_associate 1 3 bond_type 1
fix_modify   0 virial yes
```

The `fix` command `dplr` calculates the position of WCs by the DW model and back-propagates the long-range interaction on virtual atoms to real atoms. The atom names specified in `pair_style deepmd` will be used to determine elements. If it is not set, the training parameter `type_map` will be mapped to LAMMPS atom types.

To use a time-dependent electric field, LAMMPS's variable feature can be utilized:

```
variable EFIELD_Z equal 2*sin(2*PI*time/0.006)
fix 0 all dplr model ener.pb type_associate 1 3 bond_type 1 efield 0 0 v_EFIELD_Z
fix_modify 0 energy yes virial yes
```

The `efield` feature of `fix dplr` behaves similarly to LAMMPS's `fix efield`. Note that the atomic energy or potential in `fix efield` is not yet supported in `fix dplr`. For a detailed description on how a time-dependent variable can be defined, refer to LAMMPS's document of [variable](#).

```
# compute the temperature of real atoms, excluding virtual atom contribution
compute          real_temp real_atom temp
compute          real_press all pressure real_temp
fix              1 real_atom nvt temp ${TEMP} ${TEMP} ${TAU_T}
fix_modify       1 temp real_temp
```

The temperature of the system should be computed from the real atoms. The kinetic contribution in the pressure tensor is also computed from the real atoms. The thermostat is applied to only real atoms. The computed temperature and pressure of real atoms can be accessed by, e.g.

```
fix              thermo_print all print ${THERMO_FREQ} "$(step) $(pe) $(ke) $(etotal) $(enthalpy)
↪$(c_real_temp) $(c_real_press) $(vol) $(c_real_press[1]) $(c_real_press[2]) $(c_real_press[3])"
↪append thermo.out screen no title "# step pe ke etotal enthalpy temp press vol pxx pyy pzz"
```

The LAMMPS simulation can be started from the example directory by

```
lmp -i in.lammps
```

If LAMMPS complains that no model file `ener.pb` exists, it can be copied from the training example directory.

The MD simulation lasts for only 20 steps. If one runs a longer simulation, it will blow up, because the model is trained with a very limited dataset for very short training steps, thus is of poor quality.

Another restriction that should be noted is that the energies printed at the zero steps are not correct. This is because at the zero steps the position of the WC has not been updated with the DW model. The energies printed in later steps are correct.

## 4.15 Deep Potential - Range Correction (DPRc)

Deep Potential - Range Correction (DPRc) is designed to combine with QM/MM method, and corrects energies from a low-level QM/MM method to a high-level QM/MM method:

$$E = E_{\text{QM}}(\mathbf{R}; \mathbf{P}) + E_{\text{QM/MM}}(\mathbf{R}; \mathbf{P}) + E_{\text{MM}}(\mathbf{R}) + E_{\text{DPRc}}(\mathbf{R})$$

### 4.15.1 Theory

Deep Potential - Range Correction (DPRc) was initially designed to correct the potential energy from a fast, linear-scaling low-level semiempirical QM/MM theory to a high-level “ab initio” QM/MM theory in a range-correction way to quantitatively correct short and mid-range non-bonded interactions leveraging the non-bonded lists routinely used in molecular dynamics simulations using molecular mechanical force fields such as AMBER. In this way, long-ranged electrostatic interactions can be modeled efficiently using the particle mesh Ewald method or its extensions for multipolar and QM/MM potentials. In a DPRc model, the switch function is modified to disable MM-MM interaction:

$$s_{\text{DPRc}}(r_{ij}) = \begin{cases} 0, & \text{if } i \in \text{MM} \wedge j \in \text{MM}, \\ s(r_{ij}), & \text{otherwise,} \end{cases}$$

where  $s_{\text{DPRc}}(r_{ij})$  is the new switch function and  $s(r_{ij})$  is the old one. This ensures the forces between MM atoms are zero, i.e.

$$\mathbf{F}_{ij} = -\frac{\partial E}{\partial \mathbf{r}_{ij}} = 0, \quad i \in \text{MM} \wedge j \in \text{MM}.$$

The fitting network is revised to remove energy bias from MM atoms:

$$E_i = \begin{cases} \mathcal{F}_0(\mathcal{D}^i), & \text{if } i \in \text{QM}, \\ \mathcal{F}_0(\mathcal{D}^i) - \mathcal{F}_0(0), & \text{if } i \in \text{MM}, \end{cases}$$

where 0 is a zero matrix. It is worth mentioning that usage of DPRc is not limited to its initial design for QM/MM correction and can be expanded to any similar interaction.<sup>1</sup>

See the [JCTC paper](#) for details.

## 4.15.2 Training data

Instead the normal ab initio data, one needs to provide the correction from a low-level QM/MM method to a high-level QM/MM method:

$$E = E_{\text{high-level QM/MM}} - E_{\text{low-level QM/MM}}$$

Two levels of data use the same MM method, so  $E_{\text{MM}}$  is eliminated.

## 4.15.3 Training the DPRc model

In a DPRc model, QM atoms and MM atoms have different atom types. Assuming we have 4 QM atom types (C, H, O, P) and 2 MM atom types (HW, OW):

```
"type_map": ["C", "H", "HW", "O", "OW", "P"]
```

As described in the paper, the DPRc model only corrects  $E_{\text{QM}}$  and  $E_{\text{QM/MM}}$  within the cutoff, so we use a hybrid descriptor to describe them separately:

```
"descriptor" :{
  "type":      "hybrid",
  "list" : [
    {
      "type":    "se_a_ebd_v2",
      "sel":     [6, 11, 0, 6, 0, 1],
      "rcut_smth": 1.00,
      "rcut":     9.00,
      "neuron":   [12, 25, 50],
      "exclude_types": [[2, 2], [2, 4], [4, 4], [0, 2], [0, 4], [1, 2], [1, 4], [3, 2],
↪ [3, 4], [5, 2], [5, 4]],
      "axis_neuron": 12,
      "_comment": " QM/QM interaction"
    }
  ],
}
```

(continues on next page)

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a [Creative Commons Attribution \(CC BY\)](#) license.

(continued from previous page)

```

    {
      "type":      "se_a_ebd_v2",
      "sel":       [6, 11, 100, 6, 50, 1],
      "rcut_smth": 0.50,
      "rcut":       6.00,
      "neuron":     [12, 25, 50],
      "exclude_types": [[0, 0], [0, 1], [0, 3], [0, 5], [1, 1], [1, 3], [1, 5], [3, 3],
↪ [3, 5], [5, 5], [2, 2], [2, 4], [4, 4]],
      "axis_neuron": 12,
      "set_davg_zero": true,
      "_comment": " QM/MM interaction"
    }
  ]
}

```

exclude\_types can be generated by the following Python script:

```

from itertools import combinations_with_replacement, product

qm = (0, 1, 3, 5)
mm = (2, 4)
print(
    "QM/QM:",
    list(map(list, list(combinations_with_replacement(mm, 2)) + list(product(qm, mm)))),
)
print(
    "QM/MM:",
    list(
        map(
            list,
            list(combinations_with_replacement(qm, 2))
            + list(combinations_with_replacement(mm, 2)),
        )
    ),
)

```

Also, DPRc assumes MM atom energies (atom\_ener) are zero:

```

"fitting_net": {
  "neuron": [240, 240, 240],
  "resnet_dt": true,
  "atom_ener": [null, null, 0.0, null, 0.0, null]
}

```

Note that atom\_ener only works when descriptor/set\_davg\_zero of the QM/MM part is true.

#### 4.15.4 Run MD simulations

The DPRc model has the best practices with the [AMBER](#) QM/MM module. An example is given by [GitLab RutgersLBSR/AmberDPRc](#). In theory, DPRc is able to be used with any QM/MM package, as long as the DeePMD-kit package accepts QM atoms and MM atoms within the cutoff range and returns energies and forces.

#### 4.15.5 Pairwise DPRc

If one wants to correct from a low-level method into a full DFT level, and the system is too large to do full DFT calculation, one may try the experimental pairwise DPRc model. In a pairwise DPRc model, the total energy is divided into QM internal energy and the sum of QM/MM energy for each MM residue  $l$ :

$$E = E_{\text{QM}} + \sum_l E_{\text{QM/MM},l}$$

In this way, the interaction between the QM region and each MM fragmentation can be computed and trained separately. Thus, the pairwise DPRc model is divided into two sub-DPRc models. `qm_model` is for the QM internal interaction and `qmmm_model` is for the QM/MM interaction. The configuration for these two models is similar to the non-pairwise DPRc model. It is noted that the `se_attn` descriptor should be used, as it is the only descriptor to support the mixed type.

```
{
  "model": {
    "type": "pairwise_dprc",
    "type_map": [
      "C",
      "P",
      "O",
      "H",
      "OW",
      "HW"
    ],
    "type_embedding": {
      "neuron": [
        8
      ],
      "precision": "float32"
    },
    "qm_model": {
      "descriptor": {
        "type": "se_attn_v2",
        "sel": 24,
        "rcut_smth": 0.50,
        "rcut": 9.00,
        "attn_layer": 0,
        "neuron": [
          25,
          50,
          100
        ],
        "resnet_dt": false,
        "axis_neuron": 12,
        "precision": "float32",
        "seed": 1
      },
    },
  },
}
```

(continues on next page)

(continued from previous page)

```

    "fitting_net": {
      "type": "ener",
      "neuron": [
        240,
        240,
        240
      ],
      "resnet_dt": true,
      "precision": "float32",
      "atom_ener": [
        null,
        null,
        null,
        null,
        0.0,
        0.0
      ],
      "seed": 1
    }
  },
  "qmmm_model": {
    "descriptor": {
      "type": "se_atten_v2",
      "sel": 27,
      "rcut_smth": 0.50,
      "rcut": 6.00,
      "attn_layer": 0,
      "neuron": [
        25,
        50,
        100
      ],
      "resnet_dt": false,
      "axis_neuron": 12,
      "set_davg_zero": true,
      "exclude_types": [
        [
          0,
          0
        ],
        [
          0,
          1
        ],
        [
          0,
          2
        ],
        [
          0,
          3
        ],
        [
          1,
          1
        ]
      ]
    }
  }
}

```

(continues on next page)



(continued from previous page)

```

        [
            1,
            2
        ],
        [
            1,
            3
        ],
        [
            2,
            2
        ],
        [
            2,
            3
        ],
        [
            3,
            3
        ],
        [
            4,
            4
        ],
        [
            4,
            5
        ],
        [
            5,
            5
        ]
    ],
    "precision": "float32",
    "seed": 1
},
"fitting_net": {
    "type": "ener",
    "neuron": [
        240,
        240,
        240
    ],
    "resnet_dt": true,
    "seed": 1,
    "precision": "float32",
    "atom_ener": [
        0.0,
        0.0,
        0.0,
        0.0,
        0.0,
        0.0
    ]
}
}

```

(continues on next page)

(continued from previous page)

```
}
}
```

The pairwise model needs information for MM residues. The model uses `aparam` with the shape of `nframes` x `natoms` to get the residue index. The QM residue should always use 0 as the index. For example, 0 0 0 1 1 2 2 2 means these 9 atoms are grouped into one QM residue and two MM residues.

## 4.16 Linear model

One can linearly combine existing models with arbitrary coefficients:

```
"model": {
  "type": "linear_ener",
  "models": [
    {
      "type": "frozen",
      "model_file": "model0.pb"
    },
    {
      "type": "frozen",
      "model_file": "model1.pb"
    }
  ],
  "weights": [0.5, 0.5]
},
```

`weights` can be a list of floats, `mean`, or `sum`.

To obtain the model, one needs to execute `dp train` to do a zero-step training with `numb_steps` set to 0, and then freeze the model with `dp freeze`.

## 4.17 Interpolation or combination with a pairwise potential

### 4.17.1 Theory

In applications like the radiation damage simulation, the interatomic distance may become too close, so that the DFT calculations fail. In such cases, the DP model that is an approximation of the DFT potential energy surface is usually replaced by an empirical potential, like the Ziegler-Biersack-Littmark (ZBL) screened nuclear repulsion potential in the radiation damage simulations. The DeePMD-kit package supports the interpolation between DP and an empirical pairwise potential

$$E_i = (1 - w_i)E_i^{\text{DP}} + w_i(E_i^0 + E_i^{\text{pair}}),$$

where the  $w_i$  is the interpolation weight and the  $E_i^{\text{pair}}$  is the atomic contribution due to the pairwise potential  $u^{\text{pair}}(r)$ , i.e.

$$E_i^{\text{pair}} = \sum_{j \in n(i)} u^{\text{pair}}(r_{ij}).$$

The interpolation weight  $w_i$  is defined by

$$w_i = \begin{cases} 1, & \sigma_i r_a, \\ u_i^3(-6u_i^2 + 15u_i - 10) + 1, & r_a \leq \sigma_i r_b, \\ 0, & \sigma_i \geq r_b, \end{cases}$$

where  $u_i = (\sigma_i - r_a)/(r_b - r_a)$ .  $E_i^0$  is the atom energy bias. In the range  $[r_a, r_b]$ , the DP model smoothly switched off and the pairwise potential smoothly switched on from  $r_b$  to  $r_a$ . The  $\sigma_i$  is the softmin of the distance between atom  $i$  and its neighbors,

$$\sigma_i = \frac{\sum_{j \in n(i)} r_{ij} e^{-r_{ij}/\alpha_s}}{\sum_{j \in n(i)} e^{-r_{ij}/\alpha_s}},$$

where the scale  $\alpha_s$  is a tunable scale of the interatomic distance  $r_{ij}$ . The pairwise potential  $u^{\text{pair}}(r)$  is defined by a user-defined table that provides the value of  $u^{\text{pair}}$  on an evenly discretized grid from 0 to the cutoff distance.<sup>1</sup>

DeePMD-kit also supports combination with a pairwise potential:

$$E_i = E_i^{\text{DP}} + E_i^{\text{pair}},$$

### 4.17.2 Table file

The table file should be a text file that can be read by `numpy.loadtxt()`. The first column is the distance between two atoms, where upper range should be larger than the cutoff radius. Other columns are two-body interaction energies for pairs of certain types, in the order of Type\_0-Type\_0, Type\_0-Type\_1, ..., Type\_0-Type\_N, Type\_1-Type\_1, ..., Type\_1-Type\_N, ..., and Type\_N-Type\_N.

The interaction should be smooth at the cut-off distance.

### 4.17.3 Interpolation with a short-range pairwise potential

```
"model": {
  "use_srtab": "H2O_tab_potential.txt",
  "smin_alpha": 0.1,
  "sw_rmin": 0.8,
  "sw_rmax": 1.0,
  "_comment": "Below uses a normal DP model"
}
```

`sw_rmin` and `sw_rmax` must be smaller than the cutoff radius of the DP model.

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

#### 4.17.4 Combination with a pairwise potential

To combine with a pairwise potential, use the [linear model](#):

```
"model": {
  "type": "linear_ener",
  "weights": "sum",
  "models": [
    {
      "_comment": "Here uses a normal DP model"
    },
    {
      "type": "pairtab",
      "tab_file": "dftd3.txt",
      "rcut": 10.0,
      "sel": 534
    }
  ]
}
```

The `rcut` can be larger than that of the DP model.

---

## TRAINING

### 5.1 Train a model

Several examples of training can be found in the `examples` directory:

```
$ cd $deepmd_source_dir/examples/water/se_e2_a/
```

After switching to that directory, the training can be invoked by

```
$ dp train input.json
```

where `input.json` is the name of the input script.

By default, the verbosity level of the DeePMD-kit is `INFO`, one may see a lot of important information on the code and environment showing on the screen. Among them two pieces of information regarding data systems are worth special notice.

```
DEEPMD INFO    ---Summary of DataSystem: training  -----
↪--
DEEPMD INFO    found 3 system(s):
DEEPMD INFO
DEEPMD INFO          system  natoms  bch_sz  n_bch  prob  pbc
DEEPMD INFO    ../data_water/data_0/    192    1    80  0.250  T
DEEPMD INFO    ../data_water/data_1/    192    1   160  0.500  T
DEEPMD INFO    ../data_water/data_2/    192    1    80  0.250  T
DEEPMD INFO    -----
↪--
DEEPMD INFO    ---Summary of DataSystem: validation -----
↪--
DEEPMD INFO    found 1 system(s):
DEEPMD INFO
DEEPMD INFO          system  natoms  bch_sz  n_bch  prob  pbc
DEEPMD INFO    ../data_water/data_3    192    1    80  1.000  T
DEEPMD INFO    -----
↪--
```

The DeePMD-kit prints detailed information on the training and validation data sets. The data sets are defined by `training_data` and `validation_data` defined in the `training` section of the input script. The training data set is composed of three data systems, while the validation data set is composed by one data system. The number of atoms, batch size, the number of batches in the system and the probability of using the system are all shown on the screen. The last column presents if the periodic boundary condition is assumed for the system.

During the training, the error of the model is tested every `disp_freq` training steps with the batch used to train the model and with `numb_btch` batches from the validating data. The training error and validation error are printed correspondingly in the file `disp_file` (default is `lcurve.out`). The batch size can be set in the

input script by the key `batch_size` in the corresponding sections for the training and validation data set. An example of the output

#	step	rmse_val	rmse_trn	rmse_e_val	rmse_e_trn	rmse_f_val	rmse_f_trn	lr
	0	3.33e+01	3.41e+01	1.03e+01	1.03e+01	8.39e-01	8.72e-01	1.0e-03
	100	2.57e+01	2.56e+01	1.87e+00	1.88e+00	8.03e-01	8.02e-01	1.0e-03
	200	2.45e+01	2.56e+01	2.26e-01	2.21e-01	7.73e-01	8.10e-01	1.0e-03
	300	1.62e+01	1.66e+01	5.01e-02	4.46e-02	5.11e-01	5.26e-01	1.0e-03
	400	1.36e+01	1.32e+01	1.07e-02	2.07e-03	4.29e-01	4.19e-01	1.0e-03
	500	1.07e+01	1.05e+01	2.45e-03	4.11e-03	3.38e-01	3.31e-01	1.0e-03

The file contains 8 columns, from left to right, which are the training step, the validation loss, training loss, root mean square (RMS) validation error of energy, RMS training error of energy, RMS validation error of force, RMS training error of force and the learning rate. The RMS error (RMSE) of the energy is normalized by the number of atoms in the system. One can visualize this file with a simple Python script:

```
import numpy as np
import matplotlib.pyplot as plt

data = np.genfromtxt("lcurve.out", names=True)
for name in data.dtype.names[1:-1]:
    plt.plot(data["step"], data[name], label=name)
plt.legend()
plt.xlabel("Step")
plt.ylabel("Loss")
plt.xscale("symlog")
plt.yscale("log")
plt.grid()
plt.show()
```

Checkpoints will be written to files with the prefix `save_ckpt` every `save_freq` training steps.

Warning: It is warned that the example water data (in folder `examples/water/data`) is of very limited amount, is provided only for testing purposes, and should not be used to train a production model.

## 5.2 Advanced options

In this section, we will take `$deepmd_source_dir/examples/water/se_e2_a/input.json` as an example of the input file.

### 5.2.1 Learning rate

#### Theory

The learning rate  $\gamma$  decays exponentially:

$$\gamma(\tau) = \gamma^0 r^{\lfloor \tau/s \rfloor},$$

where  $\tau \in \mathbb{N}$  is the index of the training step,  $\gamma^0 \in \mathbb{R}$  is the learning rate at the first step, and the decay rate  $r$  is given by

$$r = \left( \frac{\gamma^{\text{stop}}}{\gamma^0} \right)^{\frac{s}{\tau^{\text{stop}}}},$$

where  $\tau^{\text{stop}} \in \mathbb{N}$ ,  $\gamma^{\text{stop}} \in \mathbb{R}$ , and  $s \in \mathbb{N}$  are the stopping step, the stopping learning rate, and the decay steps, respectively, all of which are hyperparameters provided in advance.<sup>1</sup>

## Instructions

The `learning_rate` section in `input.json` is given as follows

```
"learning_rate" :{
  "type":          "exp",
  "start_lr":      0.001,
  "stop_lr":       3.51e-8,
  "decay_steps":   5000,
  "_comment":      "that's all"
}
```

- `start_lr` gives the learning rate at the beginning of the training.
- `stop_lr` gives the learning rate at the end of the training. It should be small enough to ensure that the network parameters satisfactorily converge.
- During the training, the learning rate decays exponentially from `start_lr` to `stop_lr` following the formula:

$$\text{lr}(t) = \text{start\_lr} * \text{decay\_rate} ^ ( t / \text{decay\_steps} )$$

## 5.2.2 Training parameters

Other training parameters are given in the `training` section.

```
"training": {
  "training_data": {
    "systems":          ["../data_water/data_0/", "../data_water/data_1/", "../data_
↵water/data_2/"],
    "batch_size":       "auto"
  },
  "validation_data":{
    "systems":          ["../data_water/data_3"],
    "batch_size":       1,
    "numb_btch":        3
  },
  "mixed_precision": {
    "output_prec":      "float32",
    "compute_prec":     "float16"
  },

  "numb_steps":        1000000,
  "seed":              1,
  "disp_file":          "lcurve.out",
  "disp_freq":          100,
```

(continues on next page)

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

(continued from previous page)

```

    "save_freq": 1000
}

```

The sections `training_data` and `validation_data` give the training dataset and validation dataset, respectively. Taking the training dataset for example, the keys are explained below:

- `systems` provide paths of the training data systems. DeePMD-kit allows you to provide multiple systems with different numbers of atoms. This key can be a `list` or a `str`.
  - `list`: `systems` gives the training data systems.
  - `str`: `systems` should be a valid path. DeePMD-kit will recursively search all data systems in this path.
- At each training step, DeePMD-kit randomly picks `batch_size` frame(s) from one of the systems. The probability of using a system is by default in proportion to the number of batches in the system. More options are available for automatically determining the probability of using systems. One can set the key `auto_prob` to
  - `"prob_uniform"` all systems are used with the same probability.
  - `"prob_sys_size"` the probability of using a system is proportional to its size (number of frames).
  - `"prob_sys_size; idx_0:idx_0:w_0; idx_1:idx_1:w_1;..."` the list of systems is divided into blocks. Block `i` has systems ranging from `idx_i` to `idx_i+1`. The probability of using a system from block `i` is proportional to `w_i`. Within one block, the probability of using a system is proportional to its size.
- An example of using `"auto_prob"` is given below. The probability of using `systems[2]` is 0.4, and the sum of the probabilities of using `systems[0]` and `systems[1]` is 0.6. If the number of frames in `systems[1]` is twice of `system[0]`, then the probability of using `system[1]` is 0.4 and that of `system[0]` is 0.2.

```

"training_data": {
    "systems": ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
    "auto_prob": "prob_sys_size; 0:2:0.6; 2:3:0.4",
    "batch_size": "auto"
}

```

- The probability of using systems can also be specified explicitly with key `sys_probs` which is a list having the length of the number of systems. For example

```

"training_data": {
    "systems": ["../data_water/data_0/", "../data_water/data_1/", "../data_
↪water/data_2/"],
    "sys_probs": [0.5, 0.3, 0.2],
    "batch_size": "auto:32"
}

```

- The key `batch_size` specifies the number of frames used to train or validate the model in a training step. It can be set to
  - `list`: the length of which is the same as the systems. The batch size of each system is given by the elements of the list.
  - `int`: all systems use the same batch size.
  - `"auto"`: the same as `"auto:32"`, see `"auto:N"`



- "auto:N": automatically determines the batch size so that the `batch_size` times the number of atoms in the system is no less than N.
- The key `numb_batch` in `validate_data` gives the number of batches of model validation. Note that the batches may not be from the same system

The section `mixed_precision` specifies the mixed precision settings, which will enable the mixed precision training workflow for DeePMD-kit. The keys are explained below:

- `output_prec` precision used in the output tensors, only `float32` is supported currently.
- `compute_prec` precision used in the computing tensors, only `float16` is supported currently. Note there are several limitations about mixed precision training:
- Only `se_e2_a` type descriptor is supported by the mixed precision training workflow.
- The precision of the embedding net and the fitting net are forced to be set to `float32`.

Other keys in the `training` section are explained below:

- `numb_steps` The number of training steps.
- `seed` The random seed for getting frames from the training data set.
- `disp_file` The file for printing learning curve.
- `disp_freq` The frequency of printing learning curve. Set in the unit of training steps
- `save_freq` The frequency of saving checkpoint.

### 5.2.3 Options and environment variables

Several command line options can be passed to `dp train`, which can be checked with

```
$ dp train --help
```

An explanation will be provided

```
positional arguments:
  INPUT                the input json database

optional arguments:
  -h, --help            show this help message and exit

  --init-model INIT_MODEL
                        Initialize a model by the provided checkpoint

  --restart RESTART      Restart the training from the provided checkpoint

  --init-frz-model INIT_FRZ_MODEL
                        Initialize the training from the frozen model.

  --skip-neighbor-stat  Skip calculating neighbor statistics. Sel checking, automatic sel, and
  ↪ model compression will be disabled. (default: False)
```

`--init-model model.ckpt`, initializes the model training with an existing model that is stored in the path prefix of checkpoint files `model.ckpt`, the network architectures should match.

`--restart model.ckpt`, continues the training from the checkpoint `model.ckpt`.

`--init-frz-model frozen_model.pb`, initializes the training with an existing model that is stored in `frozen_model.pb`.

`--skip-neighbor-stat` will skip calculating neighbor statistics if one is concerned about performance. Some features will be disabled.

To maximize the performance, one should follow [FAQ: How to control the parallelism of a job](#) to control the number of threads.

One can set other environmental variables:

Environment variables	Allowed value	Default value	Usage
DP_INTERFACE_PRECISION	high, low	high	Control high (double) or low (float) precision of training.
DP_AUTO_PARALLELIZATION	0, 1	0	Enable auto parallelization for CPU operators.
DP_JIT	0, 1	0	Enable JIT. Note that this option may either improve or decrease the performance. Requires TensorFlow supports JIT.

## 5.2.4 Adjust *sel* of a frozen model

One can use `--init-frz-model` features to adjust (increase or decrease) *sel* of a existing model. Firstly, one needs to adjust *sel* in `input.json`. For example, adjust from `[46, 92]` to `[23, 46]`.

```
"model": {
  "descriptor": {
    "sel": [23, 46]
  }
}
```

To obtain the new model at once, *numb\_steps* should be set to zero:

```
"training": {
  "numb_steps": 0
}
```

Then, one can initialize the training from the frozen model and freeze the new model at once:

```
dp train input.json --init-frz-model frozen_model.pb
dp freeze -o frozen_model_adjusted_sel.pb
```

Two models should give the same result when the input satisfies both constraints.

Note: At this time, this feature is only supported by *se\_e2\_a* descriptor with *set\_davg\_true* enabled, or hybrid composed of the above descriptors.

## 5.3 Training Parameters

Note: One can load, modify, and export the input file by using our effective web-based tool [DP-GUI](#) online or hosted using the [command line interface](#) `dp_gui`. All training parameters below can be set in DP-GUI. By clicking “SAVE JSON”, one can download the input file for further training.

**model:**

type: dict

argument path: model

**type\_map:**

type: `typing.List[str]`, optional

argument path: model/type\_map

A list of strings. Give the name to each type of atoms. It is noted that the number of atom type of training system must be less than 128 in a GPU environment. If not given, `type.raw` in each system should use the same type indexes, and `type_map.raw` will take no effect.

**data\_stat\_nbatch:**

type: `int`, optional, default: 10

argument path: model/data\_stat\_nbatch

The model determines the normalization from the statistics of the data. This key specifies the number of frames in each system used for statistics.

**data\_stat\_protect:**

type: `float`, optional, default: 0.01

argument path: model/data\_stat\_protect

Protect parameter for atomic energy regression.

**data\_bias\_nsample:**

type: `int`, optional, default: 10

argument path: model/data\_bias\_nsample

The number of training samples in a system to compute and change the energy bias.

**use\_srtab:**

type: `str`, optional

argument path: model/use\_srtab

The table for the short-range pairwise interaction added on top of DP. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

**smin\_alpha:**

type: `float`, optional

argument path: model/smin\_alpha

The short-range tabulated interaction will be switched according to the distance of the nearest neighbor. This distance is calculated by `softmin`. This parameter

is the decaying parameter in the softmin. It is only required when `use_srtab` is provided.

**sw\_rmin:**

type: float, optional  
argument path: `model/sw_rmin`

The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

**sw\_rmax:**

type: float, optional  
argument path: `model/sw_rmax`

The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

**srtab\_add\_bias:**

type: bool, optional, default: True  
argument path: `model/srtab_add_bias`

Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when `use_srtab` is provided.

**type\_embedding:**

type: dict, optional  
argument path: `model/type_embedding`

The type embedding.

**neuron:**

type: `typing.List[int]`, optional, default: [8]  
argument path: `model/type_embedding/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**activation\_function:**

type: str, optional, default: tanh  
argument path: `model/type_embedding/activation_function`

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False  
argument path: `model/type_embedding/resnet_dt`

Whether to use a “Timestep” in the skip connection

**precision:**

type: str, optional, default: default  
argument path: `model/type_embedding/precision`

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: bool, optional, default: True  
 argument path: model/type\_embedding/trainable  
 If the parameters in the embedding net are trainable

**seed:**

type: `NoneType` | int, optional, default: None  
 argument path: model/type\_embedding/seed  
 Random seed for parameter initialization

**modifier:**

type: dict, optional  
 argument path: model/modifier  
 The modifier of model output.  
 Depending on the value of type, different sub args are accepted.

**type:**

type: str (flag key)  
 argument path: model/modifier/type  
 possible choices: *dipole\_charge*  
 The type of modifier. See explanation below.  
 -dipole\_charge: Use WFCC to model the electronic structure of the system. Correct the long-range interaction

When *type* is set to *dipole\_charge*:

**model\_name:**

type: str  
 argument path: model/modifier[dipole\_charge]/model\_name  
 The name of the frozen dipole model file.

**model\_charge\_map:**

type: typing.List[float]  
 argument path: model/modifier[dipole\_charge]/model\_charge\_map  
 The charge of the WFCC. The list length should be the same as the `'sel_type <model/fitting_net[dipole]/sel_type_>'`.

**sys\_charge\_map:**

type: typing.List[float]  
 argument path: model/modifier[dipole\_charge]/sys\_charge\_map  
 The charge of real atoms. The list length should be the same as the *type\_map*

**ewald\_beta:**

type: float, optional, default: 0.4

argument path: `model/modifier[dipole_charge]/ewald_beta`

The splitting parameter of Ewald sum. Unit is  $\text{\AA}^{-1}$

**ewald\_h:**

type: float, optional, default: 1.0

argument path: `model/modifier[dipole_charge]/ewald_h`

The grid spacing of the FFT grid. Unit is  $\text{\AA}$

**compress:**

type: dict, optional

argument path: `model/compress`

Model compression configurations

**spin:**

type: dict, optional

argument path: `model/spin`

The settings for systems with spin.

**use\_spin:**

type: `typing.List[bool]`

argument path: `model/spin/use_spin`

Whether to use atomic spin model for each atom type

**spin\_norm:**

type: `typing.List[float]`

argument path: `model/spin/spin_norm`

The magnitude of atomic spin for each atom type with spin

**virtual\_len:**

type: `typing.List[float]`

argument path: `model/spin/virtual_len`

The distance between virtual atom representing spin and its corresponding real atom for each atom type with spin

Depending on the value of type, different sub args are accepted.

**type:**

type: str (flag key), default: `standard`

argument path: `model/type`

possible choices: *standard*, *multi*, *frozen*, *pairtab*, *pairwise\_dprc*, *linear\_ener*

When `type` is set to `standard`:

Standard model, which contains a descriptor and a fitting.

**descriptor:**

type: dict

argument path: `model[standard]/descriptor`

The descriptor of atomic environment.

Depending on the value of type, different sub args are accepted.

**type:**

type: `str` (flag key)

argument path: `model[standard]/descriptor/type`

possible choices: `loc_frame`, `se_e2_a`, `se_e3`, `se_a_tpe`, `se_e2_r`,  
`hybrid`, `se_atten`, `se_atten_v2`, `se_a_ebd_v2`, `se_a_mask`

The type of the descriptor. See explanation below.

- `loc_frame`: Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.
- `se_e2_a`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor.
- `se_e2_r`: Used by the smooth edition of Deep Potential. Only the distance between atoms is used to construct the descriptor.
- `se_e3`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Three-body embedding will be used by this descriptor.
- `se_a_tpe`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Type embedding will be used by this descriptor.
- `se_atten`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Attention mechanism will be used by this descriptor.
- `se_atten_v2`: Used by the smooth edition of Deep Potential. The full relative coordinates are used to construct the descriptor. Attention mechanism with new modifications will be used by this descriptor.
- `se_a_mask`: Used by the smooth edition of Deep Potential. It can accept a variable number of atoms in a frame (Non-PBC system). aparam are required as an indicator matrix for the real/virtual sign of input atoms.
- `hybrid`: Concatenate of a list of descriptors as a new descriptor.

When `type` is set to `loc_frame`:

**sel\_a:**

type: `typing.List[int]`

argument path: `model[standard]/descriptor[loc_frame]/sel_a`

A list of integers. The length of the list should be the same as the number of atom types in the system. `sel_a[i]` gives the selected number of type-*i* neighbors. The full relative coordinates of the neighbors are used by the descriptor.

**sel\_r:**

type: `typing.List[int]`

argument path: `model[standard]/descriptor[loc_frame]/sel_r`

A list of integers. The length of the list should be the same as the number of atom types in the system. `sel_r[i]` gives the selected number of type-*i* neighbors. Only relative distance of the neighbors are used by the descriptor. `sel_a[i] + sel_r[i]` is recommended to be larger than the maximally possible number of type-*i* neighbors in the cut-off radius.

**rcut:**

type: `float`, optional, default: 6.0

argument path: `model[standard]/descriptor[loc_frame]/rcut`

The cut-off radius. The default value is 6.0

**axis\_rule:**

type: `typing.List[int]`

argument path:

`model[standard]/descriptor[loc_frame]/axis_rule`

A list of integers. The length should be 6 times of the number of types.

- `axis_rule[i*6+0]`: class of the atom defining the first axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance.
- `axis_rule[i*6+1]`: type of the atom defining the first axis of type-i atom.
- `axis_rule[i*6+2]`: index of the axis atom defining the first axis. Note that the neighbors with the same class and type are sorted according to their relative distance.
- `axis_rule[i*6+3]`: class of the atom defining the second axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance.
- `axis_rule[i*6+4]`: type of the atom defining the second axis of type-i atom.
- `axis_rule[i*6+5]`: index of the axis atom defining the second axis. Note that the neighbors with the same class and type are sorted according to their relative distance.

When `type` is set to `se_e2_a` (or its alias `se_a`):

**sel:**

type: `typing.List[int] | str`, optional, default: `auto`

argument path: `model[standard]/descriptor[se_e2_a]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- `List[int]`. The length of the list should be the same as the number of atom types in the system. `sel[i]` gives the selected number of type-i neighbors. `sel[i]` is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- `str`. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

**rcut:**

type: `float`, optional, default: 6.0

argument path: `model[standard]/descriptor[se_e2_a]/rcut`

The cut-off radius.

**rcut\_smth:**

type: `float`, optional, default: 0.5

argument path: `model[standard]/descriptor[se_e2_a]/rcut_smth`

Where to start smoothing. For example the  $1/r$  term is smoothed from `rcut` to `rcut_smth`

**neuron:**

type: `typing.List[int]`, optional, default: [10, 20, 40]



argument path: `model[standard]/descriptor[se_e2_a]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**axis\_neuron:**

type: `int`, optional, default: 4, alias: `n_axis_neuron`

argument path:

`model[standard]/descriptor[se_e2_a]/axis_neuron`

Size of the submatrix of  $G$  (embedding matrix).

**activation\_function:**

type: `str`, optional, default: `tanh`

argument path:

`model[standard]/descriptor[se_e2_a]/activation_function`

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: `bool`, optional, default: `False`

argument path: `model[standard]/descriptor[se_e2_a]/resnet_dt`

Whether to use a “Timestep” in the skip connection

**type\_one\_side:**

type: `bool`, optional, default: `False`

argument path:

`model[standard]/descriptor[se_e2_a]/type_one_side`

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{text}\{\text{types}\}}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{text}\{\text{types}\}}^2$  sets of embedding network parameters.

**precision:**

type: `str`, optional, default: `default`

argument path: `model[standard]/descriptor[se_e2_a]/precision`

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: `bool`, optional, default: `True`

argument path: `model[standard]/descriptor[se_e2_a]/trainable`

If the parameters in the embedding net is trainable

**seed:**

type: `NoneType` | `int`, optional

argument path: `model[standard]/descriptor[se_e2_a]/seed`

Random seed for parameter initialization

#### **exclude\_types:**

type: `typing.List[typing.List[int]]`, optional, default: `[]`

argument path:

`model[standard]/descriptor[se_e2_a]/exclude_types`

The excluded pairs of types which have no interaction with each other.

For example, `[[0, 1]]` means no interaction between type 0 and type 1.

#### **set\_davg\_zero:**

type: `bool`, optional, default: `False`

argument path:

`model[standard]/descriptor[se_e2_a]/set_davg_zero`

Set the normalization average to zero. This option should be set when `atom_ener` in the energy fitting is used

When `type` is set to `se_e3` (or its aliases `se_at`, `se_a_3be`, `se_t`):

#### **sel:**

type: `typing.List[int] | str`, optional, default: `auto`

argument path: `model[standard]/descriptor[se_e3]/sel`

This parameter set the number of selected neighbors for each type of atom. It can be:

- `List[int]`. The length of the list should be the same as the number of atom types in the system. `sel[i]` gives the selected number of type-*i* neighbors. `sel[i]` is recommended to be larger than the maximally possible number of type-*i* neighbors in the cut-off radius. It is noted that the total `sel` value must be less than 4096 in a GPU environment.
- `str`. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the `sel`. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

#### **rcut:**

type: `float`, optional, default: `6.0`

argument path: `model[standard]/descriptor[se_e3]/rcut`

The cut-off radius.

#### **rcut\_smth:**

type: `float`, optional, default: `0.5`

argument path: `model[standard]/descriptor[se_e3]/rcut_smth`

Where to start smoothing. For example the  $1/r$  term is smoothed from `rcut` to `rcut_smth`

#### **neuron:**

type: `typing.List[int]`, optional, default: `[10, 20, 40]`

argument path: `model[standard]/descriptor[se_e3]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**activation\_function:**

type: str, optional, default: tanh

argument path:

model[standard]/descriptor[se\_e3]/activation\_function

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False

argument path: model[standard]/descriptor[se\_e3]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**precision:**

type: str, optional, default: default

argument path: model[standard]/descriptor[se\_e3]/precision

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: bool, optional, default: True

argument path: model[standard]/descriptor[se\_e3]/trainable

If the parameters in the embedding net are trainable

**seed:**

type: NoneType | int, optional

argument path: model[standard]/descriptor[se\_e3]/seed

Random seed for parameter initialization

**set\_davg\_zero:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_e3]/set\_davg\_zero

Set the normalization average to zero. This option should be set when atom\_ener in the energy fitting is used

When [type](#) is set to `se_a_tpe` (or its alias `se_a_ebd`):

**sel:**

type: typing.List[int] | str, optional, default: auto

argument path: model[standard]/descriptor[se\_a\_tpe]/sel

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

**rcut:**

type: float, optional, default: 6.0

argument path: model[standard]/descriptor[se\_a\_tpe]/rcut

The cut-off radius.

**rcut\_smth:**

type: float, optional, default: 0.5

argument path: model[standard]/descriptor[se\_a\_tpe]/rcut\_smth

Where to start smoothing. For example the 1/r term is smoothed from rcut to rcut\_smth

**neuron:**

type: typing.List[int], optional, default: [10, 20, 40]

argument path: model[standard]/descriptor[se\_a\_tpe]/neuron

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**axis\_neuron:**

type: int, optional, default: 4, alias: n\_axis\_neuron

argument path:

model[standard]/descriptor[se\_a\_tpe]/axis\_neuron

Size of the submatrix of G (embedding matrix).

**activation\_function:**

type: str, optional, default: tanh

argument path:

model[standard]/descriptor[se\_a\_tpe]/activation\_function

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False

argument path: model[standard]/descriptor[se\_a\_tpe]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**type\_one\_side:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_a\_tpe]/type\_one\_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{types}}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{types}}^2$  sets of embedding network parameters.

**precision:**

type: str, optional, default: default

argument path: model[standard]/descriptor[se\_a\_tpe]/precision

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: bool, optional, default: True

argument path: model[standard]/descriptor[se\_a\_tpe]/trainable

If the parameters in the embedding net is trainable

**seed:**

type: NoneType | int, optional

argument path: model[standard]/descriptor[se\_a\_tpe]/seed

Random seed for parameter initialization

**exclude\_types:**

type: typing.List[typing.List[int]], optional, default: []

argument path:

model[standard]/descriptor[se\_a\_tpe]/exclude\_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

**set\_davg\_zero:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_a\_tpe]/set\_davg\_zero

Set the normalization average to zero. This option should be set when atom\_ener in the energy fitting is used

**type\_nchanl:**

type: int, optional, default: 4

argument path:

model[standard]/descriptor[se\_a\_tpe]/type\_nchanl

number of channels for type embedding

**type\_nlayer:**

type: int, optional, default: 2

argument path:  
`model[standard]/descriptor[se_a_tpe]/type_nlayer`  
 number of hidden layers of type embedding net

**numb\_aparam:**

type: `int`, optional, default: 0  
 argument path:  
`model[standard]/descriptor[se_a_tpe]/numb_aparam`  
 dimension of atomic parameter. if set to a value  $> 0$ , the atomic parameters are embedded.

When `type` is set to `se_e2_r` (or its alias `se_r`):

**sel:**

type: `typing.List[int] | str`, optional, default: `auto`  
 argument path: `model[standard]/descriptor[se_e2_r]/sel`  
 This parameter set the number of selected neighbors for each type of atom. It can be:

- `List[int]`. The length of the list should be the same as the number of atom types in the system. `sel[i]` gives the selected number of type-*i* neighbors. `sel[i]` is recommended to be larger than the maximally possible number of type-*i* neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- `str`. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

**rcut:**

type: `float`, optional, default: 6.0  
 argument path: `model[standard]/descriptor[se_e2_r]/rcut`  
 The cut-off radius.

**rcut\_smth:**

type: `float`, optional, default: 0.5  
 argument path: `model[standard]/descriptor[se_e2_r]/rcut_smth`  
 Where to start smoothing. For example the  $1/r$  term is smoothed from `rcut` to `rcut_smth`

**neuron:**

type: `typing.List[int]`, optional, default: [10, 20, 40]  
 argument path: `model[standard]/descriptor[se_e2_r]/neuron`  
 Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**activation\_function:**

type: `str`, optional, default: `tanh`  
 argument path:  
`model[standard]/descriptor[se_e2_r]/activation_function`

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False

argument path: model[standard]/descriptor[se\_e2\_r]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**type\_one\_side:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_e2\_r]/type\_one\_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{text}}\{\text{types}\}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{text}}\{\text{types}\}^2$  sets of embedding network parameters.

**precision:**

type: str, optional, default: default

argument path: model[standard]/descriptor[se\_e2\_r]/precision

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: bool, optional, default: True

argument path: model[standard]/descriptor[se\_e2\_r]/trainable

If the parameters in the embedding net are trainable

**seed:**

type: NoneType | int, optional

argument path: model[standard]/descriptor[se\_e2\_r]/seed

Random seed for parameter initialization

**exclude\_types:**

type: typing.List[typing.List[int]], optional, default: []

argument path:

model[standard]/descriptor[se\_e2\_r]/exclude\_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

**set\_davg\_zero:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_e2\_r]/set\_davg\_zero

Set the normalization average to zero. This option should be set when atom\_ener in the energy fitting is used

When `type` is set to `hybrid`:

**list:**

type: `list`

argument path: `model[standard]/descriptor[hybrid]/list`

A list of descriptor definitions

When `type` is set to `se_atten`:

**sel:**

type: `typing.List[int] | str | int`, optional, default: `auto`

argument path: `model[standard]/descriptor[se_atten]/sel`

This parameter set the number of selected neighbors. Note that this parameter is a little different from that in other descriptors. Instead of separating each type of atoms, only the summation matters. And this number is highly related with the efficiency, thus one should not make it too large. Usually 200 or less is enough, far away from the GPU limitation 4096. It can be:

- `int`. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- `List[int]`. The length of the list should be the same as the number of atom types in the system. `sel[i]` gives the selected number of type-*i* neighbors. Only the summation of `sel[i]` matters, and it is recommended to be less than 200. - `str`. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the `sel`. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

**rcut:**

type: `float`, optional, default: `6.0`

argument path: `model[standard]/descriptor[se_atten]/rcut`

The cut-off radius.

**rcut\_smth:**

type: `float`, optional, default: `0.5`

argument path: `model[standard]/descriptor[se_atten]/rcut_smth`

Where to start smoothing. For example the  $1/r$  term is smoothed from `rcut` to `rcut_smth`

**neuron:**

type: `typing.List[int]`, optional, default: `[10, 20, 40]`

argument path: `model[standard]/descriptor[se_atten]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**axis\_neuron:**

type: `int`, optional, default: `4`, alias: `n_axis_neuron`

argument path:

`model[standard]/descriptor[se_atten]/axis_neuron`



Size of the submatrix of G (embedding matrix).

**activation\_function:**

type: str, optional, default: tanh

argument path:

model[standard]/descriptor[se\_atten]/activation\_function

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False

argument path: model[standard]/descriptor[se\_atten]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**type\_one\_side:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_atten]/type\_one\_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{text}}\{\text{types}\}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{text}}\{\text{types}\}^2$  sets of embedding network parameters.

**precision:**

type: str, optional, default: default

argument path: model[standard]/descriptor[se\_atten]/precision

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: bool, optional, default: True

argument path: model[standard]/descriptor[se\_atten]/trainable

If the parameters in the embedding net is trainable

**seed:**

type: NoneType | int, optional

argument path: model[standard]/descriptor[se\_atten]/seed

Random seed for parameter initialization

**exclude\_types:**

type: typing.List[typing.List[int]], optional, default: []

argument path:

model[standard]/descriptor[se\_atten]/exclude\_types

The excluded pairs of types which have no interaction with each other. For example,  $[[0, 1]]$  means no interaction between type 0 and type 1.

**attn:**

type: int, optional, default: 128

argument path: model[standard]/descriptor[se\_atten]/attn

The length of hidden vectors in attention layers

**attn\_layer:**

type: int, optional, default: 2

argument path:

model[standard]/descriptor[se\_atten]/attn\_layer

The number of attention layers. Note that model compression of se\_atten is only enabled when attn\_layer==0 and stripped\_type\_embedding is True

**attn\_dotr:**

type: bool, optional, default: True

argument path: model[standard]/descriptor[se\_atten]/attn\_dotr

Whether to do dot product with the normalized relative coordinates

**attn\_mask:**

type: bool, optional, default: False

argument path: model[standard]/descriptor[se\_atten]/attn\_mask

Whether to do mask on the diagonal in the attention matrix

**stripped\_type\_embedding:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_atten]/stripped\_type\_embedding

Whether to strip the type embedding into a separated embedding network. Setting it to False will fall back to the previous version of se\_atten which is non-compressible.

**smooth\_type\_embdding:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_atten]/smooth\_type\_embdding

When using stripped type embedding, whether to dot smooth factor on the network output of type embedding to keep the network smooth, instead of setting set\_davg\_zero to be True.

**set\_davg\_zero:**

type: bool, optional, default: True

argument path:

model[standard]/descriptor[se\_atten]/set\_davg\_zero

Set the normalization average to zero. This option should be set when se\_atten descriptor or atom\_ener in the energy fitting is used

When `type` is set to `se_atten_v2`:

**sel:**

type: typing.List[int] | str | int, optional, default: auto

argument path: `model[standard]/descriptor[se_atten_v2]/sel`

This parameter set the number of selected neighbors. Note that this parameter is a little different from that in other descriptors. Instead of separating each type of atoms, only the summation matters. And this number is highly related with the efficiency, thus one should not make it too large. Usually 200 or less is enough, far away from the GPU limitation 4096. It can be:

- `int`. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- `List[int]`. The length of the list should be the same as the number of atom types in the system. `sel[i]` gives the selected number of type-*i* neighbors. Only the summation of `sel[i]` matters, and it is recommended to be less than 200. - `str`. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the `sel`. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

**rcut:**

type: `float`, optional, default: 6.0

argument path: `model[standard]/descriptor[se_atten_v2]/rcut`

The cut-off radius.

**rcut\_smth:**

type: `float`, optional, default: 0.5

argument path:

`model[standard]/descriptor[se_atten_v2]/rcut_smth`

Where to start smoothing. For example the  $1/r$  term is smoothed from `rcut` to `rcut_smth`

**neuron:**

type: `typing.List[int]`, optional, default: [10, 20, 40]

argument path: `model[standard]/descriptor[se_atten_v2]/neuron`

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**axis\_neuron:**

type: `int`, optional, default: 4, alias: `n_axis_neuron`

argument path:

`model[standard]/descriptor[se_atten_v2]/axis_neuron`

Size of the submatrix of *G* (embedding matrix).

**activation\_function:**

type: `str`, optional, default: `tanh`

argument path:

`model[standard]/descriptor[se_atten_v2]/activation_function`

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version,

and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_atten\_v2]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**type\_one\_side:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_atten\_v2]/type\_one\_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{text}\{\text{types}\}}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{text}\{\text{types}\}}^2$  sets of embedding network parameters.

**precision:**

type: str, optional, default: default

argument path:

model[standard]/descriptor[se\_atten\_v2]/precision

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: bool, optional, default: True

argument path:

model[standard]/descriptor[se\_atten\_v2]/trainable

If the parameters in the embedding net is trainable

**seed:**

type: NoneType | int, optional

argument path: model[standard]/descriptor[se\_atten\_v2]/seed

Random seed for parameter initialization

**exclude\_types:**

type: typing.List[typing.List[int]], optional, default: []

argument path:

model[standard]/descriptor[se\_atten\_v2]/exclude\_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

**attn:**

type: int, optional, default: 128

argument path: model[standard]/descriptor[se\_atten\_v2]/attn

The length of hidden vectors in attention layers

**attn\_layer:**

type: int, optional, default: 2

argument path:

model[standard]/descriptor[se\_atten\_v2]/attn\_layer

The number of attention layers. Note that model compression of se\_atten is only enabled when attn\_layer==0 and stripped\_type\_embedding is True

**attn\_dotr:**

type: bool, optional, default: True

argument path:

model[standard]/descriptor[se\_atten\_v2]/attn\_dotr

Whether to do dot product with the normalized relative coordinates

**attn\_mask:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_atten\_v2]/attn\_mask

Whether to do mask on the diagonal in the attention matrix

**set\_davg\_zero:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_atten\_v2]/set\_davg\_zero

Set the normalization average to zero. This option should be set when se\_atten descriptor or atom\_ener in the energy fitting is used

When [type](#) is set to `se_a_ebd_v2` (or its alias `se_a_tpe_v2`):

**sel:**

type: typing.List[int] | str, optional, default: auto

argument path: model[standard]/descriptor[se\_a\_ebd\_v2]/sel

This parameter set the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

**rcut:**

type: float, optional, default: 6.0

argument path: model[standard]/descriptor[se\_a\_ebd\_v2]/rcut

The cut-off radius.

**rcut\_smth:**

type: float, optional, default: 0.5

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/rcut\_smth

Where to start smoothing. For example the  $1/r$  term is smoothed from rcut to rcut\_smth

**neuron:**

type: typing.List[int], optional, default: [10, 20, 40]

argument path: model[standard]/descriptor[se\_a\_ebd\_v2]/neuron

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**axis\_neuron:**

type: int, optional, default: 4, alias: n\_axis\_neuron

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/axis\_neuron

Size of the submatrix of G (embedding matrix).

**activation\_function:**

type: str, optional, default: tanh

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/activation\_function

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**type\_one\_side:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/type\_one\_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{types}}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{types}}^2$  sets of embedding network parameters.

**precision:**

type: str, optional, default: default

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/precision

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**trainable:**

type: bool, optional, default: True

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/trainable

If the parameters in the embedding net is trainable

**seed:**

type: NoneType | int, optional

argument path: model[standard]/descriptor[se\_a\_ebd\_v2]/seed

Random seed for parameter initialization

**exclude\_types:**

type: typing.List[typing.List[int]], optional, default: []

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/exclude\_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

**set\_davg\_zero:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_a\_ebd\_v2]/set\_davg\_zero

Set the normalization average to zero. This option should be set when atom\_ener in the energy fitting is used

When `type` is set to `se_a_mask`:

**sel:**

type: typing.List[int] | str, optional, default: auto

argument path: model[standard]/descriptor[se\_a\_mask]/sel

This parameter sets the number of selected neighbors for each type of atom. It can be:

- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. sel[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius. It is noted that the total sel value must be less than 4096 in a GPU environment.
- str. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors within the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

**neuron:**

type: typing.List[int], optional, default: [10, 20, 40]

argument path: model[standard]/descriptor[se\_a\_mask]/neuron

Number of neurons in each hidden layers of the embedding net. When two layers are of the same size or one layer is twice as large as the previous layer, a skip connection is built.

**axis\_neuron:**

type: int, optional, default: 4, alias: n\_axis\_neuron

argument path:

model[standard]/descriptor[se\_a\_mask]/axis\_neuron

Size of the submatrix of G (embedding matrix).

**activation\_function:**

type: str, optional, default: tanh

argument path:

model[standard]/descriptor[se\_a\_mask]/activation\_function

The activation function in the embedding net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_a\_mask]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**type\_one\_side:**

type: bool, optional, default: False

argument path:

model[standard]/descriptor[se\_a\_mask]/type\_one\_side

If true, the embedding network parameters vary by types of neighbor atoms only, so there will be  $N_{\text{text}\{\text{types}\}}$  sets of embedding network parameters. Otherwise, the embedding network parameters vary by types of centric atoms and types of neighbor atoms, so there will be  $N_{\text{text}\{\text{types}\}}^2$  sets of embedding network parameters.

**exclude\_types:**

type: typing.List[typing.List[int]], optional, default: []

argument path:

model[standard]/descriptor[se\_a\_mask]/exclude\_types

The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

**precision:**

type: str, optional, default: default

argument path:

model[standard]/descriptor[se\_a\_mask]/precision

The precision of the embedding net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.



**trainable:**

type: bool, optional, default: True

argument path:

model[standard]/descriptor[se\_a\_mask]/trainable

If the parameters in the embedding net is trainable

**seed:**

type: NoneType | int, optional

argument path: model[standard]/descriptor[se\_a\_mask]/seed

Random seed for parameter initialization

**fitting\_net:**

type: dict

argument path: model[standard]/fitting\_net

The fitting of physical properties.

Depending on the value of type, different sub args are accepted.

**type:**

type: str (flag key), default: ener

argument path: model[standard]/fitting\_net/type

possible choices: *ener*, *dos*, *polar*, *dipole*

The type of the fitting. See explanation below.

- *ener*: Fit an energy model (potential energy surface).
- *dos* : Fit a density of states model. The total density of states / site-projected density of states labels should be provided by dos.npy or atom\_dos.npy in each data system. The file has number of frames lines and number of energy grid columns (times number of atoms in atom\_dos.npy). See loss parameter.
- *dipole*: Fit an atomic dipole model. Global dipole labels or atomic dipole labels for all the selected atoms (see sel\_type) should be provided by dipole.npy in each data system. The file either has number of frames lines and 3 times of number of selected atoms columns, or has number of frames lines and 3 columns. See loss parameter.
- *polar*: Fit an atomic polarizability model. Global polarizability labels or atomic polarizability labels for all the selected atoms (see sel\_type) should be provided by polarizability.npy in each data system. The file either has number of frames lines and 9 times of number of selected atoms columns, or has number of frames lines and 9 columns. See loss parameter.

When *type* is set to *ener*:

**numb\_fparam:**

type: int, optional, default: 0

argument path: model[standard]/fitting\_net[ener]/numb\_fparam

The dimension of the frame parameter. If set to >0, file fparam.npy should be included to provided the input fparams.

**numb\_aparam:**

type: int, optional, default: 0

argument path: model[standard]/fitting\_net[ener]/numb\_aparam

The dimension of the atomic parameter. If set to  $>0$ , file `aparam.npy` should be included to provide the input `aparams`.

**neuron:**

type: `typing.List[int]`, optional, default: `[120, 120, 120]`, alias: `n_neuron`

argument path: `model[standard]/fitting_net[ener]/neuron`

The number of neurons in each hidden layer of the fitting net. When two hidden layers are of the same size, a skip connection is built.

**activation\_function:**

type: `str`, optional, default: `tanh`

argument path:

`model[standard]/fitting_net[ener]/activation_function`

The activation function in the fitting net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**precision:**

type: `str`, optional, default: `default`

argument path: `model[standard]/fitting_net[ener]/precision`

The precision of the fitting net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**resnet\_dt:**

type: `bool`, optional, default: `True`

argument path: `model[standard]/fitting_net[ener]/resnet_dt`

Whether to use a “Timestep” in the skip connection

**trainable:**

type: `bool` | `typing.List[bool]`, optional, default: `True`

argument path: `model[standard]/fitting_net[ener]/trainable`

Whether the parameters in the fitting net are trainable. This option can be

- `bool`: `True` if all parameters of the fitting net are trainable, `False` otherwise.
- `list of bool`: Specifies if each layer is trainable. Since the fitting net is composed by hidden layers followed by an output layer, the length of this list should be equal to `len(neuron)+1`.

**rcond:**

type: `float` | `NoneType`, optional, default: `None`

argument path: `model[standard]/fitting_net[ener]/rcond`

The condition number used to determine the initial energy shift for each type of atoms. See `rcond` in `numpy.linalg.lstsq()` for more details.

**seed:**

type: `NoneType` | `int`, optional

argument path: `model[standard]/fitting_net[ener]/seed`

Random seed for parameter initialization of the fitting net

**atom\_ener:**

type: `typing.List[typing.Optional[float]]`, optional, default: `[]`

argument path: `model[standard]/fitting_net[ener]/atom_ener`

Specify the atomic energy in vacuum for each type

**layer\_name:**

type: `typing.List[str]`, optional

argument path: `model[standard]/fitting_net[ener]/layer_name`

The name of the each layer. The length of this list should be equal to `n_neuron + 1`. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters. The shape of these layers should be the same. If null is given for a layer, parameters will not be shared.

**use\_aparam\_as\_mask:**

type: `bool`, optional, default: `False`

argument path:

`model[standard]/fitting_net[ener]/use_aparam_as_mask`

Whether to use the aparam as a mask in input. If `True`, the aparam will not be used in fitting net for embedding. When `descript` is `se_a_mask`, the aparam will be used as a mask to indicate the input atom is real/virtual. And `use_aparam_as_mask` should be set to `True`.

When `type` is set to `dos`:

**numb\_fparam:**

type: `int`, optional, default: `0`

argument path: `model[standard]/fitting_net[dos]/numb_fparam`

The dimension of the frame parameter. If set to `>0`, file `fparam.npy` should be included to provided the input fparams.

**numb\_aparam:**

type: `int`, optional, default: `0`

argument path: `model[standard]/fitting_net[dos]/numb_aparam`

The dimension of the atomic parameter. If set to `>0`, file `aparam.npy` should be included to provided the input aparams.

**neuron:**

type: `typing.List[int]`, optional, default: `[120, 120, 120]`

argument path: `model[standard]/fitting_net[dos]/neuron`

The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

**activation\_function:**

type: `str`, optional, default: `tanh`

argument path:

`model[standard]/fitting_net[dos]/activation_function`

The activation function in the fitting net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**precision:**

type: str, optional, default: float64

argument path: model[standard]/fitting\_net[dos]/precision

The precision of the fitting net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**resnet\_dt:**

type: bool, optional, default: True

argument path: model[standard]/fitting\_net[dos]/resnet\_dt

Whether to use a “Timestep” in the skip connection

**trainable:**

type: bool | typing.List[bool], optional, default: True

argument path: model[standard]/fitting\_net[dos]/trainable

Whether the parameters in the fitting net are trainable. This option can be

- bool: True if all parameters of the fitting net are trainable, False otherwise.
- list of bool: Specifies if each layer is trainable. Since the fitting net is composed by hidden layers followed by a output layer, the length of this list should be equal to len(neuron)+1.

**rcond:**

type: float | NoneType, optional, default: None

argument path: model[standard]/fitting\_net[dos]/rcond

The condition number used to determine the initial energy shift for each type of atoms. See rcond in `numpy.linalg.lstsq()` for more details.

**seed:**

type: NoneType | int, optional

argument path: model[standard]/fitting\_net[dos]/seed

Random seed for parameter initialization of the fitting net

**numb\_dos:**

type: int, optional, default: 300

argument path: model[standard]/fitting\_net[dos]/numb\_dos

The number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

When `type` is set to polar:

**neuron:**

type: typing.List[int], optional, default: [120, 120, 120], alias: n\_neuron

argument path: `model[standard]/fitting_net[polar]/neuron`

The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

**activation\_function:**

type: `str`, optional, default: `tanh`

argument path:

`model[standard]/fitting_net[polar]/activation_function`

The activation function in the fitting net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: `bool`, optional, default: `True`

argument path: `model[standard]/fitting_net[polar]/resnet_dt`

Whether to use a “Timestep” in the skip connection

**precision:**

type: `str`, optional, default: `default`

argument path: `model[standard]/fitting_net[polar]/precision`

The precision of the fitting net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**fit\_diag:**

type: `bool`, optional, default: `True`

argument path: `model[standard]/fitting_net[polar]/fit_diag`

Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

**scale:**

type: `float` | `typing.List[float]`, optional, default: `1.0`

argument path: `model[standard]/fitting_net[polar]/scale`

The output of the fitting net (polarizability matrix) will be scaled by `scale`

**shift\_diag:**

type: `bool`, optional, default: `True`

argument path: `model[standard]/fitting_net[polar]/shift_diag`

Whether to shift the diagonal of polar, which is beneficial to training. Default is true.

**sel\_type:**

type: `typing.List[int]` | `NoneType` | `int`, optional, alias: `pol_type`

argument path: `model[standard]/fitting_net[polar]/sel_type`

The atom types for which the atomic polarizability will be provided. If not set, all types will be selected.

**seed:**

type: `NoneType` | `int`, optional  
argument path: `model[standard]/fitting_net[polar]/seed`  
Random seed for parameter initialization of the fitting net

When `type` is set to `dipole`:

**neuron:**

type: `typing.List[int]`, optional, default: `[120, 120, 120]`, alias: `n_neuron`  
argument path: `model[standard]/fitting_net[dipole]/neuron`  
The number of neurons in each hidden layers of the fitting net. When two hidden layers are of the same size, a skip connection is built.

**activation\_function:**

type: `str`, optional, default: `tanh`  
argument path: `model[standard]/fitting_net[dipole]/activation_function`  
The activation function in the fitting net. Supported activation functions are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”. Note that “gelu” denotes the custom operator version, and “gelu\_tf” denotes the TF standard version. If you set “None” or “none” here, no activation function will be used.

**resnet\_dt:**

type: `bool`, optional, default: `True`  
argument path: `model[standard]/fitting_net[dipole]/resnet_dt`  
Whether to use a “Timestep” in the skip connection

**precision:**

type: `str`, optional, default: `default`  
argument path: `model[standard]/fitting_net[dipole]/precision`  
The precision of the fitting net parameters, supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”. Default follows the interface precision.

**sel\_type:**

type: `typing.List[int]` | `NoneType` | `int`, optional, alias: `dipole_type`  
argument path: `model[standard]/fitting_net[dipole]/sel_type`  
The atom types for which the atomic dipole will be provided. If not set, all types will be selected.

**seed:**

type: `NoneType` | `int`, optional  
argument path: `model[standard]/fitting_net[dipole]/seed`  
Random seed for parameter initialization of the fitting net

When `type` is set to `multi`:

Multiple-task model.

**descriptor:**

type: dict  
 argument path: model[multi]/descriptor

The descriptor of atomic environment. See model[standard]/descriptor for details.

**fitting\_net\_dict:**

type: dict  
 argument path: model[multi]/fitting\_net\_dict

The dictionary of multiple fitting nets in multi-task mode. Each fitting\_net\_dict[fitting\_key] is the single definition of fitting of physical properties with user-defined name fitting\_key.

When type is set to frozen:

**model\_file:**

type: str  
 argument path: model[frozen]/model\_file

Path to the frozen model file.

When type is set to pairtab:

Pairwise tabulation energy model.

**tab\_file:**

type: str  
 argument path: model[pairtab]/tab\_file

Path to the tabulation file.

**rcut:**

type: float  
 argument path: model[pairtab]/rcut

The cut-off radius.

**sel:**

type: typing.List[int] | str | int  
 argument path: model[pairtab]/sel

This parameter set the number of selected neighbors. Note that this parameter is a little different from that in other descriptors. Instead of separating each type of atoms, only the summation matters. And this number is highly related with the efficiency, thus one should not make it too large. Usually 200 or less is enough, far away from the GPU limitation 4096. It can be:

- int. The maximum number of neighbor atoms to be considered. We recommend it to be less than 200.
- List[int]. The length of the list should be the same as the number of atom types in the system. sel[i] gives the selected number of type-i neighbors. Only the summation of sel[i] matters, and it is recommended to be less than 200. - str. Can be “auto:factor” or “auto”. “factor” is a float number larger than 1. This option will automatically determine the sel. In detail it counts the maximal number of neighbors with in the cutoff radius for each type of neighbor, then multiply the maximum by the “factor”. Finally the number is wrapped up to 4 divisible. The option “auto” is equivalent to “auto:1.1”.

When `type` is set to `pairwise_dprc`:

**qm\_model:**

type: dict

argument path: `model[pairwise_dprc]/qm_model`

**qmmm\_model:**

type: dict

argument path: `model[pairwise_dprc]/qmmm_model`

When `type` is set to `linear_ener`:

**models:**

type: list | dict

argument path: `model[linear_ener]/models`

The sub-models.

**weights:**

type: list | str

argument path: `model[linear_ener]/weights`

If the type is list of float, a list of weights for each model. If “mean”, the weights are set to be  $1 / \text{len}(\text{models})$ . If “sum”, the weights are set to be 1.

**learning\_rate:**

type: dict, optional

argument path: `learning_rate`

The definitio of learning rate

**scale\_by\_worker:**

type: str, optional, default: linear

argument path: `learning_rate/scale_by_worker`

When parallel training or batch size scaled, how to alter learning rate. Valid values are linear(default), 'sqrt or none.

Depending on the value of `type`, different sub args are accepted.

**type:**

type: str (flag key), default: `exp`

argument path: `learning_rate/type`

possible choices: `exp`

The type of the learning rate.

When `type` is set to `exp`:

**start\_lr:**

type: float, optional, default: 0.001

argument path: `learning_rate[exp]/start_lr`

The learning rate at the start of the training.

**stop\_lr:**

type: float, optional, default: 1e-08

argument path: `learning_rate[exp]/stop_lr`

The desired learning rate at the end of the training.



**decay\_steps:**

type: int, optional, default: 5000  
 argument path: `learning_rate[exp]/decay_steps`

The learning rate is decaying every this number of training steps.

**learning\_rate\_dict:**

type: dict, optional  
 argument path: `learning_rate_dict`

The dictionary of definitions of learning rates in multi-task mode. Each `learning_rate_dict[fitting_key]`, with user-defined name `fitting_key` in `model/fitting_net_dict`, is the single definition of learning rate.

**loss:**

type: dict, optional  
 argument path: `loss`

The definition of loss function. The loss type should be set to tensor, ener or left unset.

Depending on the value of type, different sub args are accepted.

**type:**

type: str (flag key), default: `ener`  
 argument path: `loss/type`  
 possible choices: *ener, ener\_spin, dos, tensor*

The type of the loss. When the fitting type is `ener`, the loss type should be set to `ener` or left unset. When the fitting type is `dipole` or `polar`, the loss type should be set to `tensor`.

When `type` is set to `ener`:

**start\_pref\_e:**

type: float | int, optional, default: 0.02  
 argument path: `loss[ener]/start_pref_e`

The prefactor of energy loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the energy label should be provided by file `energy.npy` in each data system. If both `start_pref_e` and `limit_pref_e` are set to 0, then the energy will be ignored.

**limit\_pref\_e:**

type: float | int, optional, default: 1.0  
 argument path: `loss[ener]/limit_pref_e`

The prefactor of energy loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_f:**

type: float | int, optional, default: 1000  
 argument path: `loss[ener]/start_pref_f`

The prefactor of force loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the force label should be provided by file `force.npy` in each data system. If both `start_pref_f` and `limit_pref_f` are set to 0, then the force will be ignored.

**limit\_pref\_f:**  
type: float | int, optional, default: 1.0  
argument path: `loss[ener]/limit_pref_f`  
The prefactor of force loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_v:**  
type: float | int, optional, default: 0.0  
argument path: `loss[ener]/start_pref_v`  
The prefactor of virial loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the virial label should be provided by file `virial.npy` in each data system. If both `start_pref_v` and `limit_pref_v` are set to 0, then the virial will be ignored.

**limit\_pref\_v:**  
type: float | int, optional, default: 0.0  
argument path: `loss[ener]/limit_pref_v`  
The prefactor of virial loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_ae:**  
type: float | int, optional, default: 0.0  
argument path: `loss[ener]/start_pref_ae`  
The prefactor of atomic energy loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the `atom_ener` label should be provided by file `atom_ener.npy` in each data system. If both `start_pref_ae` and `limit_pref_ae` are set to 0, then the atomic energy will be ignored.

**limit\_pref\_ae:**  
type: float | int, optional, default: 0.0  
argument path: `loss[ener]/limit_pref_ae`  
The prefactor of atomic energy loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_pf:**  
type: float | int, optional, default: 0.0  
argument path: `loss[ener]/start_pref_pf`  
The prefactor of atomic prefactor force loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the `atom_pref` label should be provided by file `atom_pref.npy` in each data system. If both `start_pref_pf` and `limit_pref_pf` are set to 0, then the atomic prefactor force will be ignored.

**limit\_pref\_pf:**  
type: float | int, optional, default: 0.0  
argument path: `loss[ener]/limit_pref_pf`  
The prefactor of atomic prefactor force loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**relative\_f:**  
type: float | NoneType, optional

argument path: `loss[ener]/relative_f`

If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by `relative_f`, i.e.  $DF_i / (\|F\| + \text{relative\_f})$  with  $DF$  denoting the difference between prediction and label and  $\|F\|$  denoting the L2 norm of the label.

**enable\_atom\_ener\_coeff:**

type: bool, optional, default: False

argument path: `loss[ener]/enable_atom_ener_coeff`

If true, the energy will be computed as  $\sum_i c_i E_i$ .  $c_i$  should be provided by file `atom_ener_coeff.npy` in each data system, otherwise it's 1.

**start\_pref\_gf:**

type: float, optional, default: 0.0

argument path: `loss[ener]/start_pref_gf`

The prefactor of generalized force loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the `drdq` label should be provided by file `drdq.npy` in each data system. If both `start_pref_gf` and `limit_pref_gf` are set to 0, then the generalized force will be ignored.

**limit\_pref\_gf:**

type: float, optional, default: 0.0

argument path: `loss[ener]/limit_pref_gf`

The prefactor of generalized force loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**numb\_generalized\_coord:**

type: int, optional, default: 0

argument path: `loss[ener]/numb_generalized_coord`

The dimension of generalized coordinates. Required when generalized force loss is used.

When `type` is set to `ener_spin`:

**start\_pref\_e:**

type: float | int, optional, default: 0.02

argument path: `loss[ener_spin]/start_pref_e`

The prefactor of energy loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the energy label should be provided by file `energy.npy` in each data system. If both `start_pref_energy` and `limit_pref_energy` are set to 0, then the energy will be ignored.

**limit\_pref\_e:**

type: float | int, optional, default: 1.0

argument path: `loss[ener_spin]/limit_pref_e`

The prefactor of energy loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_fr:**

type: float | int, optional, default: 1000

argument path: `loss[ener_spin]/start_pref_fr`

The prefactor of `force_real_atom` loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the `force_real_atom` label should be provided by file `force_real_atom.npy` in each data system. If both `start_pref_force_real_atom` and `limit_pref_force_real_atom` are set to 0, then the `force_real_atom` will be ignored.

**limit\_pref\_fr:**

type: `float | int`, optional, default: 1.0

argument path: `loss[ener_spin]/limit_pref_fr`

The prefactor of `force_real_atom` loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_fm:**

type: `float | int`, optional, default: 10000

argument path: `loss[ener_spin]/start_pref_fm`

The prefactor of `force_magnetic` loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the `force_magnetic` label should be provided by file `force_magnetic.npy` in each data system. If both `start_pref_force_magnetic` and `limit_pref_force_magnetic` are set to 0, then the `force_magnetic` will be ignored.

**limit\_pref\_fm:**

type: `float | int`, optional, default: 10.0

argument path: `loss[ener_spin]/limit_pref_fm`

The prefactor of `force_magnetic` loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_v:**

type: `float | int`, optional, default: 0.0

argument path: `loss[ener_spin]/start_pref_v`

The prefactor of `virial` loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the `virial` label should be provided by file `virial.npy` in each data system. If both `start_pref_virial` and `limit_pref_virial` are set to 0, then the `virial` will be ignored.

**limit\_pref\_v:**

type: `float | int`, optional, default: 0.0

argument path: `loss[ener_spin]/limit_pref_v`

The prefactor of `virial` loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_ae:**

type: `float | int`, optional, default: 0.0

argument path: `loss[ener_spin]/start_pref_ae`

The prefactor of `atom_ener` loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the `atom_ener` label should be provided by file `atom_ener.npy` in each data system. If both `start_pref_atom_ener` and `limit_pref_atom_ener` are set to 0, then the `atom_ener` will be ignored.

**limit\_pref\_ae:**

type: float | int, optional, default: 0.0

argument path: `loss[ener_spin]/limit_pref_ae`

The prefactor of atom\_ener loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_pf:**

type: float | int, optional, default: 0.0

argument path: `loss[ener_spin]/start_pref_pf`

The prefactor of atom\_pref loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atom\_pref label should be provided by file `atom_pref.npy` in each data system. If both `start_pref_atom_pref` and `limit_pref_atom_pref` are set to 0, then the atom\_pref will be ignored.

**limit\_pref\_pf:**

type: float | int, optional, default: 0.0

argument path: `loss[ener_spin]/limit_pref_pf`

The prefactor of atom\_pref loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**relative\_f:**

type: float | NoneType, optional

argument path: `loss[ener_spin]/relative_f`

If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by `relative_f`, i.e.  $DF_i / (\|F\| + \text{relative\_f})$  with  $DF$  denoting the difference between prediction and label and  $\|F\|$  denoting the L2 norm of the label.

**enable\_atom\_ener\_coeff:**

type: bool, optional, default: False

argument path: `loss[ener_spin]/enable_atom_ener_coeff`

If true, the energy will be computed as  $\sum_i c_i E_i$ .  $c_i$  should be provided by file `atom_ener_coeff.npy` in each data system, otherwise it's 1.

When `type` is set to `dos`:

**start\_pref\_dos:**

type: float | int, optional, default: 0.0

argument path: `loss[dos]/start_pref_dos`

The prefactor of Density of State (DOS) loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the Density of State (DOS) label should be provided by file `Density of State (DOS).npy` in each data system. If both `start_pref_Density of State (DOS)` and `limit_pref_Density of State (DOS)` are set to 0, then the Density of State (DOS) will be ignored.

**limit\_pref\_dos:**

type: float | int, optional, default: 0.0

argument path: `loss[dos]/limit_pref_dos`

The prefactor of Density of State (DOS) loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_cdf:**

type: float | int, optional, default: 0.0  
argument path: `loss[dos]/start_pref_cdf`

The prefactor of Cumulative Distribution Function (cumulative intergral of DOS) loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the Cumulative Distribution Function (cumulative intergral of DOS) label should be provided by file Cumulative Distribution Function (cumulative intergral of DOS).npz in each data system. If both `start_pref_Cumulative Distribution Function (cumulative intergral of DOS)` and `limit_pref_Cumulative Distribution Function (cumulative intergral of DOS)` are set to 0, then the Cumulative Distribution Function (cumulative intergral of DOS) will be ignored.

**limit\_pref\_cdf:**

type: float | int, optional, default: 0.0  
argument path: `loss[dos]/limit_pref_cdf`

The prefactor of Cumulative Distribution Function (cumulative intergral of DOS) loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_ados:**

type: float | int, optional, default: 1.0  
argument path: `loss[dos]/start_pref_ados`

The prefactor of atomic DOS (site-projected DOS) loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the atomic DOS (site-projected DOS) label should be provided by file atomic DOS (site-projected DOS).npz in each data system. If both `start_pref_atomic DOS (site-projected DOS)` and `limit_pref_atomic DOS (site-projected DOS)` are set to 0, then the atomic DOS (site-projected DOS) will be ignored.

**limit\_pref\_ados:**

type: float | int, optional, default: 1.0  
argument path: `loss[dos]/limit_pref_ados`

The prefactor of atomic DOS (site-projected DOS) loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

**start\_pref\_acdf:**

type: float | int, optional, default: 0.0  
argument path: `loss[dos]/start_pref_acdf`

The prefactor of Cumulative integral of atomic DOS loss at the start of the training. Should be larger than or equal to 0. If set to none-zero value, the Cumulative integral of atomic DOS label should be provided by file Cumulative integral of atomic DOS.npz in each data system. If both `start_pref_Cumulative integral of atomic DOS` and `limit_pref_Cumulative integral of atomic DOS` are set to 0, then the Cumulative integral of atomic DOS will be ignored.

**limit\_pref\_acdf:**

type: float | int, optional, default: 0.0  
argument path: `loss[dos]/limit_pref_acdf`

The prefactor of Cumulative integral of atomic DOS loss at the limit of the training, Should be larger than or equal to 0. i.e. the training step goes to infinity.

When `type` is set to `tensor`:

**pref:**

type: float | int

argument path: `loss[tensor]/pref`

The prefactor of the weight of global loss. It should be larger than or equal to 0. It controls the weight of loss corresponding to global label, i.e. 'polarizability.npy' or 'dipole.npy', whose shape should be #frames x [9 or 3]. If it's larger than 0.0, this npy should be included.

**pref\_atomic:**

type: float | int

argument path: `loss[tensor]/pref_atomic`

The prefactor of the weight of atomic loss. It should be larger than or equal to 0. It controls the weight of loss corresponding to atomic label, i.e. 'atomic\_polarizability.npy' or 'atomic\_dipole.npy', whose shape should be #frames x ([9 or 3] x #selected atoms). If it's larger than 0.0, this npy should be included. Both `pref` and `pref_atomic` should be provided, and either can be set to 0.0.

**loss\_dict:**

type: dict, optional

argument path: `loss_dict`

The dictionary of definitions of multiple loss functions in multi-task mode. Each `loss_dict[fitting_key]`, with user-defined name `fitting_key` in `model/fitting_net_dict`, is the single definition of loss function, whose type should be set to tensor, ener or left unset.

**training:**

type: dict

argument path: `training`

The training options.

**training\_data:**

type: dict, optional

argument path: `training/training_data`

Configurations of training data.

**systems:**

type: `typing.List[str]` | str

argument path: `training/training_data/systems`

The data systems for training. This key can be provided with a list that specifies the systems, or be provided with a string by which the prefix of all systems are given and the list of the systems is automatically generated.

**set\_prefix:**

type: str, optional, default: set

argument path: `training/training_data/set_prefix`

The prefix of the sets in the [systems](#).

**batch\_size:**

type: `typing.List[int]` | str | int, optional, default: auto

argument path: `training/training_data/batch_size`

This key can be

- list: the length of which is the same as the `systems`. The batch size of each system is given by the elements of the list.
- int: all `systems` use the same batch size.
- string “auto”: automatically determines the batch size so that the `batch_size` times the number of atoms in the system is no less than 32.
- string “auto:N”: automatically determines the batch size so that the `batch_size` times the number of atoms in the system is no less than N.
- string “mixed:N”: the batch data will be sampled from all systems and merged into a mixed system with the batch size N. Only support the `se_atten` descriptor.

If MPI is used, the value should be considered as the batch size per task.

#### **auto\_prob:**

type: `str`, optional, default: `prob_sys_size`, alias: `auto_prob_style`  
 argument path: `training/training_data/auto_prob`

Determine the probability of systems automatically. The method is assigned by this key and can be

- “prob\_uniform” : the probability all the systems are equal, namely  $1.0/\text{self.get\_nsystems}()$
- “prob\_sys\_size” : the probability of a system is proportional to the number of batches in the system
- “prob\_sys\_size;stt\_idx:end\_idx:weight;stt\_idx:end\_idx:weight;...” : the list of systems is divided into blocks. A block is specified by `stt_idx:end_idx:weight`, where `stt_idx` is the starting index of the system, `end_idx` is then ending (not including) index of the system, the probabilities of the systems in this block sums up to weight, and the relatively probabilities within this block is proportional to the number of batches in the system.

#### **sys\_probs:**

type: `typing.List[float] | NoneType`, optional, default: `None`, alias: `sys_weights`  
 argument path: `training/training_data/sys_probs`

A list of float if specified. Should be of the same length as `systems`, specifying the probability of each system.

#### **validation\_data:**

type: `dict | NoneType`, optional, default: `None`  
 argument path: `training/validation_data`

Configurations of validation data. Similar to that of training data, except that a `numb_btch` argument may be configured

#### **systems:**

type: `typing.List[str] | str`  
 argument path: `training/validation_data/systems`

The data systems for validation. This key can be provided with a list that specifies the systems, or be provided with a string by which the prefix of all systems are given and the list of the systems is automatically generated.

#### **set\_prefix:**

type: `str`, optional, default: `set`  
 argument path: `training/validation_data/set_prefix`



The prefix of the sets in the [systems](#).

**batch\_size:**

type: `typing.List[int] | str | int`, optional, default: `auto`  
 argument path: `training/validation_data/batch_size`

This key can be

- list: the length of which is the same as the [systems](#). The batch size of each system is given by the elements of the list.
- int: all [systems](#) use the same batch size.
- string “auto”: automatically determines the batch size so that the `batch_size` times the number of atoms in the system is no less than 32.
- string “auto:N”: automatically determines the batch size so that the `batch_size` times the number of atoms in the system is no less than N.

**auto\_prob:**

type: `str`, optional, default: `prob_sys_size`, alias: `auto_prob_style`  
 argument path: `training/validation_data/auto_prob`

Determine the probability of systems automatically. The method is assigned by this key and can be

- “prob\_uniform” : the probability all the systems are equal, namely `1.0/self.get_nsystems()`
- “prob\_sys\_size” : the probability of a system is proportional to the number of batches in the system
- “prob\_sys\_size;stt\_idx:end\_idx:weight;stt\_idx:end\_idx:weight;...” : the list of systems is divided into blocks. A block is specified by `stt_idx:end_idx:weight`, where `stt_idx` is the starting index of the system, `end_idx` is then ending (not including) index of the system, the probabilities of the systems in this block sums up to `weight`, and the relatively probabilities within this block is proportional to the number of batches in the system.

**sys\_probs:**

type: `typing.List[float] | NoneType`, optional, default: `None`, alias: `sys_weights`  
 argument path: `training/validation_data/sys_probs`

A list of float if specified. Should be of the same length as `systems`, specifying the probability of each system.

**numb\_btch:**

type: `int`, optional, default: `1`, alias: `numb_batch`  
 argument path: `training/validation_data/numb_btch`

An integer that specifies the number of batches to be sampled for each validation period.

**mixed\_precision:**

type: `dict`, optional  
 argument path: `training/mixed_precision`

Configurations of mixed precision.

**output\_prec:**

type: `str`, optional, default: `float32`

argument path: `training/mixed_precision/output_prec`

The precision for mixed precision params. ” “The trainable variables precision during the mixed precision training process, ” “supported options are float32 only currently.

**compute\_prec:**

type: `str`

argument path: `training/mixed_precision/compute_prec`

The precision for mixed precision compute. ” “The compute precision during the mixed precision training process, “” “supported options are float16 and bfloat16 currently.

**numb\_steps:**

type: `int`, alias: `stop_batch`

argument path: `training/numb_steps`

Number of training batch. Each training uses one batch of data.

**seed:**

type: `NoneType` | `int`, optional

argument path: `training/seed`

The random seed for getting frames from the training data set.

**disp\_file:**

type: `str`, optional, default: `lcurve.out`

argument path: `training/disp_file`

The file for printing learning curve.

**disp\_freq:**

type: `int`, optional, default: 1000

argument path: `training/disp_freq`

The frequency of printing learning curve.

**save\_freq:**

type: `int`, optional, default: 1000

argument path: `training/save_freq`

The frequency of saving check point.

**save\_ckpt:**

type: `str`, optional, default: `model.ckpt`

argument path: `training/save_ckpt`

The path prefix of saving check point files.

**disp\_training:**

type: `bool`, optional, default: `True`

argument path: `training/disp_training`

Displaying verbose information during training.

**time\_training:**

type: `bool`, optional, default: `True`

argument path: `training/time_training`  
Timing during training.

**profiling:**  
type: bool, optional, default: False  
argument path: `training/profiling`  
Profiling during training.

**profiling\_file:**  
type: str, optional, default: `timeline.json`  
argument path: `training/profiling_file`  
Output file for profiling.

**enable\_profiler:**  
type: bool, optional, default: False  
argument path: `training/enable_profiler`  
Enable TensorFlow Profiler (available in TensorFlow 2.3) to analyze performance. The log will be saved to `tensorboard_log_dir`.

**tensorboard:**  
type: bool, optional, default: False  
argument path: `training/tensorboard`  
Enable tensorboard

**tensorboard\_log\_dir:**  
type: str, optional, default: `log`  
argument path: `training/tensorboard_log_dir`  
The log directory of tensorboard outputs

**tensorboard\_freq:**  
type: int, optional, default: 1  
argument path: `training/tensorboard_freq`  
The frequency of writing tensorboard events.

**data\_dict:**  
type: dict, optional  
argument path: `training/data_dict`  
The dictionary of multi DataSystems in multi-task mode. Each `data_dict[fitting_key]`, with user-defined name `fitting_key` in `model/fitting_net_dict`, contains training data and optional validation data definitions.

**fitting\_weight:**  
type: dict, optional  
argument path: `training/fitting_weight`  
Each `fitting_weight[fitting_key]`, with user-defined name `fitting_key` in `model/fitting_net_dict`, is the training weight of fitting net `fitting_key`. Fitting nets with higher weights will be selected with higher probabilities to be trained in one step. Weights will be normalized and minus ones will be ignored. If not set, each fitting net will be equally selected when training.

**nvnmmd:**

type: dict, optional  
argument path: `nvnmmd`

The nvnmmd options.

**version:**

type: int  
argument path: `nvnmmd/version`  
configuration the nvnmmd version (0 | 1), 0 for 4 types, 1 for 32 types

**max\_nnei:**

type: int  
argument path: `nvnmmd/max_nnei`  
configuration the max number of neighbors, 128|256 for version 0, 128 for version 1

**net\_size:**

type: int  
argument path: `nvnmmd/net_size`  
configuration the number of nodes of fitting\_net, just can be set as 128

**map\_file:**

type: str  
argument path: `nvnmmd/map_file`  
A file containing the mapping tables to replace the calculation of embedding nets

**config\_file:**

type: str  
argument path: `nvnmmd/config_file`  
A file containing the parameters about how to implement the model in certain hardware

**weight\_file:**

type: str  
argument path: `nvnmmd/weight_file`  
a \*.npz file containing the weights of the model

**enable:**

type: bool  
argument path: `nvnmmd/enable`  
enable the nvnmmd training

**restore\_descriptor:**

type: bool  
argument path: `nvnmmd/restore_descriptor`  
enable to restore the parameter of embedding\_net from weight.npz

```

restore_fitting_net:
    type: bool
    argument path: nvnmmd/restore_fitting_net
    enable to restore the parameter of fitting_net from weight.npy

quantize_descriptor:
    type: bool
    argument path: nvnmmd/quantize_descriptor
    enable the quantization of descriptor

quantize_fitting_net:
    type: bool
    argument path: nvnmmd/quantize_fitting_net
    enable the quantization of fitting_net

```

## 5.4 Parallel training

Currently, parallel training is enabled in a synchronized way with help of [Horovod](#). Depending on the number of training processes (according to MPI context) and the number of GPU cards available, DeePMD-kit will decide whether to launch the training in parallel (distributed) mode or in serial mode. Therefore, no additional options are specified in your JSON/YAML input file.

### 5.4.1 Tuning learning rate

Horovod works in the data-parallel mode, resulting in a larger global batch size. For example, the real batch size is 8 when `batch_size` is set to 2 in the input file and you launch 4 workers. Thus, `learning_rate` is automatically scaled by the number of workers for better convergence. Technical details of such heuristic rule are discussed at [Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour](#).

The number of decay steps required to achieve the same accuracy can decrease by the number of cards (e.g., 1/2 of steps in the above case), but needs to be scaled manually in the input file.

In some cases, it won't work well when scaling the learning rate by worker count in a linear way. Then you can try `sqrt` or `none` by setting argument `scale_by_worker` like below.

```

"learning_rate" :{
  "scale_by_worker": "none",
  "type": "exp"
}

```

### 5.4.2 Scaling test

Testing `examples/water/se_e2_a` on an 8-GPU host, linear acceleration can be observed with the increasing number of cards.

Num of GPU cards	Seconds every 100 samples	Samples per second	Speed up
1	1.4515	68.89	1.00
2	1.5962	62.65*2	1.82
4	1.7635	56.71*4	3.29
8	1.7267	57.91*8	6.72

### 5.4.3 How to use

Training workers can be launched with `horovodrun`. The following command launches 4 processes on the same host:

```
CUDA_VISIBLE_DEVICES=4,5,6,7 horovodrun -np 4 \
  dp train --mpi-log=workers input.json
```

Need to mention, the environment variable `CUDA_VISIBLE_DEVICES` must be set to control parallelism on the occupied host where one process is bound to one GPU card.

To maximize the performance, one should follow [FAQ: How to control the parallelism of a job](#) to control the number of threads.

When using MPI with Horovod, `horovodrun` is a simple wrapper around `mpirun`. In the case where fine-grained control over options is passed to `mpirun`, `mpirun` can be invoked directly, and it will be detected automatically by Horovod, e.g.,

```
CUDA_VISIBLE_DEVICES=4,5,6,7 mpirun -l -launcher=fork -hosts=localhost -np 4 \
  dp train --mpi-log=workers input.json
```

this is sometimes necessary for an HPC environment.

Whether distributed workers are initiated can be observed in the “Summary of the training” section in the log (world size > 1, and distributed).

```
[0] DEEPMO INFO    ---Summary of the training-----
[0] DEEPMO INFO    distributed
[0] DEEPMO INFO    world size:           4
[0] DEEPMO INFO    my rank:              0
[0] DEEPMO INFO    node list:           ['exp-13-57']
[0] DEEPMO INFO    running on:          exp-13-57
[0] DEEPMO INFO    computing device:    gpu:0
[0] DEEPMO INFO    CUDA_VISIBLE_DEVICES: 0,1,2,3
[0] DEEPMO INFO    Count of visible GPU: 4
[0] DEEPMO INFO    num_intra_threads:    0
[0] DEEPMO INFO    num_inter_threads:    0
[0] DEEPMO INFO    -----
```

### 5.4.4 Logging

What’s more, 2 command-line arguments are defined to control the logging behavior when performing parallel training with MPI.

```
optional arguments:
  -l LOG_PATH, --log-path LOG_PATH
                        set log file to log messages to disk, if not
                        specified, the logs will only be output to console
                        (default: None)
  -m {master,collect,workers}, --mpi-log {master,collect,workers}
                        Set the manner of logging when running with MPI.
                        'master' logs only on main process, 'collect'
                        broadcasts logs from workers to master and 'workers'
                        means each process will output its own log (default:
                        master)
```

## 5.5 Multi-task training

### 5.5.1 Theory

The multi-task training process can simultaneously handle different datasets with properties that cannot be fitted in one network (e.g. properties from DFT calculations under different exchange-correlation functionals or different basis sets). These datasets are denoted by  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n_t)}$ . For each dataset, a training task is defined as

$$\min_{\boldsymbol{\theta}} L^{(t)}(\mathbf{x}^{(t)}; \boldsymbol{\theta}^{(t)}, \tau), \quad t = 1, \dots, n_t.$$

During the multi-task training process, all tasks share one descriptor with trainable parameters  $\boldsymbol{\theta}_d$ , while each of them has its own fitting network with trainable parameters  $\boldsymbol{\theta}_f^{(t)}$ , thus  $\boldsymbol{\theta}^{(t)} = \{\boldsymbol{\theta}_d, \boldsymbol{\theta}_f^{(t)}\}$ . At each training step, a task is randomly picked from  $1, \dots, n_t$ , and the Adam optimizer is executed to minimize  $L^{(t)}$  for one step to update the parameter  $\boldsymbol{\theta}^{(t)}$ . If different fitting networks have the same architecture, they can share the parameters of some layers to improve training efficiency.<sup>1</sup>

### 5.5.2 Perform the multi-task training

Training on multiple data sets (each data set contains several data systems) can be performed in multi-task mode, with one common descriptor and multiple specific fitting nets for each data set. One can simply switch the following parameters in training input script to perform multi-task mode:

- `fitting_net`  $\rightarrow$  `fitting_net_dict`, each key of which can be one individual fitting net.
- `training_data`, `validation_data`  $\rightarrow$  `data_dict`, each key of which can be one individual data set contains several data systems for corresponding fitting net, the keys must be consistent with those in `fitting_net_dict`.
- `loss`  $\rightarrow$  `loss_dict`, each key of which can be one individual loss setting for corresponding fitting net, the keys must be consistent with those in `fitting_net_dict`, if not set, the corresponding fitting net will use the default loss.
- (Optional) `fitting_weight`, each key of which can be a non-negative integer or float, deciding the chosen probability for corresponding fitting net in training, if not set or invalid, the corresponding fitting net will not be used.

The training procedure will automatically choose single-task or multi-task mode, based on the above parameters. Note that parameters of single-task mode and multi-task mode can not be mixed.

An example input for training energy and dipole in water system can be found here: multi-task input on water.

The supported descriptors for multi-task mode are listed:

- `se_a` (`se_e2_a`)
- `se_r` (`se_e2_r`)
- `se_at` (`se_e3`)
- `se_atten`

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yiibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

- se\_attn\_v2
- hybrid

The supported fitting nets for multi-task mode are listed:

- ener
- dipole
- polar

The output of `dp freeze` command in multi-task mode can be seen in [freeze command](#).

### 5.5.3 Initialization from pretrained multi-task model

For advance training in multi-task mode, one can first train the descriptor on several upstream datasets and then transfer it on new downstream ones with newly added fitting nets. At the second step, you can also inherit some fitting nets trained on upstream datasets, by merely adding fitting net keys in `fitting_net_dict` and optional fitting net weights in `fitting_weight`.

Take multi-task input on water again for example. You can first train a multi-task model using input script with the following `model` part:

```
"model": {
  "type_map": ["O", "H"],
  "descriptor": {
    "type": "se_e2_a",
    "sel": [46, 92],
    "rcut_smth": 0.5,
    "rcut": 6.0,
    "neuron": [25, 50, 100],
  },
  "fitting_net_dict": {
    "water_dipole": {
      "type": "dipole",
      "neuron": [100, 100, 100],
    },
    "water_ener": {
      "neuron": [240, 240, 240],
      "resnet_dt": true,
    }
  }
}
```

After training, you can freeze this multi-task model into one unit graph:

```
$ dp freeze -o graph.pb --united-model
```

Then if you want to transfer the trained descriptor and some fitting nets (take `water_ener` for example) to newly added datasets with new fitting net `water_ener_2`, you can modify the `model` part of the new input script in a more simplified way:

```
"model": {
  "type_map": ["O", "H"],
  "descriptor": {},
  "fitting_net_dict": {
    "water_ener": {},
```

(continues on next page)



(continued from previous page)

```

        "water_ener_2": {
            "neuron":    [240, 240, 240],
            "resnet_dt":  true,
        }
    },
}

```

It will autocomplete the configurations according to the frozen graph.

Note that for newly added fitting net keys, other parts in the input script, including `data_dict` and `loss_dict` (optionally `fitting_weight`), should be set explicitly. While for old fitting net keys, it will inherit the old configurations if not set.

Finally, you can perform the modified multi-task training from the frozen model with command:

```
$ dp train input.json --init_frz_model graph.pb
```

### 5.5.4 Share layers among energy fitting networks

The multi-task training can be used to train multiple levels of energies (e.g. DFT and CCSD(T)) at the same time. In this situation, one can set `model/fitting_net[ener]/layer_name>` to share some of layers among fitting networks. The architecture of the layers with the same name should be the same.

For example, if one want to share the first and the third layers for two three-hidden-layer fitting networks, the following parameters should be set.

```

"fitting_net_dict": {
    "ccsd": {
        "neuron": [
            240,
            240,
            240
        ],
        "layer_name": ["l0", null, "l2", null]
    },
    "wb97m": {
        "neuron": [
            240,
            240,
            240
        ],
        "layer_name": ["l0", null, "l2", null]
    }
}

```

## 5.6 TensorBoard Usage

TensorBoard provides the visualization and tooling needed for machine learning experimentation. Full instructions for TensorBoard can be found [here](#).

### 5.6.1 Highlighted features

DeePMD-kit can now use most of the interesting features enabled by TensorBoard!

- Tracking and visualizing metrics, such as `l2_loss`, `l2_energy_loss` and `l2_force_loss`
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time.
- Viewing summaries of trainable variables

### 5.6.2 How to use Tensorboard with DeePMD-kit

Before running TensorBoard, make sure you have generated summary data in a log directory by modifying the input script, setting `tensorboard` to true in the training subsection will enable the TensorBoard data analysis. eg. `water_se_a.json`.

```
"training" : {
  "systems":      ["../data/"],
  "set_prefix":    "set",
  "stop_batch":    1000000,
  "batch_size":    1,

  "seed":          1,

  "_comment": " display and restart",
  "_comment": " frequencies counted in batch",
  "disp_file":     "lcurve.out",
  "disp_freq":     100,
  "numb_test":     10,
  "save_freq":     1000,
  "save_ckpt":     "model.ckpt",

  "disp_training":true,
  "time_training":true,
  "tensorboard":   true,
  "tensorboard_log_dir":"log",
  "tensorboard_freq": 1000,
  "profiling":     false,
  "profiling_file":"timeline.json",
  "_comment":     "that's all"
}
```

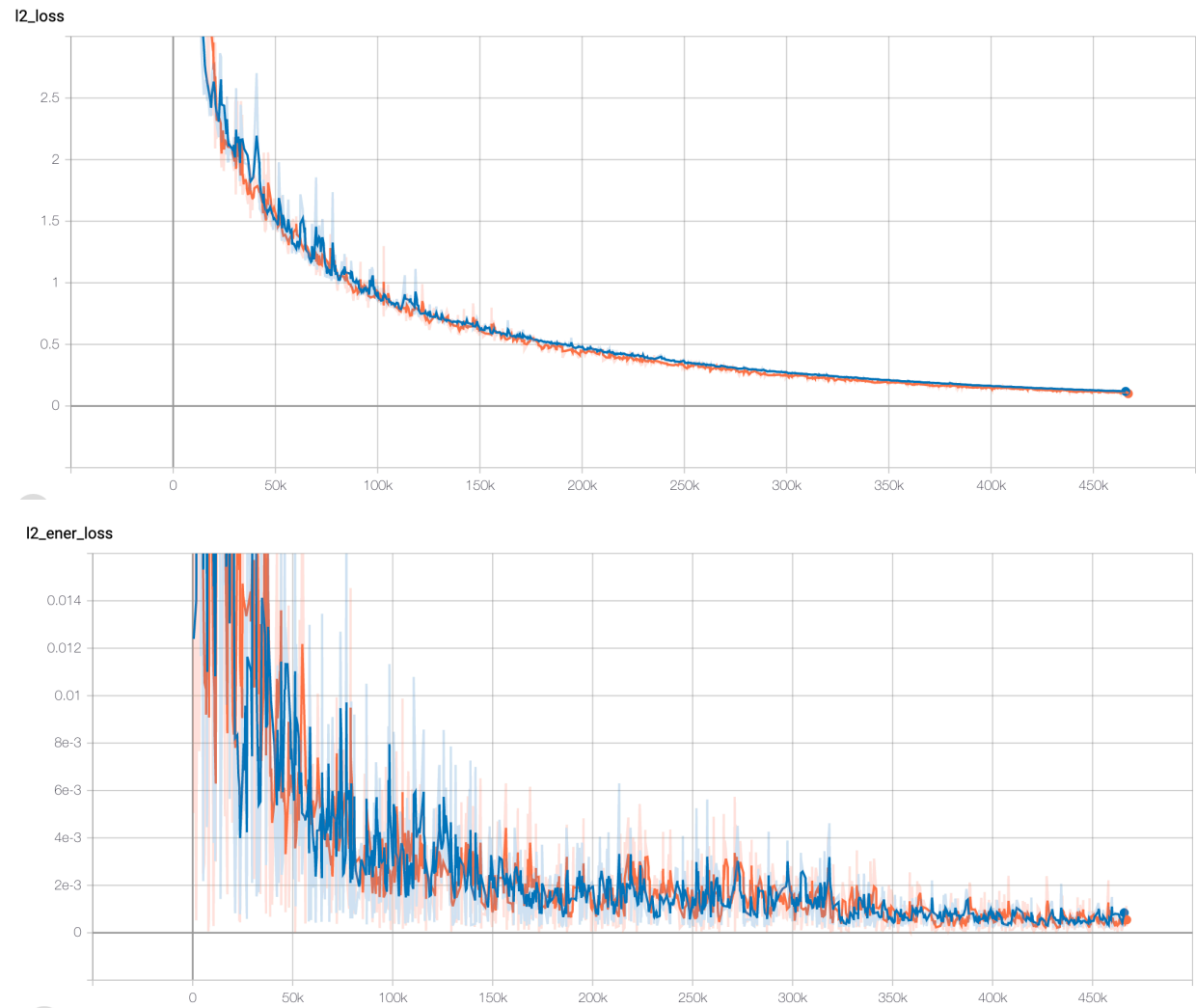
Once you have event files, run TensorBoard and provide the log directory. This should print that TensorBoard has started. Next, connect to `http://tensorboard_server_ip:6006`.

TensorBoard requires a logdir to read logs from. For info on configuring TensorBoard, run `TensorBoard -help`. One can easily change the log name with `"tensorboard_log_dir"` and the sampling frequency with `"tensorboard_freq"`.

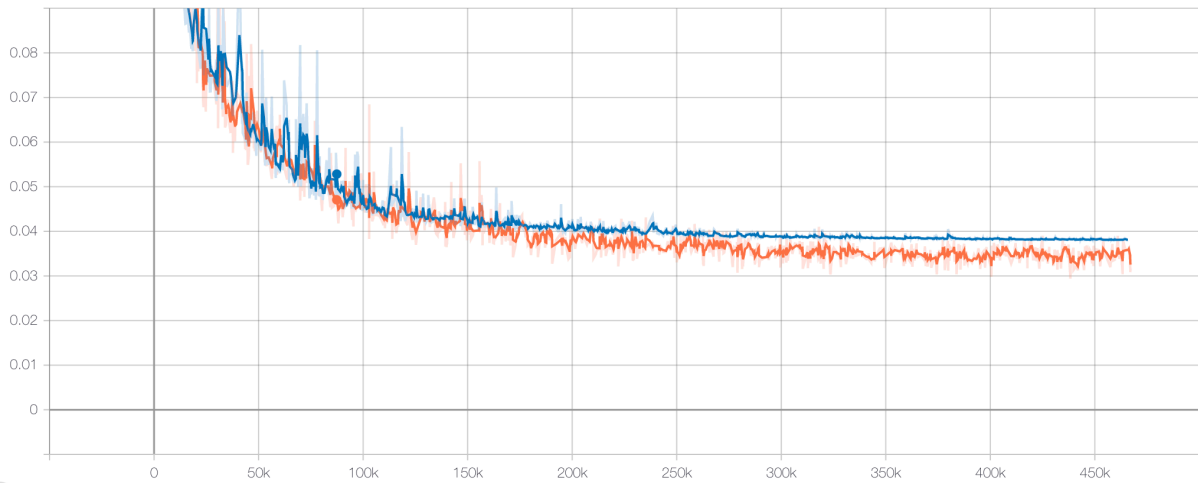
```
tensorboard --logdir path/to/logs
```

### 5.6.3 Examples

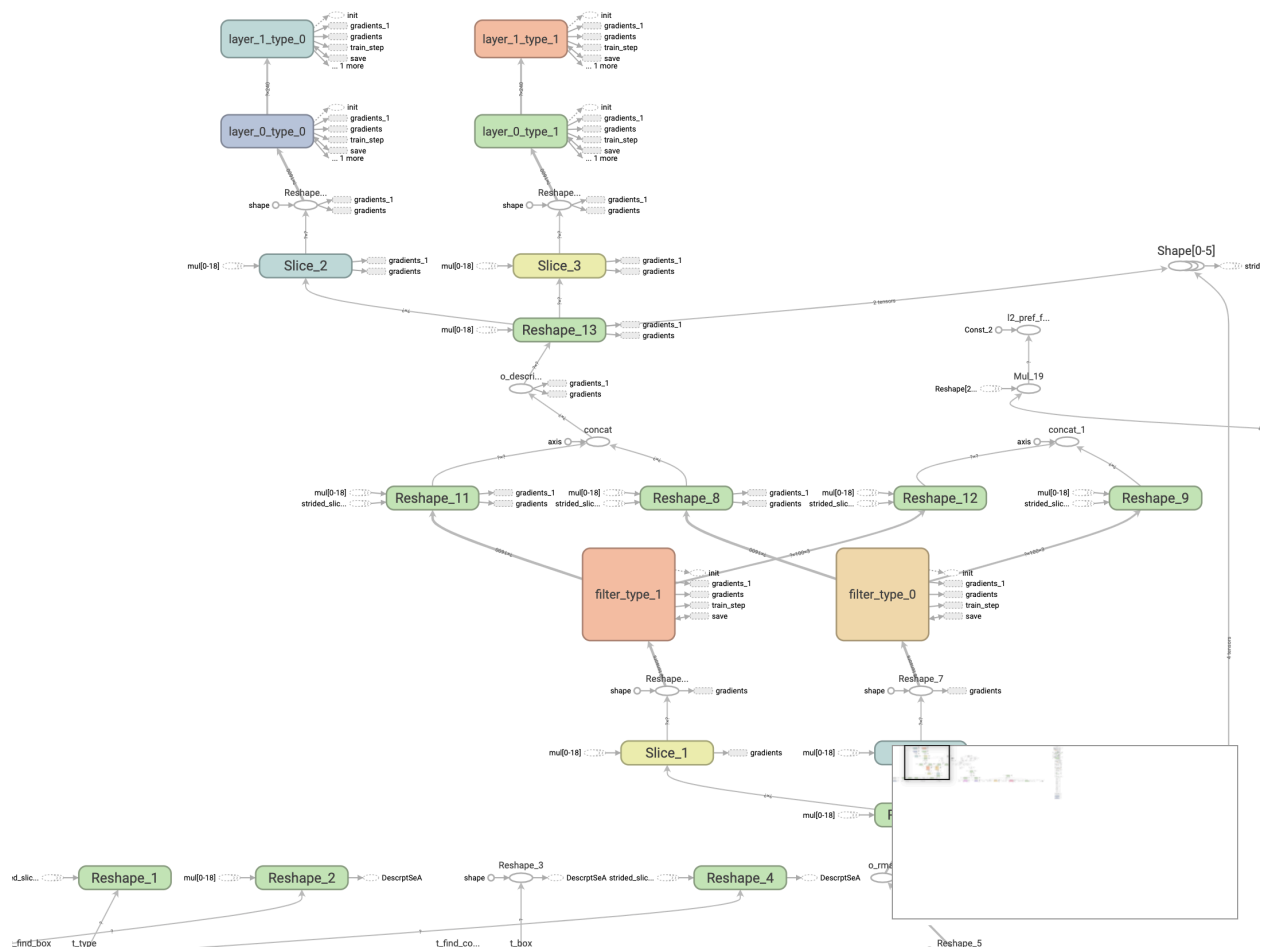
Tracking and visualizing loss metrics(red:train, blue:test)



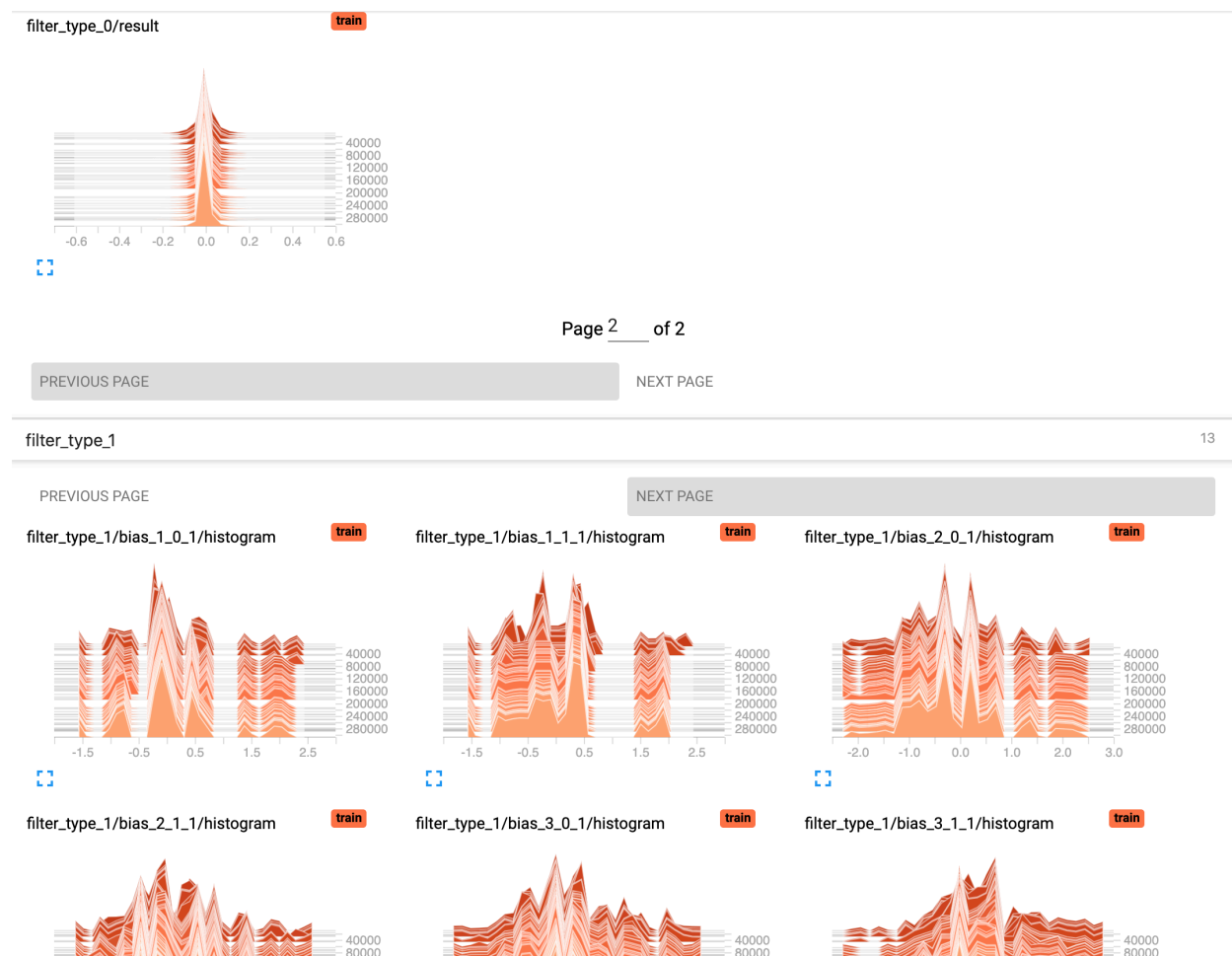
l2\_force\_loss



## Visualizing DeePMD-kit model graph



## Viewing histograms of weights, biases, or other tensors as they change over time

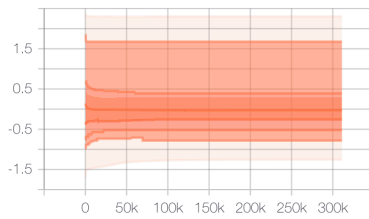


filter\_type\_0

PREVIOUS PAGE

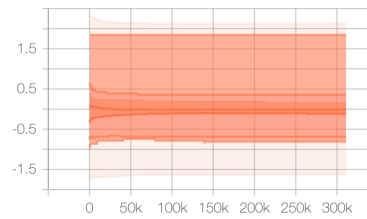
NEXT PAGE

filter\_type\_0/bias\_1\_0\_1/histogram



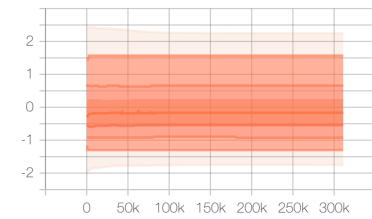
train

filter\_type\_0/bias\_1\_1\_1/histogram



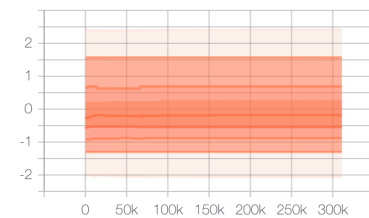
train

filter\_type\_0/bias\_2\_0\_1/histogram



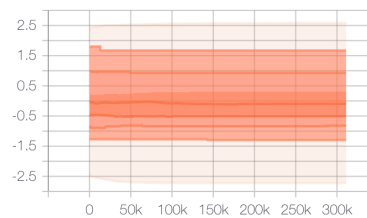
train

filter\_type\_0/bias\_2\_1\_1/histogram



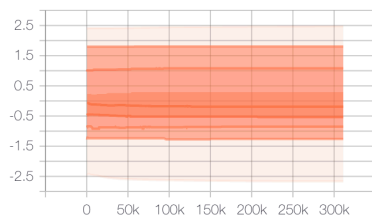
train

filter\_type\_0/bias\_3\_0\_1/histogram



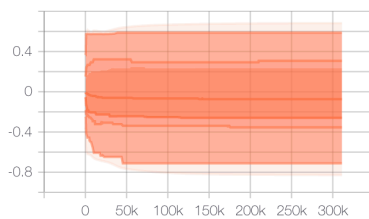
train

filter\_type\_0/bias\_3\_1\_1/histogram



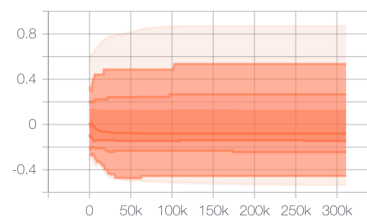
train

filter\_type\_0/matrix\_1\_0\_1/histogram



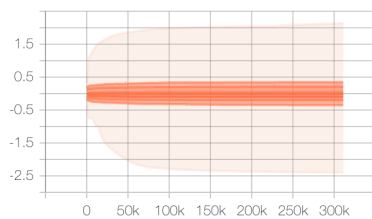
train

filter\_type\_0/matrix\_1\_1\_1/histogram



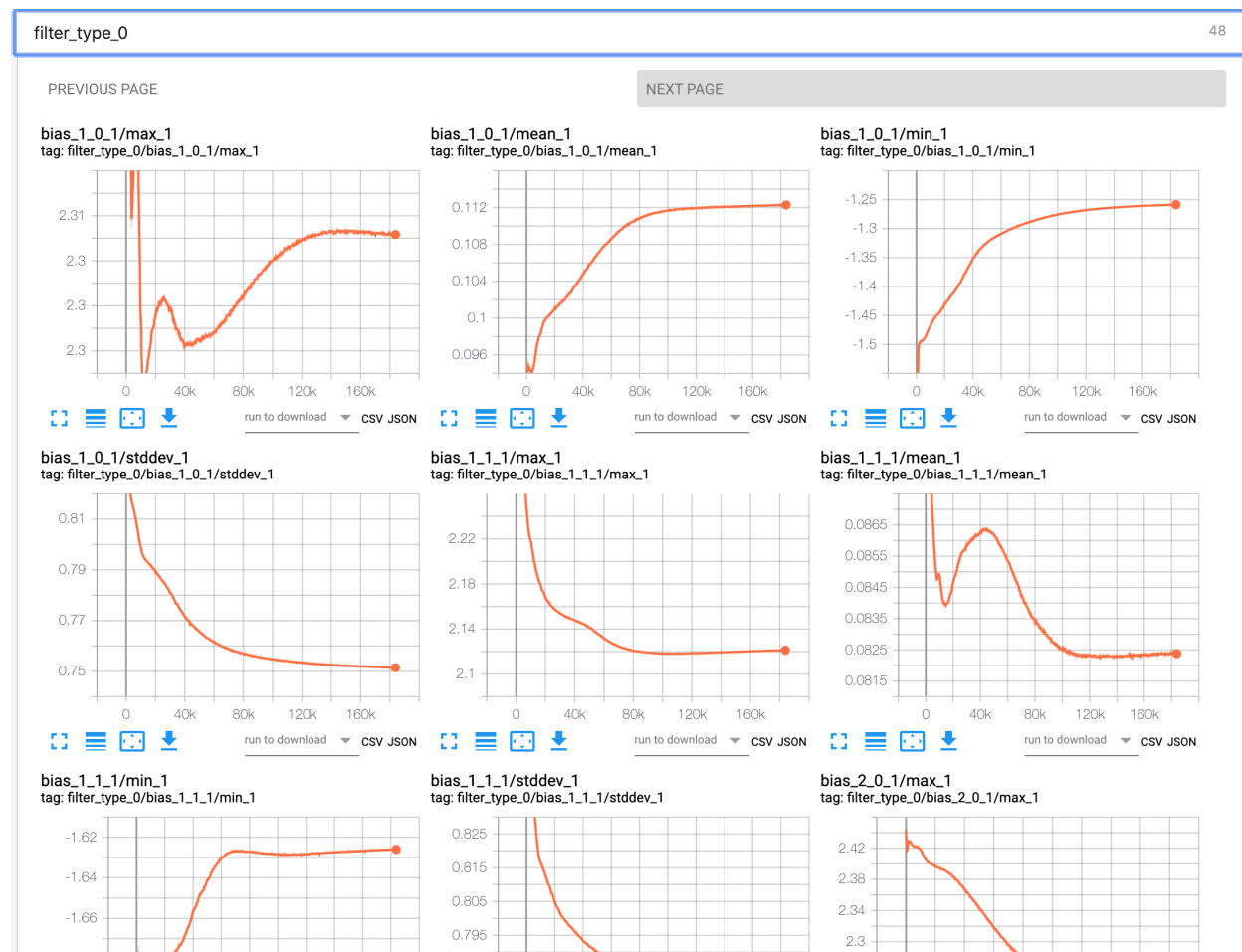
train

filter\_type\_0/matrix\_2\_0\_1/histogram



train

## Viewing summaries of trainable variables



### 5.6.4 Attention

Allowing the tensorboard analysis will takes extra execution time.(eg, 15% increasing @Nvidia GTX 1080Ti double precision with default water sample)

TensorBoard can be used in Google Chrome or Firefox. Other browsers might work, but there may be bugs or performance issues.

## 5.7 Known limitations of using GPUs

If you use DeePMD-kit in a GPU environment, the acceptable value range of some variables is additionally restricted compared to the CPU environment due to the software's GPU implementations:

1. The number of atom types of a given system must be less than 128.
2. The maximum distance between an atom and its neighbors must be less than 128. It can be controlled by setting the rcut value of training parameters.

3. Theoretically, the maximum number of atoms that a single GPU can accept is about 10,000,000. However, this value is limited by the GPU memory size currently, usually within 1000,000 atoms even in the model compression mode.
4. The total sel value of training parameters(in `model/descriptor` section) must be less than 4096.
5. The size of the last layer of the embedding net must be less than 1024 during the model compression process.

## 5.8 Finetune the pretrained model

Pretraining-and-finetuning is a widely used approach in other fields such as Computer Vision (CV) or Natural Language Processing (NLP) to vastly reduce the training cost, while it's not trivial in potential models. Compositions and configurations of data samples or even computational parameters in upstream software (such as VASP) may be different between the pretrained and target datasets, leading to energy shifts or other diversities of training data.

Recently the emerging of methods such as [DPA-1](#) has brought us to a new stage where we can perform similar pretraining-finetuning approaches. DPA-1 can hopefully learn the common knowledge in the pretrained dataset (especially the `force` information) and thus reduce the computational cost in downstream training tasks. If you have a pretrained model `pretrained.pb` (here we support models using `se_atten` descriptor and `ener` fitting net) on a large dataset (for example, [OC2M](#) in [DPA-1 paper](#)), a finetuning strategy can be performed by simply running:

```
$ dp train input.json --finetune pretrained.pb
```

The command above will change the energy bias in the last layer of the fitting net in `pretrained.pb`, according to the training dataset in `input.json`.

Warning: Note that the elements in the training dataset must be contained in the pretrained dataset.

The finetune procedure will inherit the model structures in `pretrained.pb`, and thus it will ignore the model parameters in `input.json`, such as `descriptor`, `fitting_net`, `type_embedding` and `type_map`. However, you can still set the `trainable` parameters in each part of `input.json` to control the training procedure.

To obtain a more simplified script, for example, you can change the `model` part in `input.json` to perform finetuning:

```
{
  "model": {
    "type_map": ["O", "H"],
    "type_embedding": {"trainable": true},
    "descriptor": {},
    "fitting_net": {}
  }
}
```



## FREEZE AND COMPRESS

### 6.1 Freeze a model

The trained neural network is extracted from a checkpoint and dumped into a protobuf(.pb) file. This process is called “freezing” a model. The idea and part of our code are from [Morgan](#). To freeze a model, typically one does

```
$ dp freeze -o graph.pb
```

in the folder where the model is trained. The output model is called `graph.pb`.

In [multi-task mode](#):

- This process will in default output several models, each of which contains the common descriptor and one of the user-defined fitting nets in `fitting_net_dict`, let's name it `fitting_key`, together frozen in `graph_{fitting_key}.pb`. Those frozen models are exactly the same as single-task output with fitting net `fitting_key`.
- If you add `--united-model` option in this situation, the total multi-task model will be frozen into one unit `graph.pb`, which is mainly for multi-task initialization and can not be used directly for inference.

### 6.2 Compress a model

#### 6.2.1 Theory

The compression of the DP model uses three techniques, tabulated inference, operator merging, and precise neighbor indexing, to improve the performance of model training and inference when the model parameters are properly trained.

For better performance, the NN inference can be replaced by tabulated function evaluations if the input of the NN is of dimension one. The idea is to approximate the output of the NN by a piece-wise polynomial fitting. The input domain (a compact domain in  $\mathbb{R}$ ) is divided into  $L_c$  equally spaced intervals, in which we apply a fifth-order polynomial  $g_m^l(x)$  approximation of the  $m$ -th output component of the NN function:

$$g_m^l(x) = a_m^l x^5 + b_m^l x^4 + c_m^l x^3 + d_m^l x^2 + e_m^l x + f_m^l, \quad x \in [x_l, x_{l+1}),$$

where  $l = 1, 2, \dots, L_c$  is the index of the intervals,  $x_1, \dots, x_{L_c}, x_{L_c+1}$  are the endpoints of the intervals, and  $a_m^l, b_m^l, c_m^l, d_m^l, e_m^l$ , and  $f_m^l$  are the fitting parameters. The fitting parameters can be computed by the equations below:

$$a_m^l = \frac{1}{2\Delta x_l^5} [12h_{m,l} - 6(y'_{m,l+1} + y'_{m,l})\Delta x_l + (y''_{m,l+1} - y''_{m,l})\Delta x_l^2],$$

$$\begin{aligned}
b_m^l &= \frac{1}{2\Delta x_l^4}[-30h_{m,l} + (14y'_{m,l+1} + 16y'_{m,l})\Delta x_l + (-2y''_{m,l+1} + 3y''_{m,l})\Delta x_l^2], \\
c_m^l &= \frac{1}{2\Delta x_l^3}[20h_{m,l} - (8y'_{m,l+1} + 12y'_{m,l})\Delta x_l + (y''_{m,l+1} - 3y''_{m,l})\Delta x_l^2], \\
d_m^l &= \frac{1}{2}y''_{m,l}, \\
e_m^l &= y'_{m,l}, \\
f_m^l &= y_{m,l},
\end{aligned}$$

where  $\Delta x_l = x_{l+1} - x_l$  denotes the size of the interval.  $h_{m,l} = y_{m,l+1} - y_{m,l}$ ,  $y_{m,l} = y_m(x_l)$ ,  $y'_{m,l} = y'_m(x_l)$  and  $y''_{m,l} = y''_m(x_l)$  are the value, the first-order derivative, and the second-order derivative of the  $m$ -th component of the target NN function at the interval point  $x_l$ , respectively. The first and second-order derivatives are easily calculated by the back-propagation of the NN functions.

In the standard DP model inference, taking the [two-body embedding descriptor](#) as an example, the matrix product  $(\mathcal{G}^i)^T \mathcal{R}$  requires the transfer of the tensor  $\mathcal{G}^i$  between the register and the host/device memories, which usually becomes the bottle-neck of the computation due to the relatively small memory bandwidth of the GPUs. The compressed DP model merges the matrix multiplication  $(\mathcal{G}^i)^T \mathcal{R}$  with the tabulated inference step. More specifically, once one column of the  $(\mathcal{G}^i)^T$  is evaluated, it is immediately multiplied with one row of the environment matrix in the register, and the outer product is deposited to the result of  $(\mathcal{G}^i)^T \mathcal{R}$ . By the operator merging technique, the allocation of  $\mathcal{G}^i$  and the memory movement between register and host/device memories is avoided. The operator merging of the three-body embedding can be derived analogously.

The first dimension,  $N_c$ , of the environment  $(\mathcal{R}^i)$  and embedding  $(\mathcal{G}^i)$  matrices is the expected maximum number of neighbors. If the number of neighbors of an atom is smaller than  $N_c$ , the corresponding positions of the matrices are pad with zeros. In practice, if the real number of neighbors is significantly smaller than  $N_c$ , a notable operation is spent on the multiplication of padding zeros. In the compressed DP model, the number of neighbors is precisely indexed at the tabulated inference stage, further saving computational costs.<sup>1</sup>

## 6.2.2 Instructions

Once the frozen model is obtained from DeePMD-kit, we can get the neural network structure and its parameters (weights, biases, etc.) from the trained model, and compress it in the following way:

```
dp compress -i graph.pb -o graph-compress.pb
```

where `-i` gives the original frozen model, `-o` gives the compressed model. Several other command line options can be passed to `dp compress`, which can be checked with

```
$ dp compress --help
```

An explanation will be provided

```
usage: dp compress [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                  [-m {master,collect,workers}] [-i INPUT] [-o OUTPUT]
                  [-s STEP] [-e EXTRAPOLATE] [-f FREQUENCY]
                  [-c CHECKPOINT_FOLDER]
```

(continues on next page)

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a [Creative Commons Attribution \(CC BY\)](#) license.

(continued from previous page)

optional arguments:

```

-h, --help          show this help message and exit
-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}, --log-level {DEBUG,3,INFO,2,WARNING,1,ERROR,0}
                    set verbosity level by string or number, 0=ERROR,
                    1=WARNING, 2=INFO and 3=DEBUG (default: INFO)
-l LOG_PATH, --log-path LOG_PATH
                    set log file to log messages to disk, if not
                    specified, the logs will only be output to console
                    (default: None)
-m {master,collect,workers}, --mpi-log {master,collect,workers}
                    Set the manner of logging when running with MPI.
                    'master' logs only on main process, 'collect'
                    broadcasts logs from workers to master and 'workers'
                    means each process will output its own log (default:
                    master)
-i INPUT, --input INPUT
                    The original frozen model, which will be compressed by
                    the code (default: frozen_model.pb)
-o OUTPUT, --output OUTPUT
                    The compressed model (default:
                    frozen_model_compressed.pb)
-s STEP, --step STEP Model compression uses fifth-order polynomials to
                    interpolate the embedding-net. It introduces two
                    tables with different step size to store the
                    parameters of the polynomials. The first table covers
                    the range of the training data, while the second table
                    is an extrapolation of the training data. The domain
                    of each table is uniformly divided by a given step
                    size. And the step(parameter) denotes the step size of
                    the first table and the second table will use 10 *
                    step as it's step size to save the memory. Usually the
                    value ranges from 0.1 to 0.001. Smaller step means
                    higher accuracy and bigger model size (default: 0.01)
-e EXTRAPOLATE, --extrapolate EXTRAPOLATE
                    The domain range of the first table is automatically
                    detected by the code: [d_low, d_up]. While the second
                    table ranges from the first table's upper
                    boundary(d_up) to the extrapolate(parameter) * d_up:
                    [d_up, extrapolate * d_up] (default: 5)
-f FREQUENCY, --frequency FREQUENCY
                    The frequency of tabulation overflow check(Whether the
                    input environment matrix overflow the first or second
                    table range). By default do not check the overflow
                    (default: -1)
-c CHECKPOINT_FOLDER, --checkpoint-folder CHECKPOINT_FOLDER
                    path to checkpoint folder (default: .)
-t TRAINING_SCRIPT, --training-script TRAINING_SCRIPT
                    The training script of the input frozen model
                    (default: None)

```

### Parameter explanation

Model compression, which includes tabulating the embedding net. The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. For model descriptor with `se_e2_a` type, the first sub-table takes the `stride(parameter)` as its uniform stride, while the second sub-table takes `10 * stride` as its uniform stride; For model descriptor with `se_e3` type, the first sub-table takes `10 * stride` as its uniform stride, while the second sub-table takes `100 * stride` as its uniform stride. The range of the first table is automati-

cally detected by DeePMD-kit, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) \* upper. Finally, we added a check frequency parameter. It indicates how often the program checks for overflow(if the input environment matrix overflows the first or second table range) during the MD inference.

#### Justification of model compression

Model compression, with little loss of accuracy, can greatly speed up MD inference time. According to different simulation systems and training parameters, the speedup can reach more than 10 times at both CPU and GPU devices. At the same time, model compression can greatly change memory usage, reducing as much as 20 times under the same hardware conditions.

#### Acceptable original model version

The model compression interface requires the version of DeePMD-kit used in the original model generation should be 2.0.0-alpha.0 or above. If one has a frozen 1.2 or 1.3 model, one can upgrade it through the `dp convert-from` interface. (eg: `dp convert-from 1.2/1.3 -i old_frozen_model.pb -o new_frozen_model.pb`)

#### Acceptable descriptor type

Descriptors with `se_e2_a`, `se_e3`, `se_e2_r` and `se_atten_v2` types are supported by the model compression feature. `Hybrid` mixed with the above descriptors is also supported.

Available activation functions for descriptor:

- tanh
- gelu
- relu
- relu6
- softplus
- sigmoid

## TEST

### 7.1 Test a model

The frozen model can be used in many ways. The most straightforward test can be performed using `dp test`. A typical usage of `dp test` is

```
dp test -m graph.pb -s /path/to/system -n 30
```

where `-m` gives the tested model, `-s` the path to the tested system and `-n` the number of tested frames. Several other command line options can be passed to `dp test`, which can be checked with

```
$ dp test --help
```

An explanation will be provided

```
usage: dp test [-h] [-m MODEL] [-s SYSTEM] [-S SET_PREFIX] [-n NUMB_TEST]
              [-r RAND_SEED] [--shuffle-test] [-d DETAIL_FILE]

optional arguments:
  -h, --help            show this help message and exit
  -m MODEL, --model MODEL
                        Frozen model file to import
  -s SYSTEM, --system SYSTEM
                        The system dir
  -S SET_PREFIX, --set-prefix SET_PREFIX
                        The set prefix
  -n NUMB_TEST, --numb-test NUMB_TEST
                        The number of data for test
  -r RAND_SEED, --rand-seed RAND_SEED
                        The random seed
  --shuffle-test         Shuffle test data
  -d DETAIL_FILE, --detail-file DETAIL_FILE
                        The prefix to files where details of energy, force and virial accuracy/
                        accuracy per atom will be written
  -a, --atomic           Test the accuracy of atomic label, i.e. energy / tensor (dipole, polar)
```

## 7.2 Calculate Model Deviation

### 7.2.1 Theory

Model deviation  $\epsilon_y$  is the standard deviation of properties  $y$  inferred by an ensemble of models  $\mathcal{M}_1, \dots, \mathcal{M}_{n_m}$  that are trained by the same dataset(s) with the model parameters initialized independently. The DeePMD-kit supports  $y$  to be the atomic force  $\mathbf{F}_i$  and the virial tensor  $\Xi$ . The model deviation is used to estimate the error of a model at a certain data frame, denoted by  $\mathbf{x}$ , containing the coordinates and chemical species of all atoms. We present the model deviation of the atomic force and the virial tensor

$$\epsilon_{\mathbf{F},i}(\mathbf{x}) = \sqrt{\langle \|\mathbf{F}_i(\mathbf{x}; \boldsymbol{\theta}_k) - \langle \mathbf{F}_i(\mathbf{x}; \boldsymbol{\theta}_k) \rangle\|^2 \rangle},$$

$$\epsilon_{\Xi,\alpha\beta}(\mathbf{x}) = \frac{1}{N} \sqrt{\langle (\Xi_{\alpha\beta}(\mathbf{x}; \boldsymbol{\theta}_k) - \langle \Xi_{\alpha\beta}(\mathbf{x}; \boldsymbol{\theta}_k) \rangle)^2 \rangle},$$

where  $\boldsymbol{\theta}_k$  is the parameters of the model  $\mathcal{M}_k$ , and the ensemble average  $\langle \cdot \rangle$  is estimated by

$$\langle y(\mathbf{x}; \boldsymbol{\theta}_k) \rangle = \frac{1}{n_m} \sum_{k=1}^{n_m} y(\mathbf{x}; \boldsymbol{\theta}_k).$$

Small  $\epsilon_{\mathbf{F},i}$  means the model has learned the given data; otherwise, it is not covered, and the training data needs to be expanded. If the magnitude of  $\mathbf{F}_i$  or  $\Xi$  is quite large, a relative model deviation  $\epsilon_{\mathbf{F},i,\text{rel}}$  or  $\epsilon_{\Xi,\alpha\beta,\text{rel}}$  can be used instead of the absolute model deviation:

$$\epsilon_{\mathbf{F},i,\text{rel}}(\mathbf{x}) = \frac{|\epsilon_{\mathbf{F},i}(\mathbf{x})|}{|\langle \mathbf{F}_i(\mathbf{x}; \boldsymbol{\theta}_k) \rangle| + \nu},$$

$$\epsilon_{\Xi,\alpha\beta,\text{rel}}(\mathbf{x}) = \frac{\epsilon_{\Xi,\alpha\beta}(\mathbf{x})}{|\langle \Xi(\mathbf{x}; \boldsymbol{\theta}_k) \rangle| + \nu},$$

where  $\nu$  is a small constant used to protect an atom where the magnitude of  $\mathbf{F}_i$  or  $\Xi$  is small from having a large model deviation.

Statistics of  $\epsilon_{\mathbf{F},i}$  and  $\epsilon_{\Xi,\alpha\beta}$  can be provided, including the maximum, average, and minimal model deviation over the atom index  $i$  and over the component index  $\alpha, \beta$ , respectively. The maximum model deviation of forces  $\epsilon_{\mathbf{F},\text{max}}$  in a frame was found to be the best error indicator in a concurrent or active learning algorithm.<sup>1</sup>

### 7.2.2 Instructions

One can also use a subcommand to calculate the deviation of predicted forces or virials for a bunch of models in the following way:

```
dp model-devi -m graph.000.pb graph.001.pb graph.002.pb graph.003.pb -s ./data -o model_devi.out
```

where `-m` specifies graph files to be calculated, `-s` gives the data to be evaluated, `-o` the file to which model deviation results is dumped. Here is more information on this sub-command:

<sup>1</sup> This section is built upon Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E. A. Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M. York, Weinan E, Roberto Car, Linfeng Zhang, Han Wang, J. Chem. Phys. 159, 054801 (2023) licensed under a Creative Commons Attribution (CC BY) license.

```
usage: dp model-devi [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}]
                    [-l LOG_PATH] [-m MODELS [MODELS ...]] [-s SYSTEM]
                    [-S SET_PREFIX] [-o OUTPUT] [-f FREQUENCY] [-i ITEMS]

optional arguments:
  -h, --help            show this help message and exit
  -v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}, --log-level {DEBUG,3,INFO,2,WARNING,1,ERROR,0}
                        set verbosity level by string or number, 0=ERROR,
                        1=WARNING, 2=INFO and 3=DEBUG (default: INFO)
  -l LOG_PATH, --log-path LOG_PATH
                        set log file to log messages to disk, if not
                        specified, the logs will only be output to console
                        (default: None)
  -m MODELS [MODELS ...], --models MODELS [MODELS ...]
                        Frozen models file to import (default:
                        ['graph.000.pb', 'graph.001.pb', 'graph.002.pb',
                        'graph.003.pb'])
  -s SYSTEM, --system SYSTEM
                        The system directory, not support recursive detection.
                        (default: .)
  -S SET_PREFIX, --set-prefix SET_PREFIX
                        The set prefix (default: set)
  -o OUTPUT, --output OUTPUT
                        The output file for results of model deviation
                        (default: model_devi.out)
  -f FREQUENCY, --frequency FREQUENCY
                        The trajectory frequency of the system (default: 1)
```

For more details concerning the definition of model deviation and its application, please refer to Yuzhi Zhang, Haidi Wang, Weijie Chen, Jinzhe Zeng, Linfeng Zhang, Han Wang, and Weinan E, DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models, Computer Physics Communications, 2020, 253, 107206.

### 7.2.3 Relative model deviation

By default, the model deviation is output in absolute value. If the argument `--relative` is passed, then the relative model deviation of the force will be output, including values output by the argument `--atomic`. The relative model deviation of the force on atom  $i$  is defined by

$$E_{f_i} = \frac{|D_{f_i}|}{|f_i| + l}$$

where  $D_{f_i}$  is the absolute model deviation of the force on atom  $i$ ,  $f_i$  is the norm of the force and  $l$  is provided as the parameter of the keyword `relative`. If the argument `--relative_v` is set, then the relative model deviation of the virial will be output instead of the absolute value, with the same definition of that of the force:

$$E_{v_i} = \frac{|D_{v_i}|}{|v_i| + l}$$





## INFERENCE

Note that the model for inference is required to be compatible with the DeePMD-kit package. See [Model compatibility](#) for details.

### 8.1 Python interface

One may use the python interface of DeePMD-kit for model inference, an example is given as follows

```
from deepmd.infer import DeepPot
import numpy as np

dp = DeepPot("graph.pb")
coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
cell = np.diag(10 * np.ones(3)).reshape([1, -1])
atype = [1, 0, 1]
e, f, v = dp.eval(coord, cell, atype)
```

where *e*, *f* and *v* are predicted energy, force and virial of the system, respectively.

Furthermore, one can use the python interface to calculate model deviation.

```
from deepmd.infer import calc_model_devi
from deepmd.infer import DeepPot as DP
import numpy as np

coord = np.array([[1, 0, 0], [0, 0, 1.5], [1, 0, 3]]).reshape([1, -1])
cell = np.diag(10 * np.ones(3)).reshape([1, -1])
atype = [1, 0, 1]
graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
model_devi = calc_model_devi(coord, cell, atype, graphs)
```

Note that if the model inference or model deviation is performed cyclically, one should avoid calling the same model multiple times. Otherwise, tensorflow will never release the memory and this may lead to an out-of-memory (OOM) error.

### 8.1.1 External neighbor list algorithm

The native neighbor list algorithm of the DeePMD-kit is in  $O(N^2)$  complexity ( $N$  is the number of atoms). While this is not a problem for small systems that quantum methods can afford, the large systems for molecular dynamics have slow performance. In this case, one may pass an external neighbor list that has lower complexity to *DeepPot*, once it is compatible with `ase.neighborlist.NewPrimitiveNeighborList`.

```
import ase.neighborlist

neighbor_list = ase.neighborlist.NewPrimitiveNeighborList(
    cutoffs=6, bothways=True, self_interaction=False
)
dp = DeepPot("graph.pb", neighbor_list=neighbor_list)
```

The `update` and `build` methods will be called by *DeepPot*, and `first_neigh`, `pair_second`, and `offset_vec` properties will be used.

## 8.2 C/C++ interface

### 8.2.1 C++ interface

The C++ interface of DeePMD-kit is also available for the model interface, which is considered faster than the Python interface. An example `infer_water.cpp` is given below:

```
#include "deepmd/DeepPot.h"

int main(){
    deepmd::DeepPot dp ("graph.pb");
    std::vector<double> coord = {1., 0., 0., 0., 0., 1.5, 1., 0., 3.};
    std::vector<double> cell = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
    std::vector<int> atype = {1, 0, 1};
    double e;
    std::vector<double> f, v;
    dp.compute (e, f, v, coord, atype, cell);
}
```

where `e`, `f` and `v` are predicted energy, force and virial of the system, respectively. See `deepmd::DeepPot` for details.

You can compile `infer_water.cpp` using `gcc`:

```
gcc infer_water.cpp -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--no-
↳ as-needed -ldeepmd_cc -lstdc++ -ltensorflow_cc -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=
↳ $tensorflow_root/lib -o infer_water
```

and then run the program:

```
./infer_water
```

## 8.2.2 C interface

Although C is harder to write, the C library will not be affected by different versions of C++ compilers.

An example `infer_water.c` is given below:

```
#include <stdio.h>
#include <stdlib.h>
#include "deepmd/c_api.h"

int main(){
    const char* model = "graph.pb";
    double coord[] = {1., 0., 0., 0., 0., 1.5, 1., 0., 3.};
    double cell[] = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
    int atype[] = {1, 0, 1};
    // init C pointers with given memory
    double* e = malloc(sizeof(*e));
    double* f = malloc(sizeof(*f) * 9); // natoms * 3
    double* v = malloc(sizeof(*v) * 9);
    double* ae = malloc(sizeof(*ae) * 9); // natoms
    double* av = malloc(sizeof(*av) * 27); // natoms * 9
    // DP model
    DP_DeepPot* dp = DP_NewDeepPot(model);
    DP_DeepPotCompute(dp, 3, coord, atype, cell, e, f, v, ae, av);
    // print results
    printf("energy: %f\n", *e);
    for (int ii = 0; ii < 9; ++ii)
        printf("force[%d]: %f\n", ii, f[ii]);
    for (int ii = 0; ii < 9; ++ii)
        printf("force[%d]: %f\n", ii, v[ii]);
    // free memory
    free(e);
    free(f);
    free(v);
    free(ae);
    free(av);
    DP_DeleteDeepPot(dp);
}
```

where `e`, `f` and `v` are predicted energy, force and virial of the system, respectively. `ae` and `av` are atomic energy and atomic virials, respectively. See `DP_DeepPotCompute()` for details.

You can compile `infer_water.c` using `gcc`:

```
gcc infer_water.c -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--no-as-needed -ldeepmd_c -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=$tensorflow_root/lib -o infer_water
```

and then run the program:

```
./infer_water
```

## 8.2.3 Header-only C++ library interface (recommended)

The header-only C++ library is built based on the C library. Thus, it has the same ABI compatibility as the C library but provides a powerful C++ interface. To use it, include `deepmd/deepmd.hpp`.

```
#include "deepmd/deepmd.hpp"

int main(){
    deepmd::hpp::DeepPot dp ("graph.pb");
    std::vector<double> coord = {1., 0., 0., 0., 0., 1.5, 1., 0., 3.};
    std::vector<double> cell = {10., 0., 0., 0., 10., 0., 0., 0., 10.};
    std::vector<int> atype = {1, 0, 1};
    double e;
    std::vector<double> f, v;
    dp.compute (e, f, v, coord, atype, cell);
}
```

Note that the feature of the header-only C++ library is still limited compared to the original C++ library. See `deepmd::hpp::DeepPot` for details.

You can compile `infer_water_hpp.cpp` using `gcc`:

```
gcc infer_water_hpp.cpp -L $deepmd_root/lib -L $tensorflow_root/lib -I $deepmd_root/include -Wl,--
no-as-needed -ldeepmd_c -Wl,-rpath=$deepmd_root/lib -Wl,-rpath=$tensorflow_root/lib -o infer_
water_hpp
```

and then run the program:

```
./infer_water_hpp
```

In some cases, one may want to pass the custom neighbor list instead of the native neighbor list. The above code can be revised as follows:

```
// neighbor list
std::vector<std::vector<int>> nlist_vec = {
    {1, 2},
    {0, 2},
    {0, 1}
};
std::vector<int> ilist(3), numneigh(3);
std::vector<int*> firstneigh(3);
InputNlist nlist(3, &ilist[0], &numneigh[0], &firstneigh[0]);
convert_nlist(nlist, nlist_vec);
dp.compute (e, f, v, coord, atype, cell, 0, nlist, 0);
```

Here, `nlist_vec` means the neighbors of atom 0 are atom 1 and atom 2, the neighbors of atom 1 are atom 0 and atom 2, and the neighbors of atom 2 are atom 0 and atom 1.

## 8.3 Node.js interface

If Node.js interface is installed, one can use the Node.js interface for model inference, which is a wrapper of the header-only C++ API.

A simple example is shown below.

```
const deepmd = require("deepmd-kit");

const dp = new deepmd.DeepPot("graph.pb");

const coord = [1., 0., 0., 0., 0., 1.5, 1., 0., 3.];
const atype = [1, 0, 1];
const cell = [10., 0., 0., 0., 10., 0., 0., 0., 10.];

const v_coord = new deepmd.vectord(coord.length);
const v_atype = new deepmd.vectori(atype.length);
const v_cell = new deepmd.vectord(cell.length);
for (var i = 0; i < coord.length; i++) v_coord.set(i, coord[i]);
for (var i = 0; i < atype.length; i++) v_atype.set(i, atype[i]);
for (var i = 0; i < cell.length; i++) v_cell.set(i, cell[i]);

var energy = 0.0
var v_forces = new deepmd.vectord();
var v_virials = new deepmd.vectord();

energy = dp.compute(energy, v_forces, v_virials, v_coord, v_atype, v_cell);

console.log("energy:", energy);
console.log("forces:", [...Array(v_forces.size()).keys()].map(i => v_forces.get(i)));
console.log("virials:", [...Array(v_virials.size()).keys()].map(i => v_virials.get(i)));
```

Energy, forces, and virials will be printed to the screen.



## COMMAND LINE INTERFACE

DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics

```
usage: dp [-h] [--version]
          {transfer,train,freeze,test,compress,doc-train-input,model-devi,convert-from,neighbor-
↪stat,train-nvnmd,gui}
          ...
```

### 9.1 Named Arguments

<code>--version</code>	show program's version number and exit
------------------------	--

### 9.2 Valid subcommands

<code>command</code>	Possible choices: transfer, train, freeze, test, compress, doc-train-input, model-devi, convert-from, neighbor-stat, train-nvnmd, gui
----------------------	---

### 9.3 Sub-commands

#### 9.3.1 transfer

pass parameters to another model

```
dp transfer [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
            [-r RAW_MODEL] [-O OLD_MODEL] [-o OUTPUT]
```

### Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-r, --raw-model	the model receiving parameters Default: "raw_frozen_model.pb"
-O, --old-model	the model providing parameters Default: "old_frozen_model.pb"
-o, --output	the model after passing parameters Default: "frozen_model.pb"

### 9.3.2 train

train a model

```
dp train [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
        [-m {master,collect,workers}]
        [-i INIT_MODEL | -r RESTART | -f INIT_FRZ_MODEL | -t FINETUNE]
        [-o OUTPUT] [--skip-neighbor-stat]
        INPUT
```

### Positional Arguments

INPUT	the input parameter file in json or yaml format
-------	---

### Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-m, --mpi-log	Possible choices: master, collect, workers Set the manner of logging when running with MPI. 'master' logs only on main process, 'collect' broadcasts logs from workers to master and 'workers' means each process will output its own log Default: "master"
-i, --init-model	Initialize the model by the provided path prefix of checkpoint files.



- r, --restart Restart the training from the provided path prefix of checkpoint files.
- f, --init-frz-model Initialize the training from the frozen model.
- t, --finetune Finetune the frozen pretrained model.
- o, --output The output file of the parameters used in training.  
Default: "out.json"
- skip-neighbor-stat Skip calculating neighbor statistics. Sel checking, automatic sel, and model compression will be disabled.  
Default: False

examples:

```
dp train input.json dp train input.json -restart model.ckpt dp train input.json -init-model model.ckpt
```

### 9.3.3 freeze

freeze the model

```
dp freeze [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
          [-c CHECKPOINT_FOLDER] [-o OUTPUT] [-n NODE_NAMES] [-w NVNMD_WEIGHT]
          [--united-model]
```

#### Named Arguments

- v, --log-level Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0  
set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG  
Default: "INFO"
- l, --log-path set log file to log messages to disk, if not specified, the logs will only be output to console
- c, --checkpoint-folder path to checkpoint folder  
Default: "."
- o, --output name of graph, will output to the checkpoint folder  
Default: "frozen\_model.pb"
- n, --node-names the frozen nodes, if not set, determined from the model type
- w, --nvnmd-weight the name of weight file (.npy), if set, save the model's weight into the file
- united-model When in multi-task mode, freeze all nodes into one united model  
Default: False

examples:

```
dp freeze dp freeze -o graph.pb
```

### 9.3.4 test

test the model

```
dp test [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH] [-m MODEL]
        [-s SYSTEM | -f DATAFILE] [-S SET_PREFIX] [-n NUMB_TEST]
        [-r RAND_SEED] [--shuffle-test] [-d DETAIL_FILE] [-a]
```

#### Named Arguments

<code>-v, --log-level</code>	<p>Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0</p> <p>set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG</p> <p>Default: "INFO"</p>
<code>-l, --log-path</code>	<p>set log file to log messages to disk, if not specified, the logs will only be output to console</p>
<code>-m, --model</code>	<p>Frozen model file to import</p> <p>Default: "frozen_model.pb"</p>
<code>-s, --system</code>	<p>The system dir. Recursively detect systems in this directory</p> <p>Default: "."</p>
<code>-f, --datafile</code>	<p>The path to file of test list.</p>
<code>-S, --set-prefix</code>	<p>The set prefix</p> <p>Default: "set"</p>
<code>-n, --numb-test</code>	<p>The number of data for test. 0 means all data.</p> <p>Default: 0</p>
<code>-r, --rand-seed</code>	<p>The random seed</p>
<code>--shuffle-test</code>	<p>Shuffle test data</p> <p>Default: False</p>
<code>-d, --detail-file</code>	<p>The prefix to files where details of energy, force and virial accuracy/accuracy per atom will be written</p>
<code>-a, --atomic</code>	<p>Test the accuracy of atomic label, i.e. energy / tensor (dipole, polar)</p> <p>Default: False</p>

examples:

```
dp test -m graph.pb -s /path/to/system -n 30
```

### 9.3.5 compress

compress a model

```
dp compress [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
            [-m {master,collect,workers}] [-i INPUT] [-o OUTPUT] [-s STEP]
            [-e EXTRAPOLATE] [-f FREQUENCY] [-c CHECKPOINT_FOLDER]
            [-t TRAINING_SCRIPT]
```

#### Named Arguments

-v, --log-level	<p>Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0</p> <p>set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG</p> <p>Default: "INFO"</p>
-l, --log-path	<p>set log file to log messages to disk, if not specified, the logs will only be output to console</p>
-m, --mpi-log	<p>Possible choices: master, collect, workers</p> <p>Set the manner of logging when running with MPI. 'master' logs only on main process, 'collect' broadcasts logs from workers to master and 'workers' means each process will output its own log</p> <p>Default: "master"</p>
-i, --input	<p>The original frozen model, which will be compressed by the code</p> <p>Default: "frozen_model.pb"</p>
-o, --output	<p>The compressed model</p> <p>Default: "frozen_model_compressed.pb"</p>
-s, --step	<p>Model compression uses fifth-order polynomials to interpolate the embedding-net. It introduces two tables with different step size to store the parameters of the polynomials. The first table covers the range of the training data, while the second table is an extrapolation of the training data. The domain of each table is uniformly divided by a given step size. And the step(parameter) denotes the step size of the first table and the second table will use 10 * step as it's step size to save the memory. Usually the value ranges from 0.1 to 0.001. Smaller step means higher accuracy and bigger model size</p> <p>Default: 0.01</p>
-e, --extrapolate	<p>The domain range of the first table is automatically detected by the code: [d_low, d_up]. While the second table ranges from the first table's upper boundary(d_up) to the extrapolate(parameter) * d_up: [d_up, extrapolate * d_up]</p> <p>Default: 5</p>
-f, --frequency	<p>The frequency of tabulation overflow check(Whether the input environment matrix overflow the first or second table range). By default do not check the overflow</p> <p>Default: -1</p>

- c, --checkpoint-folder path to checkpoint folder  
Default: “model-compression”
- t, --training-script The training script of the input frozen model

examples:

dp compress dp compress -i graph.pb -o compressed.pb

### 9.3.6 doc-train-input

print the documentation (in rst format) of input training parameters.

```
dp doc-train-input [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                  [--out-type {rst,json}]
```

#### Named Arguments

- v, --log-level Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0  
set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG  
Default: “INFO”
- l, --log-path set log file to log messages to disk, if not specified, the logs will only be output to console
- out-type Possible choices: rst, json  
The output type  
Default: “rst”

### 9.3.7 model-devi

calculate model deviation

```
dp model-devi [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
               [-m MODELS [MODELS ...]] [-s SYSTEM] [-S SET_PREFIX] [-o OUTPUT]
               [-f FREQUENCY] [--real_error] [--atomic] [--relative RELATIVE]
               [--relative_v RELATIVE_V]
```

#### Named Arguments

- v, --log-level Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0  
set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG  
Default: “INFO”
- l, --log-path set log file to log messages to disk, if not specified, the logs will only be output to console
- m, --models Frozen models file to import  
Default: [‘graph.000.pb’, ‘graph.001.pb’, ‘graph.002.pb’, ‘graph.003.pb’]

<code>-s, --system</code>	The system directory. Recursively detect systems in this directory. Default: “.”
<code>-S, --set-prefix</code>	The set prefix Default: “set”
<code>-o, --output</code>	The output file for results of model deviation Default: “model_devi.out”
<code>-f, --frequency</code>	The trajectory frequency of the system Default: 1
<code>--real_error</code>	Calculate the RMS real error of the model. The real data should be given in the systems. Default: False
<code>--atomic</code>	Print the force model deviation of each atom. Default: False
<code>--relative</code>	Calculate the relative model deviation of force. The level parameter for computing the relative model deviation of the force should be given.
<code>--relative_v</code>	Calculate the relative model deviation of virial. The level parameter for computing the relative model deviation of the virial should be given.

examples:

```
dp model-devi -m graph.000.pb graph.001.pb graph.002.pb graph.003.pb -s ./data -o model_devi.out
```

### 9.3.8 convert-from

convert lower model version to supported version

```
dp convert-from [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                [-i INPUT_MODEL] [-o OUTPUT_MODEL]
                [{auto,0.12,1.0,1.1,1.2,1.3,2.0,pbtxt}]
```

#### Positional Arguments

FROM	Possible choices: auto, 0.12, 1.0, 1.1, 1.2, 1.3, 2.0, pbtxt The original model compatibility Default: “auto”
------	---

### Named Arguments

- |                    |   |
|--------------------|---|
| -v, --log-level    | Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0<br>set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG<br>Default: "INFO"     |
| -l, --log-path     | set log file to log messages to disk, if not specified, the logs will only be output to console   |
| -i, --input-model  | the input model<br>Default: "frozen_model.pb"   |
| -o, --output-model | the output model If OUTPUT_MODEL ends with '.pbtxt', the provided model will be converted to pbtxt format, without version conversion.<br>Default: "convert_out.pb" |

examples:

```
dp convert-from -i graph.pb -o graph_new.pb dp convert-from auto -i graph.pb -o graph_new.pb dp
convert-from 1.0 -i graph.pb -o graph_new.pb
```

### 9.3.9 neighbor-stat

Calculate neighbor statistics

```
dp neighbor-stat [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
                 [-s SYSTEM] -r RCUT -t TYPE_MAP [TYPE_MAP ...] [--one-type]
```

### Named Arguments

- |                 |   |
|-----------------|---|
| -v, --log-level | Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0<br>set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG<br>Default: "INFO" |
| -l, --log-path  | set log file to log messages to disk, if not specified, the logs will only be output to console   |
| -s, --system    | The system dir. Recursively detect systems in this directory<br>Default: "."  |
| -r, --rcut      | cutoff radius   |
| -t, --type-map  | type map  |
| --one-type      | treat all types as a single type. Used with se_atten descriptor.<br>Default: False  |

examples:

```
dp neighbor-stat -s data -r 6.0 -t O H
```

### 9.3.10 train-nvnmd

train nvnmd model

```
dp train-nvnmd [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH]
               [-i INIT_MODEL] [-r RESTART] [-s {s1,s2}]
               [--skip-neighbor-stat]
               INPUT
```

#### Positional Arguments

INPUT	the input parameter file in json format
-------	---

#### Named Arguments

-v, --log-level	Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0 set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG Default: "INFO"
-l, --log-path	set log file to log messages to disk, if not specified, the logs will only be output to console
-i, --init-model	Initialize the model by the provided path prefix of checkpoint files.
-r, --restart	Restart the training from the provided prefix of checkpoint files.
-s, --step	Possible choices: s1, s2 steps to train model of NVNMD: s1 (train CNN), s2 (train QNN) Default: "s1"
--skip-neighbor-stat	Skip calculating neighbor statistics. Sel checking, automatic sel, and model compression will be disabled. Default: False

examples:

```
dp train-nvnmd input_cnn.json -s s1 dp train-nvnmd input_qnn.json -s s2 dp train-nvnmd in-
put_cnn.json -s s1 -restart model.ckpt dp train-nvnmd input_cnn.json -s s2 -init-model model.ckpt
```

### 9.3.11 gui

Serve DP-GUI.

```
dp gui [-h] [-v {DEBUG,3,INFO,2,WARNING,1,ERROR,0}] [-l LOG_PATH] [-p PORT]
       [--bind_all]
```

**Named Arguments**

<code>-v, --log-level</code>	<p>Possible choices: DEBUG, 3, INFO, 2, WARNING, 1, ERROR, 0</p> <p>set verbosity level by string or number, 0=ERROR, 1=WARNING, 2=INFO and 3=DEBUG</p> <p>Default: "INFO"</p>
<code>-l, --log-path</code>	<p>set log file to log messages to disk, if not specified, the logs will only be output to console</p>
<code>-p, --port</code>	<p>The port to serve DP-GUI on.</p> <p>Default: 6042</p>
<code>--bind_all</code>	<p>Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).</p> <p>Default: False</p>



## INTEGRATE WITH THIRD-PARTY PACKAGES

Note that the model for inference is required to be compatible with the DeePMD-kit package. See [Model compatibility](#) for details.

### 10.1 Use deep potential with ASE

Deep potential can be set up as a calculator with ASE to obtain potential energies and forces.

```
from ase import Atoms
from deepmd.calculator import DP

water = Atoms(
    "H2O",
    positions=[(0.7601, 1.9270, 1), (1.9575, 1, 1), (1.0, 1.0, 1.0)],
    cell=[100, 100, 100],
    calculator=DP(model="frozen_model.pb"),
)
print(water.get_potential_energy())
print(water.get_forces())
```

Optimization is also available:

```
from ase.optimize import BFGS

dyn = BFGS(water)
dyn.run(fmax=1e-6)
print(water.get_positions())
```

### 10.2 Run MD with LAMMPS

#### 10.2.1 units

All units in LAMMPS except `lj` are supported. `lj` is not supported.

The most commonly used units are `metal`, since the internal units of distance, energy, force, and charge in DeePMD-kit are `\AA`, `eV`, `eV / \AA`, and `proton charge`, respectively. These units are consistent with the `metal` units in LAMMPS.

If one wants to use other units like `real` or `si`, it is welcome to do so. There is no need to do the unit conversion manually. The unit conversion is done automatically by LAMMPS.

The only thing that one needs to take care is the unit of the output of `compute deeptensor/atom`. Working with `metal` units for `compute deeptensor/atom` is totally fine, since there is no unit conversion. For other unit styles, we currently assume that the output of the `compute deeptensor/atom` command has the unit of distance and have applied the unit conversion factor of distance. If a user wants to infer quantities with units other than distance, the user is encouraged to open a GitHub feature request, so that the unit conversion factor can be added.

### 10.2.2 Enable DeePMD-kit plugin (plugin mode)

If you are using the plugin mode, enable DeePMD-kit package in LAMMPS with `plugin` command:

```
plugin load libdeepmd_lmp.so
```

After LAMMPS version `patch_24Mar2022`, another way to load plugins is to set the environmental variable `LAMMPS_PLUGIN_PATH`:

```
LAMMPS_PLUGIN_PATH=$deepmd_root/lib/deepmd_lmp
```

where `$deepmd_root` is the directory to [install C++ interface](#).

The built-in mode doesn't need this step.

### 10.2.3 pair\_style deepmd

The DeePMD-kit package provides the `pair_style deepmd`

```
pair_style deepmd models ... keyword value ...
```

- `deepmd` = style of this `pair_style`
- `models` = frozen model(s) to compute the interaction. If multiple models are provided, then only the first model serves to provide energy and force prediction for each timestep of molecular dynamics, and the model deviation will be computed among all models every `out_freq` timesteps.
- `keyword` = `out_file` or `out_freq` or `fparam` or `fparam_from_compute` or `aparam_from_compute` or `atomic` or `relative` or `relative_v` or `aparam` or `ttn`

#### Examples

```
pair_style deepmd graph.pb
pair_style deepmd graph.pb fparam 1.2
pair_style deepmd graph_0.pb graph_1.pb graph_2.pb out_file md.out out_freq 10 atomic relative 1.0
pair_coeff * * O H

pair_style deepmd cp.pb fparam_from_compute TEMP
compute TEMP all temp

pair_style deepmd ener.pb aparam_from_compute 1
compute 1 all ke/atom
```

## Description

Evaluate the interaction of the system by using [Deep Potential](#) or [Deep Potential Smooth Edition](#). It is noticed that deep potential is not a “pairwise” interaction, but a multi-body interaction.

This pair style takes the deep potential defined in a model file that usually has the .pb extension. The model can be trained and frozen by package [DeePMD-kit](#), which can have either double or single float precision interface.

The model deviation evaluates the consistency of the force predictions from multiple models. By default, only the maximal, minimal and average model deviations are output. If the key `atomic` is set, then the model deviation of force prediction of each atom will be output. The unit follows [LAMMPS units](#) and the `scale factor` is not applied.

By default, the model deviation is output in absolute value. If the keyword `relative` is set, then the relative model deviation of the force will be output, including values output by the keyword `atomic`. The relative model deviation of the force on atom  $i$  is defined by

$$E_{f_i} = \frac{|D_{f_i}|}{|f_i| + l}$$

where  $D_{f_i}$  is the absolute model deviation of the force on atom  $i$ ,  $f_i$  is the norm of the force and  $l$  is provided as the parameter of the keyword `relative`. If the keyword `relative_v` is set, then the relative model deviation of the virial will be output instead of the absolute value, with the same definition of that of the force:

$$E_{v_i} = \frac{|D_{v_i}|}{|v_i| + l}$$

If the keyword `fparam` is set, the given frame parameter(s) will be fed to the model. If the keyword `fparam_from_compute` is set, the global parameter(s) from compute command (e.g., temperature from [compute temp command](#)) will be fed to the model as the frame parameter(s). If the keyword `aparam_from_compute` is set, the atomic parameter(s) from compute command (e.g., per-atom translational kinetic energy from [compute ke/atom command](#)) will be fed to the model as the atom parameter(s). If the keyword `aparam` is set, the given atomic parameter(s) will be fed to the model, where each atom is assumed to have the same atomic parameter(s). If the keyword `ttm` is set, electronic temperatures from [fix ttm command](#) will be fed to the model as the atomic parameters.

Only a single `pair_coeff` command is used with the `deepmd` style which specifies atom names. These are mapped to LAMMPS atom types (integers from 1 to `Ntypes`) by specifying `Ntypes` additional arguments after `* *` in the `pair_coeff` command. If atom names are not set in the `pair_coeff` command, the training parameter `type_map` will be used by default. If a mapping value is specified as `NULL`, the mapping is not performed. This can be used when a `deepmd` potential is used as part of the hybrid pair style. The `NULL` values are placeholders for atom types that will be used with other potentials. If the training parameter `type_map` is not set, atom names in the `pair_coeff` command cannot be set. In this case, atom type indexes in `type.raw` (integers from 0 to `Ntypes-1`) will map to LAMMPS atom types.

Spin is specified by keywords `virtual_len` and `spin_norm`. If the keyword `virtual_len` is set, the distance between virtual atom and its corresponding real atom for each type of magnetic atoms will be fed to the model as the spin parameters. If the keyword `spin_norm` is set, the magnitude of the magnetic moment for each type of magnetic atoms will be fed to the model as the spin parameters.

## Restrictions

- The `deepmd` pair style is provided in the USER-DEEPMO package, which is compiled from the DeePMD-kit, visit the [DeePMD-kit website](#) for more information.

### 10.2.4 Compute tensorial properties

The DeePMD-kit package provides the `compute deepmd/atom` for computing atomic tensorial properties.

```
compute ID group-ID deepmd/atom model_file
```

- ID: user-assigned name of the computation
- group-ID: ID of the group of atoms to compute
- `deepmd/atom`: the style of this compute
- `model_file`: the name of the binary model file.

At this time, the training parameter `type_map` will be mapped to LAMMPS atom types.

## Examples

```
compute dipole all deepmd/atom dipole.pb
```

The result of the compute can be dumped to trajectory file by

```
dump 1 all custom 100 water.dump id type c_dipole[1] c_dipole[2] c_dipole[3]
```

## Restrictions

- The `deepmd/atom` compute is provided in the USER-DEEPMO package, which is compiled from the DeePMD-kit, visit the [DeePMD-kit website](#) for more information.
- For the issue of using a unit style for `compute deepmd/atom`, refer to the discussions in [units](#) of this page.

### 10.2.5 Long-range interaction

The reciprocal space part of the long-range interaction can be calculated by LAMMPS command `kpace_style`. To use it with DeePMD-kit, one writes

```
pair_style      deepmd graph.pb
pair_coeff  * *
kpace_style     ppm 1.0e-5
kpace_modify    gewald 0.45
```

Please notice that the DeePMD does nothing to the direct space part of the electrostatic interaction, because this part is assumed to be fitted in the DeePMD model (the direct space cut-off is thus the cut-off of the DeePMD model). The splitting parameter `gewald` is modified by the `kpace_modify` command.

## 10.2.6 Use of the centroid/stress/atom to get the full 3x3 “atomic-virial”

The DeePMD-kit also allows the computation of per-atom stress tensor defined as:

$$d_{\text{atom}} = - \sum_m (\mathbf{r}_n - \mathbf{r}_m) \frac{de_m}{d\mathbf{r}_n}$$

Where  $\mathbf{r}_n$  is the atomic position of nth atom,  $\mathbf{v}_n$  velocity of the atom and  $\frac{de_m}{d\mathbf{r}_n}$  the derivative of the atomic energy.

In LAMMPS one can get the per-atom stress using the command `centroid/stress/atom`:

```
compute ID group-ID centroid/stress/atom NULL virial
```

see [LAMMPS doc page](#) for more details on the meaning of the keywords.

Changed in version v2.2.3: v2.2.2 or previous versions passed per-atom stress (`cvatom`) with the per-atom pressure tensor, which is inconsistent with [LAMMPS's definition](#). LAMMPS defines per-atom stress as the negative of the per-atom pressure tensor. Such behavior is corrected in v2.2.3.

### Examples

In order of computing the 9-component per-atom stress

```
compute stress all centroid/stress/atom NULL virial
```

Thus `c_stress` is an array with 9 components in the order `xx,yy,zz,xy,xz,yz,yx,zx,zy`.

If you use this feature please cite D. Tisi, L. Zhang, R. Bertossa, H. Wang, R. Car, S. Baroni - [arXiv preprint arXiv:2108.10850](#), 2021

## 10.2.7 Computation of heat flux

Using a per-atom stress tensor one can, for example, compute the heat flux defined as:

$$\mathbf{J} = \sum_n e_n \mathbf{v}_n + \sum_{n,m} (\mathbf{r}_m - \mathbf{r}_n) \frac{de_m}{d\mathbf{r}_n} \mathbf{v}_n$$

to compute the heat flux with LAMMPS:

```
compute ke_ID all ke/atom
compute pe_ID all pe/atom
compute stress_ID group-ID centroid/stress/atom NULL virial
compute flux_ID all heat/flux ke_ID pe_ID stress_ID
```

### Examples

```
compute ke all ke/atom
compute pe all pe/atom
compute stress all centroid/stress/atom NULL virial
compute flux all heat/flux ke pe stress
```

`c_flux` is a global vector of length 6. The first three components are the  $x$ ,  $y$  and  $z$  components of the full heat flux vector. The others are the components of the so-called convective portion, see [LAMMPS doc page](#) for more details.

If you use these features please cite D. Tisi, L. Zhang, R. Bertossa, H. Wang, R. Car, S. Baroni - arXiv preprint [arXiv:2108.10850](#), 2021

## 10.3 Run path-integral MD with i-PI

The i-PI works in a client-server model. The i-PI provides the server for integrating the replica positions of atoms, while the DeePMD-kit provides a client named `dp_ipi` that computes the interactions (including energy, forces and virials). The server and client communicate via the Unix domain socket or the Internet socket. Installation instructions for i-PI can be found [here](#). The client can be started by

```
i-pi input.xml &
dp_ipi water.json
```

It is noted that multiple instances of the client allow for computing, in parallel, the interactions of multiple replicas of the path-integral MD.

`water.json` is the parameter file for the client `dp_ipi`, and an example is provided:

```
{
  "verbose":          false,
  "use_unix":          true,
  "port":              31415,
  "host":              "localhost",
  "graph_file":        "graph.pb",
  "coord_file":        "conf.xyz",
  "atom_type" : {
    "OW":              0,
    "HW1":              1,
    "HW2":              1
  }
}
```

The option `use_unix` is set to `true` to activate the Unix domain socket, otherwise, the Internet socket is used.

The option `port` should be the same as that in `input.xml`:

```
<port>31415</port>
```

The option `graph_file` provides the file name of the frozen model. The model can have either double or single float precision interface.

The `dp_ipi` gets the atom names from an [XYZ file](#) provided by `coord_file` (meanwhile ignores all coordinates in it) and translates the names to atom types by rules provided by `atom_type`.

## 10.4 Running MD with GROMACS

### 10.4.1 DP/MM Simulation

This part gives a simple tutorial on how to run a DP/MM simulation for methane in water, which means using DP for methane and TIP3P for water. All relevant files can be found in `examples/methane`.

#### Topology Preparation

Similar to QM/MM simulation, the internal interactions (including bond, angle, dihedrals, LJ, Columb) of the region described by a neural network potential (NNP) have to be turned off. In GROMACS, bonded interactions can be turned off by modifying [ `bonds` ], [ `angles` ], [ `dihedrals` ] and [ `pairs` ] sections. And LJ and Columb interactions must be turned off by [ `exclusions` ] section.

For example, if one wants to simulate ethane in water, using DeepPotential for methane and TIP3P for water, the topology of methane should be like the following (as presented in `examples/methane/methane.itp`):

```
[ atomtypes ]
;name btype mass charge ptype sigma epsilon
c3 c3 0.0 0.0 A 0.339771 0.451035
hc hc 0.0 0.0 A 0.260018 0.087027

[ moleculetype ]
;name nrexcl
methane 3

[ atoms ]
; nr type resnr residue atom cgnr charge mass
1 c3 1 MOL C1 1 -0.1068 12.010
2 hc 1 MOL H1 2 0.0267 1.008
3 hc 1 MOL H2 3 0.0267 1.008
4 hc 1 MOL H3 4 0.0267 1.008
5 hc 1 MOL H4 5 0.0267 1.008

[ bonds ]
; i j func b0 kb
1 2 5
1 3 5
1 4 5
1 5 5

[ exclusions ]
; ai aj1 aj2 aj3 aj4
1 2 3 4 5
2 1 3 4 5
3 1 2 4 5
4 1 2 3 5
5 1 2 3 4
```

For comparison, the original topology file generated by `acpype` will be:

```
; methane_GMX.itp created by acpype (v: 2021-02-05T22:15:50CET) on Wed Sep 8 01:21:53 2021

[ atomtypes ]
;name bond_type mass charge ptype sigma epsilon Amb
```

(continues on next page)

(continued from previous page)

```

c3      c3      0.00000 0.00000  A    3.39771e-01  4.51035e-01 ; 1.91  0.1078
hc      hc      0.00000 0.00000  A    2.60018e-01  8.70272e-02 ; 1.46  0.0208

[ moleculetype ]
;name          nrexcl
methane        3

[ atoms ]
;  nr  type  resi  res  atom  cgnr      charge      mass      ; qtot  bond_type
   1   c3    1    MOL   C1    1    -0.106800    12.01000 ; qtot -0.107
   2   hc    1    MOL   H1    2     0.026700     1.00800 ; qtot -0.080
   3   hc    1    MOL   H2    3     0.026700     1.00800 ; qtot -0.053
   4   hc    1    MOL   H3    4     0.026700     1.00800 ; qtot -0.027
   5   hc    1    MOL   H4    5     0.026700     1.00800 ; qtot  0.000

[ bonds ]
;  ai    aj  funct    r          k
   1     2    1     1.0970e-01  3.1455e+05 ;    C1 - H1
   1     3    1     1.0970e-01  3.1455e+05 ;    C1 - H2
   1     4    1     1.0970e-01  3.1455e+05 ;    C1 - H3
   1     5    1     1.0970e-01  3.1455e+05 ;    C1 - H4

[ angles ]
;  ai    aj    ak    funct    theta      cth
   2     1     3     1    1.0758e+02  3.2635e+02 ;    H1 - C1    - H2
   2     1     4     1    1.0758e+02  3.2635e+02 ;    H1 - C1    - H3
   2     1     5     1    1.0758e+02  3.2635e+02 ;    H1 - C1    - H4
   3     1     4     1    1.0758e+02  3.2635e+02 ;    H2 - C1    - H3
   3     1     5     1    1.0758e+02  3.2635e+02 ;    H2 - C1    - H4
   4     1     5     1    1.0758e+02  3.2635e+02 ;    H3 - C1    - H4

```

## DeepMD Settings

Before running simulations, we need to tell GROMACS to use DeepPotential by setting the environment variable `GMX_DEEPMO_INPUT_JSON`:

```
export GMX_DEEPMO_INPUT_JSON=input.json
```

Then, in your working directories, we have to write `input.json` file:

```
{
  "graph_file": "/path/to/graph.pb",
  "type_file": "type.raw",
  "index_file": "index.raw",
  "lambda": 1.0,
  "pbc": false
}
```

Here is an explanation for these settings:

- `graph_file`: The graph file (with suffix `.pb`) generated by `dp freeze` command
- `type_file`: File to specify DP atom types (in space-separated format). Here, `type.raw` looks like

```
1 0 0 0 0
```



- `index_file`: File containing indices of DP atoms (in space-separated format), which should be consistent with the indices' order in `.gro` file but starting from zero. Here, `index.raw` looks like

```
0 1 2 3 4
```

- `lambda`: Optional, default 1.0. Used in alchemical calculations.
- `pgc`: Optional, default true. If true, the GROMACS periodic condition is passed to DeepMD.

## Run Simulation

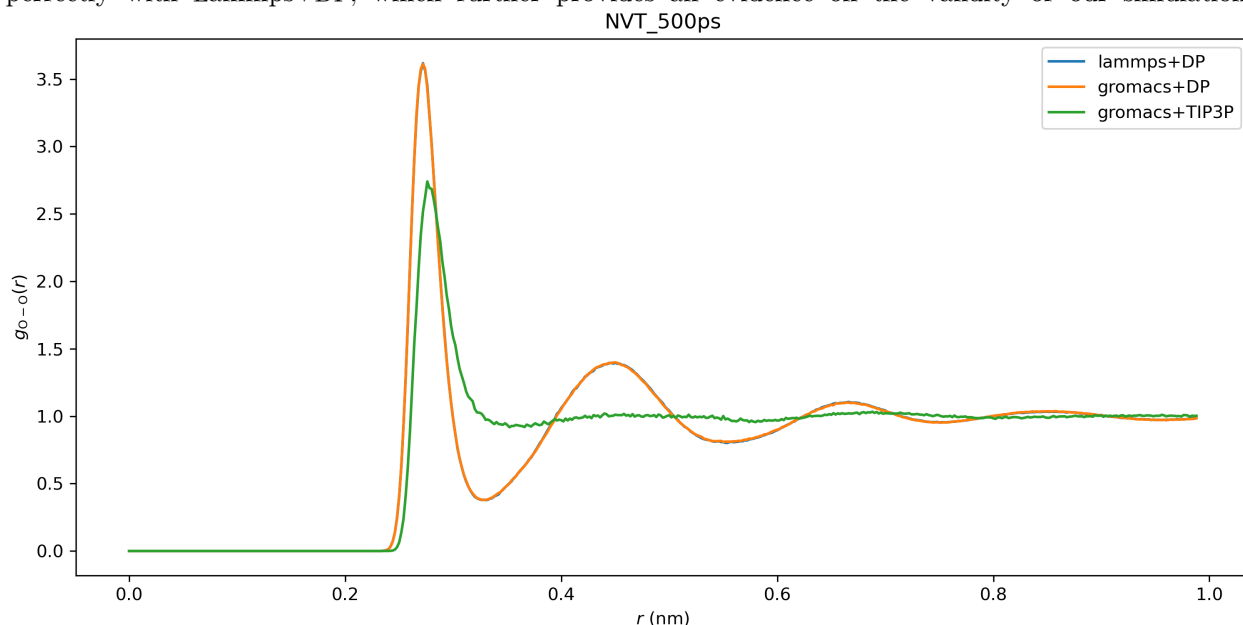
Finally, you can run GROMACS using `gmx mdrun` as usual.

### 10.4.2 All-atom DP Simulation

This part gives an example of how to simulate all atoms described by a DeepPotential with Gromacs, taking water as an example. Instead of using [ `exclusions` ] to turn off the non-bonded energies, we can simply do this by setting LJ parameters (i.e. `epsilon` and `sigma`) and partial charges to 0, as shown in `examples/water/gmx/water.top`:

```
[ atomtypes ]
; name      at.num  mass    charge ptype  sigma    epsilon
HW          1      1.008   0.0000  A    0.00000e+00  0.00000e+00
OW          8      16.00   0.0000  A    0.00000e+00  0.00000e+00
```

As mentioned in the above section, `input.json` and relevant files (`index.raw`, `type.raw`) should also be created. Then, we can start the simulation under the NVT ensemble and plot the radial distribution function (RDF) by `gmx rdf` command. We can see that the RDF given by Gromacs+DP matches perfectly with LAMMPS+DP, which further provides an evidence on the validity of our simulation.



However, we still recommend you run an all-atom DP simulation using LAMMPS since it is more stable and efficient.

## 10.5 Interfaces out of DeePMD-kit

The codes of the following interfaces are not a part of the DeePMD-kit package and maintained by other repositories. We list these interfaces here for user convenience.

### 10.5.1 dpdata

dpdata provides the `predict` method for `System` class:

```
import dpdata

dsys = dpdata.LabeledSystem("OUTCAR")
dp_sys = dsys.predict("frozen_model_compressed.pb")
```

By inferring with the DP model `frozen_model_compressed.pb`, dpdata will generate a new labeled system `dp_sys` with inferred energies, forces, and virials.

### 10.5.2 OpenMM plugin for DeePMD-kit

An OpenMM plugin is provided from [JingHuangLab/openmm\\_deepmd\\_plugin](#), written by the [Huang Lab](#) at Westlake University.

### 10.5.3 Amber interface to DeePMD-kit

Starting from [AmberTools24](#), `sander` includes an interface to the DeePMD-kit, which implements the [Deep Potential Range Corrected \(DPRc\) correction](#). The DPRc model and the interface were developed by the [York Lab](#) from Rutgers University. More details are available in

- [Amber Reference Manuals](#), providing documentation for how to enable the interface and the `&dprc` namelist;
- [GitLab RutgersLBSR/AmberDPRc](#), providing examples mdin files;
- [DP-Amber](#), a tiny tool to convert Amber trajectory to DPRc training data;
- The original DPRc paper.

### 10.5.4 CP2K interface to DeePMD-kit

[CP2K v2024.2](#) adds an interface to the DeePMD-kit for molecular dynamics. Read the [CP2K manual](#) for details.

### 10.5.5 DP-GEN

DP-GEN provides a workflow to generate accurate DP models by calling DeePMD-kit's command line interface (CLI) in the local or remote server. Details can be found in [this paper](#).

### 10.5.6 MLatom

MLatom provides an interface to the DeePMD-kit within MLatom's workflow by calling DeePMD-kit's CLI. Details can be found in [this paper](#).

### 10.5.7 ABACUS

ABACUS can run molecular dynamics with a DP model. User is required to [build ABACUS with DeePMD-kit](#).



## USE NVNMD

### 11.1 Introduction

NVNMD stands for non-von Neumann molecular dynamics.

This is the training code we used to generate the results in our paper entitled “Accurate and Efficient Molecular Dynamics based on Machine Learning and non von Neumann Architecture”, which has been accepted by npj Computational Materials (DOI: [10.1038/s41524-022-00773-z](https://doi.org/10.1038/s41524-022-00773-z)).

Any user can follow two consecutive steps to run molecular dynamics (MD) on the proposed NVNMD computer, which has been released online: (i) to train a machine learning (ML) model that can decently reproduce the potential energy surface (PES); and (ii) to deploy the trained ML model on the proposed NVNMD computer, then run MD there to obtain the atomistic trajectories.

#### 11.1.1 Training

Our training procedure consists of not only continuous neural network (CNN) training but also quantized neural network (QNN) training which uses the results of CNN as inputs. It is performed on CPU or GPU by using the training codes we open-sourced online.

To train an ML model that can decently reproduce the PES, a training and testing data set should be prepared first. This can be done by using either the state-of-the-art active learning tools or the outdated (i.e., less efficient) brute-force density functional theory (DFT)-based ab-initio molecular dynamics (AIMD) sampling.

If you just want to simply test the training function, you can use the example in the `$deepmd_source_dir/examples/nvnmd` directory. If you want to fully experience training and running MD functions, you can download the complete example from the [website](#).

Then, copy the data set to the working directory

```
mkdir -p $workspace
cd $workspace
mkdir -p data
cp -r $dataset data
```

where `$dataset` is the path to the data set and `$workspace` is the path to the working directory.

### 11.1.2 Input script

Create and go to the training directory.

```
mkdir train
cd train
```

Then copy the input script `train_cnn.json` and `train_qnn.json` to the directory `train`

```
cp -r $deepmd_source_dir/examples/nvnmd/train/train_cnn.json train_cnn.json
cp -r $deepmd_source_dir/examples/nvnmd/train/train_qnn.json train_qnn.json
```

The structure of the input script is as follows

```
{
  "nvnmd" : {},
  "learning_rate" : {},
  "loss" : {},
  "training": {}
}
```

#### nvnmd

The “nvnmd” section is defined as

```
{
  "version": 0,
  "max_nnei":128,
  "net_size":128,
  "sel":[60, 60],
  "rcut":6.0,
  "rcut_smth":0.5,
  "type_map": ["Ge", "Te"]
}
```

where items are defined as:

Item	Mean	Optional Value
version	the version of network structure	0 or 1
max_nnei	the maximum number of neighbors that do not distinguish element types	128 or 256
net_size	the size of nueral network	128
sel	the number of neighbors	version 0: integer list of lengths 1 to 4 are acceptable; version 1: integer
rcut	the cutoff radial	(0, 8.0]
rcut_smth	the smooth cutoff parameter	(0, 8.0]
type_map	mapping atom type to the name (str) of the type	string list, optional

Multiple versions of the nvnmd model correspond to different network structures. `nvnmd-v0` and `nvnmd-v1` differ in the following ways:

1. `nvnmd-v0` and `nvnmd-v1` use the `se_a` descriptor and `se_atten` descriptor, respectively
2. `nvnmd-v0` has 1 set of parameters for each element and supports up to 4 element types. `nvnmd-v1` shares 1 set of parameters for each element and supports up to 31 types.

3. `nvnmmd-v0` distinguishes between neighboring atoms, so `sel` is a list of integers. `nvnmmd-v1` does not distinguish between neighboring atoms, so `sel` is an integer.

## learning\_rate

The “learning\_rate” section is defined as

```
{
  "type": "exp",
  "start_lr": 1e-3,
  "stop_lr": 3e-8,
  "decay_steps": 5000
}
```

where items are defined as:

Item	Mean	Optional Value
type	learning rate variant type	exp
start_lr	the learning rate at the beginning of the training	a positive real number
stop_lr	the desired learning rate at the end of the training	a positive real number
decay_stops	the learning rate is decaying every {decay_stops} training steps	a positive integer

## loss

The “loss” section is defined as

```
{
  "start_pref_e": 0.02,
  "limit_pref_e": 2,
  "start_pref_f": 1000,
  "limit_pref_f": 1,
  "start_pref_v": 0,
  "limit_pref_v": 0
}
```

where items are defined as:

Item	Mean	Optional Value
start_pref_e	the loss factor of energy at the beginning of the training	zero or positive real number
limit_pref_e	the loss factor of energy at the end of the training	zero or positive real number
start_pref_f	the loss factor of force at the beginning of the training	zero or positive real number
limit_pref_f	the loss factor of force at the end of the training	zero or positive real number
start_pref_v	the loss factor of virial at the beginning of the training	zero or positive real number
limit_pref_v	the loss factor of virial at the end of the training	zero or positive real number

## training

The “training” section is defined as

```
{
  "seed": 1,
  "stop_batch": 1000000,
  "numb_test": 1,
  "disp_file": "lcurve.out",
  "disp_freq": 1000,
  "save_ckpt": "model.ckpt",
  "save_freq": 10000,
  "training_data": {
    "systems": ["system1_path", "system2_path", "..."],
    "set_prefix": "set",
    "batch_size": ["batch_size_of_system1", "batch_size_of_system2", "..."]
  }
}
```

where items are defined as:

Item	Mean	Optional Value
seed	the randome seed	a integer
stop_batch	the total training steps	a positive integer
numb_test	the accuracy is test by using {numb_test} sample	a positive integer
disp_file	the log file where the training message display	a string
disp_freq	display frequency	a positive integer
save_ckpt	path prefix of check point files	a string
save_freq	save frequency	a positive integer
systems	a list of data directory which contains the dataset	string list
set_prefix	the prefix of dataset	a string
batch_size	a list of batch size of corresponding dataset	a integer list

### 11.1.3 Training

Training can be invoked by

```
# step1: train CNN
dp train-nvnmd train_cnn.json -s s1
# step2: train QNN
dp train-nvnmd train_qnn.json -s s2
```

After the training process, you will get two folders: `nvnmmd_cnn` and `nvnmmd_qnn`. The `nvnmmd_cnn` contains the model after continuous neural network (CNN) training. The `nvnmmd_qnn` contains the model after quantized neural network (QNN) training. The binary file `nvnmmd_qnn/model.pb` is the model file that is used to perform NVNMD in the server [<http://nvnmmd.picp.vip>].

You can also restart the CNN training from the path prefix of checkpoint files (`nvnmmd_cnn/model.ckpt`) by

```
dp train-nvnmd train_cnn.json -r nvnmmd_cnn/model.ckpt -s s1
```

You can also initialize the CNN model and train it by



```
mv nvnmmd_cnn nvnmmd_cnn_bck
cp train_cnn.json train_cnn2.json
# please edit train_cnn2.json
dp train-nvnmmd train_cnn2.json -s s1 -i nvnmmd_cnn_bck/model.ckpt
```

## 11.2 Testing

The frozen model can be used in many ways. The most straightforward testing can be invoked by

```
mkdir test
dp test -m ./nvnmmd_qnn/frozen_model.pb -s path/to/system -d ./test/detail -n 99999 -l test/output.
↪ log
```

where the frozen model file to import is given via the `-m` command line flag, the path to the testing data set is given via the `-s` command line flag, and the file containing details of energy, forces and virials accuracy is given via the `-d` command line flag, the amount of data for testing is given via the `-n` command line flag.

## 11.3 Running MD in Bohrium

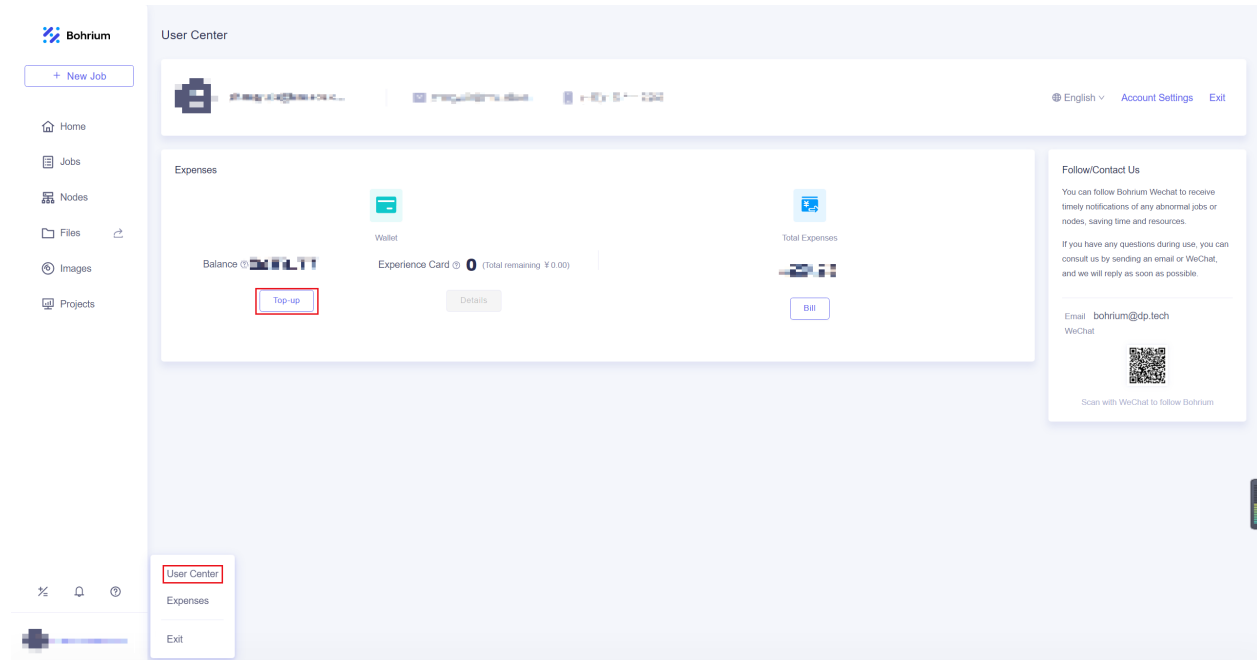
After CNN and QNN training, you can upload the ML model to our online NVNMD system and run MD there through Bohrium (<https://bohrium.dp.tech>). Bohrium is a research platform designed for AI for Science Era. For more information, please refer to [Bohrium Introduction](#).

### 11.3.1 Registration

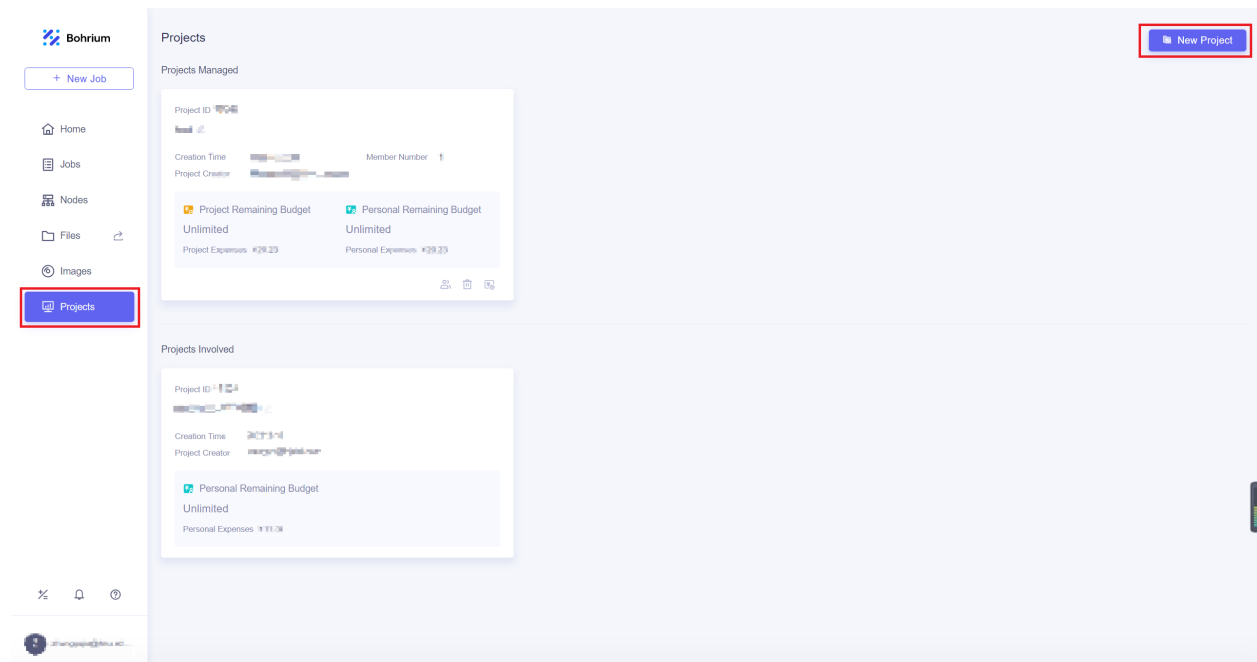
Click [here](#) to register a Bohrium account. If you already have an account for other DP products, you can skip this step and log in directly.

### 11.3.2 Top-up and create a project

After entering the homepage, you can click on the **User Center** in the lower left corner to top-up by yourself.



After completing the top-up, click on the **Projects**, and then click **New Project** in the upper right corner of the page. Give the project a name that is easy for you to recognize and click OK. If the project has other collaborators, you can refer to [Project Collaboration](#) for more information.



### 11.3.3 Run job

We will use Utility to submit jobs, you can install it with the following command

```
pip install lbg
```

When using the Lebesgue Utility for the first time, you need to configure your account by

```
lbg config account
```

Enter your Bohrium account and the corresponding password.

Then you need prepare the configuration file `job.json`, the configuration file is as follows

```
{
  "job_name": "test",
  "command": "/usr/bin/lmp_mpi < in.lmp;",
  "log_file": "OUTCAR",
  "machine_type": "c4_m16_cpu",
  "job_type": "container",
  "image_name": "lammps_dp:29Sep2021",
  "platform": "hnugba",
  "region": "default",
  "project_id": 0000
}
```

where items are defined as:

Item	Mean	Optional Value
job_name	the name of computing job, which can be named freely	a string
com- mand	the command to be executed on the computing node	a string
log_file	the log file that can be viewed at any time during the calculation process, which can be viewed on the Bohrium “Jobs” page	a string
ma- chine_type	the machine type used for the job	“c1_m4_cpu”, “c4_m16_cpu”, “c8_m32_cpu”
job_type	the job type	“container”
im- age_name	the image name used for the job	“lammps_dp:29Sep2021”
plat- form	resource provider	“hnugba”
project_id	the project ID to which the job belongs, which can be viewed on the “Projects” page	a integer

Notice The task will use 4 CPU cores for computation, so do not repeatedly use the `mpirun` command, otherwise an error will be reported. All 0000 after “project\_id” need to be replaced with your own project ID, which can be viewed on the “Projects” page. Also, the JSON file format requires that no commas be added after the last field within the {}, otherwise, there will be a syntax error. Please check the [documentation](#) for the latest hardware configuration information.

In addition, it is necessary to prepare input script of the MD simulation, the ML model named `model.pb` obtained by QNN training and data files containing information required for running an MD simulation (e.g., `coord.lmp` containing initial atom coordinates).

In the input script, one needs to specify the pair style as follows

```
pair_style nvnmmd model.pb
pair_coeff * *
```

where `model.pb` is the path to model.

After preparing the configuration file and the required files for calculation, using Lebesgue Utility to submit the job

```
lbg job submit -i job.json -p ./
```

where the configuration file for the job is given via the `-i` command line flag, the directory where the input files are located is given via the `-p` command line flag. Bohrium will package and upload the specified directory, and after decompressing it on the computing node, it will switch the working directory to that directory.

After the job is submitted successfully, the JOB ID and JOB GROUP ID will be output.

### 11.3.4 Check job status

After successfully submitting the job, you can view the progress and related logs of the submitted jobs on the Jobs page.

The screenshot displays the Bohrium web interface. On the left is a sidebar with navigation options: Home, Jobs (highlighted with a red box), Nodes, Files, Images, and Projects. The main area shows a table of jobs with columns for Job ID, Job Name, and Job Status. A job with ID 7742231 and name 'test' is shown with a 'Completed' status. Below the table are buttons for 'Select All', '0 Jobs Selected', 'Download', and 'Terminate'. To the right, the 'Job Details' section provides metadata: Image (lammps\_dp29Sep2021), Max run time (-min), Machine type (c8\_m32\_cpu), Directory (/home/zy/test), Platform (hnu), Region (default), Command (/usr/bin/lmp\_mpi < in.lmp), backward files (-), and Automatic Download? (No). Below this is the 'Log' section, which contains a table of performance metrics and a detailed log output. The log output includes a table with columns for Section, min time, avg time, max time, [Nvaravg], and [Ntotal]. The log also shows a table of result files with columns for File Name, Update Time, Size, and Operation. The 'Computing Node' section at the bottom shows a table with columns for Machine Type, Cores, Memory, GPU, Status, and Operation.

Section	min time	avg time	max time	[Nvaravg]	[Ntotal]
Pair	0.078213	0.15639	0.24332	14.5	55.60
Neigh	0.026712	0.026975	0.027214	0.1	9.59
Comm	0.000000	0.000000	0.000000	0.0	0.00

File Name	Update Time	Size	Operation
out.zip	2023-07-04 20:59:33	1.46MB	Download Save to Data Disk

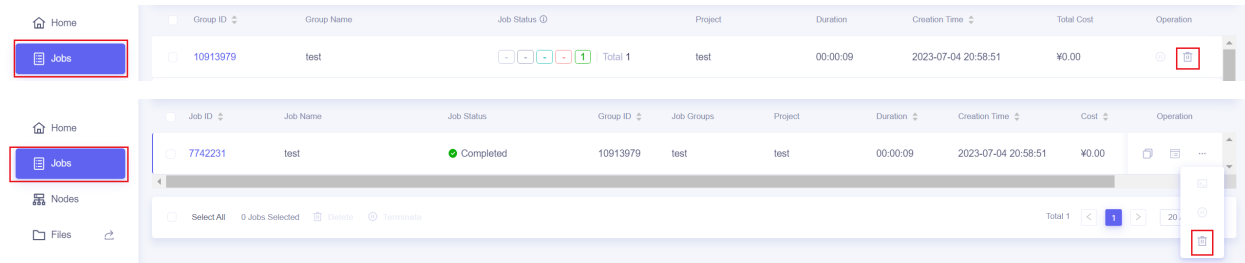
Machine Type	Cores	Memory	GPU	Status	Operation
c8_m32_cpu	8 Cores	32 GB	-	Completed	

### 11.3.5 Terminate and delete jobs

You can choose between **terminate** and **delete** operations.

- **Terminate:** To end running jobs/job groups in advance, save the generated result files, and the status of the terminated jobs will be changed to “completed”.
- **Delete:** To end running jobs/job groups, the status of the jobs will be changed to “failed”. Job result files will be deleted, and the jobs/job groups disappear from the list. The delete operation cannot be undone.

The Jobs page provides buttons to end jobs and job groups



You can also use the Lebesgue Utility tool to end jobs

```
lbg jobgroup terminate <JOB GROUP ID>
```

```
lbg job terminate <JOB ID>
```

```
lbg jobgroup rm <JOB GROUP ID>
```

```
lbg job rm <JOB ID>
```

### 11.3.6 Download Results

After the calculation is completed, you can download the results on the Jobs page, or save them to the data disk.

**Job Details**

Image	lammps dp29Sep2021	Max run time	-min	Machine type	c8_m32_cpu	
Directory	/home/czj/test	Platform	hnu	Region	default	
Command	/usr/bin/lmp_mpi < in.lmp;		backward files	-	Automatic Download?	No

Automatic Download Directory

**Log**

```

180  6010.1871  612.75139  7222.9385  342.93938  -13140.043  410636.35
Loop time of 0.281256 on 8 procs for 180 steps with 13824 atoms
Performance: 30.719 ns/day, 0.781 hours/ns, 355.547 timesteps/s
82.7% CPU use with 8 MPI tasks x no OpenMP threads
MPI task timing breakdown:
Section | min time | avg time | max time | [Xvaravg] | [Xtotal]
-----|-----|-----|-----|-----|-----
Pair | 0.079213 | 0.15639 | 0.24332 | 14.5 | 55.60
Neigh | 0.026712 | 0.026975 | 0.027214 | 0.1 | 9.59
Comm | 0.0007616 | 0.0007600 | 0.0007616 | 0.0 | 0.00

```

**Result Files**

File Name	Update Time	Size	Operation
out.zip	2023-07-04 20:59:33	1.46MB	<a href="#">Download</a> <a href="#">Save to Data Disk</a>

**Computing Node**

Machine Type	Cores	Memory	GPU	Status	Operation
c8_m32_cpu	8 Cores	32 GB	-	Completed	<a href="#">Delete</a>

You can also download it using the commands of Lebesgue Utility

```
lbg job download <JOB ID>
```

or

```
lbg jobgroup download <JOB GROUP ID>
```

## 11.4 Running MD in Nvnmd website

After CNN and QNN training, you can upload the ML model to our online NVNMD system and run MD there.

### 11.4.1 Account application

The server website of NVNMD is available at <http://nvnmd.picp.vip>. You can visit the URL and enter the login interface.

# NVNMD

[User guide](#)

---

[Switch to Chinese](#)

Username	<input type="text"/>
Password	<input type="password"/>

To apply for an account, please email:  
jie\_liu@hnu.edu.cn, liujie@uw.edu

To obtain an account, please send your application to the email ([jie\\_liu@hnu.edu.cn](mailto:jie_liu@hnu.edu.cn), [liujie@uw.edu](mailto:liujie@uw.edu)). The username and password will be sent to you by email.

### 11.4.2 Adding task

After successfully obtaining the account, enter the username and password in the login interface, and click “Login” to enter the homepage.

## NVNMD

Current user: test1 [Logout](#)

Remaining calculation time: 6:22:29

[Add a new task](#)[Operation records](#)Calculation records [Refresh](#)[Clear calculation records](#)

Submission time	Task name	Input script	Calculation status	Cancel calculation	Calculation time	Download results	Delete record
-----------------	-----------	--------------	--------------------	--------------------	------------------	------------------	---------------

The homepage displays the remaining calculation time and all calculation records not deleted. Click **Add a new task** to enter the interface for adding a new task.

## NVNMD

Current user: test1 [Return to home page](#)

Remaining calculation time: 6:22:29

Task name	<input type="text" value="test"/>
Upload mode <sup>?</sup>	<input type="button" value="Manual upload"/> <input type="button" value="Automatic upload"/>
Input script	<input type="button" value="Browse..."/> in.Imp
Model file	<input type="button" value="Browse..."/> model.pb
Data files	<input type="button" value="Browse..."/> coord.Imp

- Task name: name of the task
- Upload mode: two modes of uploading results to online data storage, including **Manual upload** and **Automatic upload**. Results need to be uploaded manually to online data storage with **Manual upload** mode and will be uploaded automatically with **Automatic upload** mode.
- Input script: input file of the MD simulation.

In the input script, one needs to specify the pair style as follows

```
pair_style nvnmd model.pb
pair_coeff * *
```

- Model file: the ML model named `model.pb` obtained by QNN training.
- Data files: data files containing the information required for running an MD simulation (e.g., `coord.lmp` containing initial atom coordinates).

Next, you can click **Submit** to submit the task and then automatically return to the homepage.

## NVNMD

---

Current user:test1 [Logout](#)  
 Remaining calculation time:6:22:29  
[Add a new task](#)  
[Operation records](#)

Calculation records [Refresh](#)  
[Clear calculation records](#)

Submission time	Task name	Input script	Calculation status	Cancel calculation	Calculation time	Download results	Delete record
2022-05-17 21:31:20	test	in.Imp	Running	<a href="#">Cancel</a>			

Then, click **Refresh** to view the latest status of all calculation tasks.

### 11.4.3 Cancelling calculation

For the task whose calculation status is **Pending** and **Running**, you can click the corresponding **Cancel** on the homepage to stop the calculation.

## NVNMD

---

Current user:test1 [Logout](#)  
 Remaining calculation time:6:21:09  
[Add a new task](#)  
[Operation records](#)

Calculation records [Refresh](#)  
[Clear calculation records](#)

Submission time	Task name	Input script	Calculation status	Cancel calculation	Calculation time	Download results	Delete record
2022-05-17 21:31:20	test	in.Imp	Cancelled		0:01:20	<a href="#">Package</a> <a href="#">Separate files</a>	<a href="#">Delete</a>



### 11.4.4 Downloading results

For the task whose calculation status is **Completed**, **Failed** and **Cancelled**, you can click the corresponding **Package** or **Separate files** in the **Download results** bar on the homepage to download results.

Click **Package** to download a zipped package of all files including input files and output results.

#### NVNMD

Current user: test1 [Return to home page](#)

Remaining calculation time: 6:21:09

##### Files

Name	Size	Download directly	Download from online data storage	Upload to online data storage②
output.zip	1.2 MB	<a href="#">Download</a>		<a href="#">Upload</a>

Click **Separate files** to download the required separate files.

#### NVNMD

Current user: test1 [Return to home page](#)

Remaining calculation time: 6:21:09

##### Files

Name	Size	Download directly	Download from online data storage	Upload to online data storage②
coord.lmp	15.4 KB	<a href="#">Download</a>		<a href="#">Upload</a>
in.lmp	3.1 KB	<a href="#">Download</a>		<a href="#">Upload</a>
lammps.xyz	2.1 MB	<a href="#">Download</a>		<a href="#">Upload</a>
log.lammps	14.0 KB	<a href="#">Download</a>		<a href="#">Upload</a>
model.pb	8.1 MB	<a href="#">Download</a>		<a href="#">Upload</a>
result.out	13.5 KB	<a href="#">Download</a>		<a href="#">Upload</a>

If **Manual upload** mode is selected or the file has expired, click **Upload** on the download interface to upload manually.

### 11.4.5 Deleting record

For the task no longer needed, you can click the corresponding **Delete** on the homepage to delete the record. Records cannot be retrieved after deletion.

### 11.4.6 Clearing records

Click `Clear calculation records` on the homepage to clear all records.

Records cannot be retrieved after clearing.

**FAQS**

As a consequence of differences in computers or systems, problems may occur. Some common circumstances are listed as follows. In addition, some frequently asked questions are listed as follows. If other unexpected problems occur, you're welcome to contact us for help.

## 12.1 How to tune Fitting/embedding-net size ?

Here are some test forms on fitting-net size tuning or embedding-net size tuning performed on several different systems.

### 12.1.1 Al2O3

**Fitting net size tuning form on Al2O3: (embedding-net size: [25,50,100])**

Fitting-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	1.742252e-02	7.259383e-05	4.014115e-02
[80,80,80]	1.799349e-02	7.497287e-05	4.042977e-02
[40,40,40]	1.799036e-02	7.495984e-05	4.068806e-02
[20,20,20]	1.834032e-02	7.641801e-05	4.094784e-02
[10,10,10]	1.913058e-02	7.971073e-05	4.154775e-02
[5,5,5]	1.932914e-02	8.053808e-05	4.188052e-02
[4,4,4]	1.944832e-02	8.103467e-05	4.217826e-02
[3,3,3]	2.068631e-02	8.619296e-05	4.300497e-02
[2,2,2]	2.267962e-02	9.449840e-05	4.413609e-02
[1,1,1]	2.813596e-02	1.172332e-04	4.781115e-02
[]	3.135002e-02	1.306251e-04	5.373120e-02

[] means no hidden layer, but there is still a linear output layer. This situation is equal to the linear regression.

**Embedding net size tuning form on Al<sub>2</sub>O<sub>3</sub>: (Fitting-net size: [240,240,240])**

Embedding-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	1.742252e-02	7.259383e-05	4.014115e-02
[10,20,40]	2.909990e-02	1.212496e-04	4.734667e-02
[5,10,20]	3.357767e-02	1.399070e-04	5.706385e-02
[4,8,16]	6.060367e-02	2.525153e-04	7.333304e-02
[3,6,12]	5.656043e-02	2.356685e-04	7.793539e-02
[2,4,8]	5.277023e-02	2.198759e-04	7.459995e-02
[1,2,4]	1.302282e-01	5.426174e-04	9.672238e-02

**12.1.2 Cu****Fitting net size tuning form on Cu: (embedding-net size: [25,50,100])**

Fitting-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	4.135548e-02	1.615449e-04	8.940946e-02
[20,20,20]	4.323858e-02	1.689007e-04	8.955762e-02
[10,10,10]	4.399364e-02	1.718502e-04	8.962891e-02
[5,5,5]	4.468404e-02	1.745470e-04	8.970111e-02
[4,4,4]	4.463580e-02	1.743586e-04	8.972011e-02
[3,3,3]	4.493758e-02	1.755374e-04	8.971303e-02
[2,2,2]	4.500736e-02	1.758100e-04	8.973878e-02
[1,1,1]	4.542073e-02	1.774247e-04	8.964761e-02
[]	4.545168e-02	1.775456e-04	8.983201e-02

**Embedding net size tuning form on Cu: (Fitting-net size: [240,240,240])**

Embedding-net size	Energy L2err(eV)	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	4.135548e-02	1.615449e-04	8.940946e-02
[20,40,80]	4.203562e-02	1.642016e-04	8.925881e-02
[15,30,60]	4.146672e-02	1.619794e-04	8.936911e-02
[10,20,40]	4.263060e-02	1.665258e-04	8.955818e-02
[5,10,20]	4.994913e-02	1.951138e-04	9.007786e-02
[4,8,16]	1.022157e-01	3.992802e-04	9.532119e-02
[3,6,12]	1.362098e-01	5.320695e-04	1.073860e-01
[2,4,8]	7.061800e-02	2.758515e-04	9.126418e-02
[1,2,4] && seed = 1	9.843161e-02	3.844985e-04	9.348505e-02
[1,2,4] && seed = 2	9.404335e-02	3.673568e-04	9.304089e-02
[1,2,4] && seed = 3	1.508016e-01	5.890688e-04	1.382356e-01
[1,2,4] && seed = 4	9.686949e-02	3.783965e-04	9.294820e-02

### 12.1.3 Water

Fitting net size tuning form on water: (embedding-net size: [25,50,100])

Fitting-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	9.1589E-04	5.1540E-02
[200,200,200]	9.3221E-04	5.2366E-02
[160,160,160]	9.4274E-04	5.3403E-02
[120,120,120]	9.5407E-04	5.3093E-02
[80,80,80]	9.4605E-04	5.3402E-02
[40,40,40]	9.8533E-04	5.5790E-02
[20,20,20]	1.0057E-03	5.8232E-02
[10,10,10]	1.0466E-03	6.2279E-02
[5,5,5]	1.1154E-03	6.7994E-02
[4,4,4]	1.1289E-03	6.9613E-02
[3,3,3]	1.2368E-03	7.9786E-02
[2,2,2]	1.3558E-03	9.7042E-02
[1,1,1]	1.4633E-03	1.1265E-01
[]	1.5193E-03	1.2136E-01

Embedding net size tuning form on water: (Fitting-net size: [240,240,240])

Embedding-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	9.1589E-04	5.1540E-02
[20,40,80]	9.5080E-04	5.3593E-02
[15,30,60]	9.7996E-04	5.6338E-02
[10,20,40]	1.0353E-03	6.2776E-02
[5,10,20]	1.1254E-03	7.3195E-02
[4,8,16]	1.2495E-03	8.0371E-02
[3,6,12]	1.3604E-03	9.9883E-02
[2,4,8]	1.4358E-03	9.7389E-02
[1,2,4]	2.1765E-03	1.7276E-01

## 12.1.4 Mg-Al

Fitting net size tuning form on Mg-Al: (embedding-net size: [25,50,100])

Fitting-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[240,240,240]	3.9606e-03	1.6289e-02
[200,200,200]	3.9449e-03	1.6471e-02
[160,160,160]	4.0947e-03	1.6413e-02
[120,120,120]	3.9234e-03	1.6283e-02
[80,80,80]	3.9758e-03	1.6506e-02
[40,40,40]	3.9142e-03	1.6348e-02
[20,20,20]	4.1302e-03	1.7006e-02
[10,10,10]	4.3433e-03	1.7524e-02
[5,5,5]	5.3154e-03	1.9716e-02
[4,4,4]	5.4210e-03	1.9710e-02
[2,2,2]	6.2667e-03	2.2568e-02
[1,1,1]	7.3676e-03	2.6375e-02
[]	7.3999e-03	2.6097e-02

Embedding net size tuning form on Mg-Al: (Fitting-net size: [240,240,240])

Embedding-net size	Energy L2err/Natoms(eV)	Force L2err(eV/Angstrom)
[25,50,100]	3.9606e-03	1.6289e-02
[20,40,80]	4.0292e-03	1.6555e-02
[15,30,60]	4.1743e-03	1.7026e-02
[10,20,40]	4.8138e-03	1.8516e-02
[5,10,20]	5.6052e-03	2.0709e-02
[4,8,16]	6.1335e-03	2.1450e-02
[3,6,12]	6.6469e-03	2.3003e-02
[2,4,8]	6.8222e-03	2.6318e-02
[1,2,4]	1.0678e-02	3.9559e-02

## 12.2 How to control the parallelism of a job?

DeePMD-kit has three levels of parallelism. To get the best performance, one should control the number of threads used by DeePMD-kit. One should make sure the product of the parallel numbers is less than or equal to the number of cores available.

### 12.2.1 MPI (optional)

Parallelism for MPI is optional and used for multiple nodes, multiple GPU cards, or sometimes multiple CPU cores.

To enable MPI support for training, one should [install horovod](#) in advance. Note that the parallelism mode is data parallelism, so it is not expected to see the training time per batch decreases.

MPI support for inference is not directly supported by DeePMD-kit, but indirectly supported by the third-party software. For example, [LAMMPS enables running simulations in parallel](#) using the MPI parallel communication standard with distributed data. That software has to build against MPI.

Set the number of processes with:

```
mpirun -np $num_nodes dp
```

Note that `mpirun` here should be the same as the MPI used to build software. For example, one can use `mpirun --version` and `lmp -h` to see if `mpirun` and LAMMPS has the same MPI version.

Sometimes, `$num_nodes` and the nodes information can be directly given by the HPC scheduler system, if the MPI used here is the same as the MPI used to build the scheduler system. Otherwise, one have to manually assign these information.

### 12.2.2 Parallelism between independent operators

For CPU devices, TensorFlow use multiple streams to run independent operators (OP).

```
export TF_INTER_OP_PARALLELISM_THREADS=3
```

However, for GPU devices, TensorFlow uses only one compute stream and multiple copy streams. Note that some of DeePMD-kit OPs do not have GPU support, so it is still encouraged to set environmental variables even if one has a GPU.

### 12.2.3 Parallelism within an individual operators

For CPU devices, `TF_INTRA_OP_PARALLELISM_THREADS` controls parallelism within TensorFlow native OPs when TensorFlow is built against Eigen.

```
export TF_INTRA_OP_PARALLELISM_THREADS=2
```

`OMP_NUM_THREADS` is threads for OpenMP parallelism. It controls parallelism within TensorFlow native OPs when TensorFlow is built by Intel OneDNN and DeePMD-kit custom CPU OPs. It may also control parallelism for NumPy when NumPy is built against OpenMP, so one who uses GPUs for training should also care this environmental variable.

```
export OMP_NUM_THREADS=2
```

There are several other environmental variables for OpenMP, such as `KMP_BLOCKTIME`. See [Intel documentation](#) for detailed information.

### 12.2.4 Tune the performance

There is no one general parallel configuration that works for all situations, so you are encouraged to tune parallel configurations yourself after empirical testing.

Here are some empirical examples. If you wish to use 3 cores of 2 CPUs on one node, you may set the environmental variables and run DeePMD-kit as follows:

```
export OMP_NUM_THREADS=3
export TF_INTRA_OP_PARALLELISM_THREADS=3
export TF_INTER_OP_PARALLELISM_THREADS=2
dp train input.json
```

For a node with 128 cores, it is recommended to start with the following variables:

```
export OMP_NUM_THREADS=16
export TF_INTRA_OP_PARALLELISM_THREADS=16
export TF_INTER_OP_PARALLELISM_THREADS=8
```

Again, in general, one should make sure the product of the parallel numbers is less than or equal to the number of cores available. In the above case,  $16 \times 8 = 128$ , so threads will not compete with each other.

## 12.3 Do we need to set $rcut < \text{half boxsize}$ ?

When seeking the neighbors of atom  $i$  under periodic boundary conditions, DeePMD-kit considers all  $j$  atoms within cutoff  $rcut$  from atom  $i$  in all mirror cells.

So, there is no limitation on the setting of  $rcut$ .

PS: The reason why some software requires  $rcut < \text{half box size}$  is that they only consider the nearest mirrors from the center cell. DeePMD-kit is different from them.

## 12.4 How to set $sel$ ?

$sel$  is short for “selected number of atoms in  $rcut$ ”.

$sel\_a[i]$  is a list of integers. The length of the list should be the same as the number of atom types in the system.

$sel\_a[i]$  gives the number of the selected number of type  $i$  neighbors within  $rcut$ . To ensure that the results are strictly accurate,  $sel\_a[i]$  should be larger than the largest number of type  $i$  neighbors in the  $rcut$ .

However, the computation overhead increases with  $sel\_a[i]$ , therefore,  $sel\_a[i]$  should be as small as possible.

The setting of  $sel\_a[i]$  should balance the above two considerations.



## 12.5 Installation

### 12.5.1 Inadequate versions of gcc/g++

Sometimes you may use a gcc/g++ of version < 4.8. In this way, you can still compile all the parts of TensorFlow and most of the parts of DeePMD-kit, but i-Pi and GROMACS plugins will be disabled automatically. Or if you have a gcc/g++ of version > 4.8, say, 7.2.0, you may choose to use it by doing

```
export CC=/path/to/gcc-7.2.0/bin/gcc
export CXX=/path/to/gcc-7.2.0/bin/g++
```

### 12.5.2 Build files left in DeePMD-kit

When you try to build a second time when installing DeePMD-kit, files produced before may contribute to failure. Thus, you may clear them by

```
cd build
rm -r *
```

and redo the cmake process.

## 12.6 The temperature undulates violently during the early stages of MD

This is probably because your structure is too far from the equilibrium configuration.

To make sure the potential model is truly accurate, we recommend checking model deviation.

## 12.7 MD: cannot run LAMMPS after installing a new version of DeePMD-kit

This typically happens when you install a new version of DeePMD-kit and copy directly the generated USER-DEEPM to a LAMMPS source code folder and re-install LAMMPS.

To solve this problem, it suffices to first remove USER-DEEPM from the LAMMPS source code by

```
make no-user-deepmd
```

and then install the new USER-DEEPM.

If this does not solve your problem, try to decompress the LAMMPS source tarball and install LAMMPS from scratch again, which typically should be very fast.

## 12.8 Model compatibility

When the version of DeePMD-kit used to train the model is different from the that of DeePMD-kit running MDs, one has the problem of model compatibility.

DeePMD-kit guarantees that the codes with the same major and minor revisions are compatible. That is to say, v0.12.5 is compatible with v0.12.0, but is not compatible with v0.11.0 or v1.0.0.

One can execute `dp convert-from` to convert an old model to a new one.

Model version	v0.12	v1.0	v1.1	v1.2	v1.3	v2.0	v2.1	v2.2
Compatibility	😊	😊	😊	😊	😊	😊	😊	😊

Legend:

- 😊: The model is compatible with the DeePMD-kit package.
- 😊: The model is incompatible with the DeePMD-kit package, but one can execute `dp convert-from` to convert an old model to v2.2.
- ☹️: The model is incompatible with the DeePMD-kit package, and there is no way to convert models.

## 12.9 Why does a model have low precision?

Many phenomena are caused by model accuracy. For example, during simulations, temperatures explode, structures fall apart, and atoms are lost. One can [test the model](#) to confirm whether the model has the enough accuracy.

There are many reasons for a low-quality model. Some common reasons are listed below.

### 12.9.1 Data

#### Data units and signs

The unit of training data should follow what is listed in [data section](#). Usually, the package to calculate the training data has different units from those of the DeePMD-kit. It is noted that some software label the energy gradient as forces, instead of the negative energy gradient. It is necessary to check them carefully to avoid inconsistent data.

#### SCF coverage and data accuracy

The accuracy of models will not exceed the accuracy of training data, so the training data should reach enough accuracy. Here is a checklist for the accuracy of data:

- SCF should converge to a suitable threshold for all points in the training data.
- The convergence of the energy, force and virial with respect to the energy cutoff and k-spacing sample is checked.
- Sometimes, QM software may generate unstable outliers, which should be removed.
- The data should be extracted with enough digits and stored with the proper precision. Large energies may have low precision when they are stored as the single-precision floating-point format (FP32).

### Enough data

If the model performs good on the training data, but has bad accuracy on another data, this means some data space is not covered by the training data. It can be validated by evaluating the [model deviation](#) with multiple models. If the model deviation of these data is high for some data, try to collect more data using [DP-GEN](#).

### Values of data

One should be aware that the errors of some data is also affected by the absolute values of this data. Stable structures tend to be more precise than unstable structures because unstable structures may have larger forces. Also, errors will be introduced in the Projector augmented wave (PAW) DFT calculations when the atoms are very close due to the overlap of pseudo-potentials. It is expected to see that data with large forces has larger errors and it is better to compare different models only with the same data.

## 12.9.2 Model

### Enough sel

The `sel` of the descriptors must be enough for both training and test data. Otherwise, the model will be unreliable and give wrong results.

### Cutoff radius

The model cannot fit the long-term interaction out of the cutoff radius. This is a designed approximation for performance, but one has to choose proper cutoff radius for the system.

### Neural network size

The size of neural networks will affect the accuracy, but if one follows the parameters in the examples, this effect is insignificant. See [FAQ: How to tune Fitting/embedding-net size](#) for details.

### Neural network precision

In some cases, one may want to use the FP32 precision to make the model faster. For some applications, FP32 is enough and thus is recommended, but one should still be aware that the precision of FP32 is not as high as that of FP64.

## 12.9.3 Training

### Training steps

Generally speaking, the longer the number of training steps, the better the model. A balance between model accuracy and training time can be achieved. If one finds that model accuracy decreases with training time, there may be a problem with the data. See the [data section](#) for details.

**Learning rate**

Both too large and too small learning rate may affect the training. It is recommended to start with a large learning rate and end with a small learning rate. The learning rate from the examples is a good choice to start.

## FIND DEEPMD-KIT C/C++ LIBRARY FROM CMAKE

After DeePMD-kit C/C++ library is installed, one can find DeePMD-kit from CMake:

```
find_package(DeePMD REQUIRED)
```

Note that you may need to add `${deepmd_root}` to the cached CMake variable `CMAKE_PREFIX_PATH`.

To link against the C interface library, using

```
target_link_libraries(some_library PRIVATE DeePMD::deepmd_c)
```

To link against the C++ interface library, using

```
target_link_libraries(some_library PRIVATE DeePMD::deepmd_cc)
```



## CREATE A MODEL

If you'd like to create a new model that isn't covered by the existing DeePMD-kit library, but reuse DeePMD-kit's other efficient modules such as data processing, trainer, etc, you may want to read this section.

To incorporate your custom model you'll need to:

1. Register and implement new components (e.g. descriptor) in a Python file. You may also want to register new TensorFlow OPs if necessary.
2. Register new arguments for user inputs.
3. Package new codes into a Python package.
4. Test new models.

### 14.1 Design a new component

When creating a new component, take descriptor as the example, you should inherit `deepmd.descriptor.descriptor.Descriptor` class and override several methods. Abstract methods such as `deepmd.descriptor.descriptor.Descriptor.build` must be implemented and others are not. You should keep arguments of these methods unchanged.

After implementation, you need to register the component with a key:

```
from deepmd.descriptor import Descriptor

@Descriptor.register("some_descrpt")
class SomeDescriptor(Descriptor):
    def __init__(self, arg1: bool, arg2: float) -> None:
        pass
```

### 14.2 Register new arguments

To let someone uses your new component in their input file, you need to create a new method that returns some `Argument` of your new component, and then register new arguments. For example, the code below

```
from typing import List

from dargs import Argument
from deepmd.utils.argcheck import descrpt_args_plugin
```

(continues on next page)

(continued from previous page)

```
@descrpt_args_plugin.register("some_descrpt")
def descrpt_some_args() -> List[Argument]:
    return [
        Argument("arg1", bool, optional=False, doc="balabala"),
        Argument("arg2", float, optional=True, default=6.0, doc="haha"),
    ]
```

allows one to use your new descriptor as below:

```
"descriptor" :{
    "type": "some_descrpt",
    "arg1": true,
    "arg2": 6.0
}
```

The arguments here should be consistent with the class arguments of your new component.

## 14.3 Package new codes

You may use `setuptools` to package new codes into a new Python package. It's crucial to add your new component to `entry_points['deepmd']` in `setup.py`:

```
entry_points = (
    {
        "deepmd": [
            "some_descrpt=deepmd_some_descrpt:SomeDescribept",
        ],
    },
)
```

where `deepmd_some_descrpt` is the module of your codes. It is equivalent to `from deepmd_some_descrpt import SomeDescribept`.

If you place `SomeDescribept` and `descrpt_some_args` into different modules, you are also expected to add `descrpt_some_args` to `entry_points`.

After you install your new package, you can now use `dp train` to run your new model.



## ATOM TYPE EMBEDDING

### 15.1 Overview

Here is an overview of the DeePMD-kit algorithm. Given a specific centric atom, we can obtain the matrix describing its local environment, named  $\mathcal{R}$ . It consists of the distance between the centric atom and its neighbors, as well as a direction vector. We can embed each distance into a vector of  $M_1$  dimension by an **embedding net**, so the environment matrix  $\mathcal{R}$  can be embedded into matrix  $\mathcal{G}$ . We can thus extract a descriptor vector (of  $M_1 \times M_2$  dim) of the centric atom from the  $\mathcal{G}$  by some matrix multiplication, and put the descriptor into **fitting net** to get the predicted energy  $E$ . The vanilla version of DeePMD-kit builds **embedding net** and **fitting net** relying on the atom type, resulting in  $O(N)$  memory usage. After applying atom type embedding, in DeePMD-kit v2.0, we can share one **embedding net** and one **fitting net** in total, which reduces training complexity largely.

### 15.2 Preliminary

In the following chart, you can find the meaning of symbols used to clarify the atom-type embedding algorithm.

$i$ : Type of centric atom

$j$ : Type of neighbor atom

$s_{ij}$ : Distance between centric atom and neighbor atom

$\mathcal{G}_{ij}(\cdot)$ : Origin embedding net, take  $s_{ij}$  as input and output embedding vector of  $M_1$  dim

$\mathcal{G}(\cdot)$ : Shared embedding net

$\text{Multi}(\cdot)$ : Matrix multiplication and flattening, output the descriptor vector of  $M_1 \times M_2$  dim

$F_i(\cdot)$ : Origin fitting net, take the descriptor vector as input and output energy

$F(\cdot)$ : Shared fitting net

$A(\cdot)$ : Atom type embedding net, input is atom type, the output is type embedding vector of dim `nchan1`

So, we can formulate the training process as follows. Vanilla DeePMD-kit algorithm:

$$E = F_i(\text{Multi}(\mathcal{G}_{ij}(s_{ij})))$$

DeePMD-kit applying atom type embedding:

$$E = F([\text{Multi}(\mathcal{G}([s_{ij}, A(i), A(j)])), A(j)])$$

or

$$E = F(\text{Multi}(\mathcal{G}([s_{ij}, A(j)])), A(j))$$

The difference between the two variants above is whether using the information of centric atom when generating the descriptor. Users can choose by modifying the `type_one_side` hyper-parameter in the input JSON file.

## 15.3 How to use

A detailed introduction can be found at [se\\_e2\\_a\\_tebd](#). Looking for a fast start-up, you can simply add a `type_embedding` section in the input JSON file as displayed in the following, and the algorithm will adopt the atom type embedding algorithm automatically. An example of `type_embedding` is like

```
"type_embedding":{
  "neuron":      [2, 4, 8],
  "resnet_dt":   false,
  "seed":        1
}
```

## 15.4 Code Modification

Atom-type embedding can be applied to varied `embedding net` and `fitting net`, as a result, we build a class `TypeEmbedNet` to support this free combination. In the following, we will go through the execution process of the code to explain our code modification.

### 15.4.1 trainer (train/trainer.py)

In `trainer.py`, it will parse the parameter from the input JSON file. If a `type_embedding` section is detected, it will build a `TypeEmbedNet`, which will be later input in the `model`. `model` will be built in the function `_build_network`.

### 15.4.2 model (model/ener.py)

When building the operation graph of the `model` in `model.build`. If a `TypeEmbedNet` is detected, it will build the operation graph of `type embed net`, `embedding net` and `fitting net` by order. The building process of `type embed net` can be found in `TypeEmbedNet.build`, which output the type embedding vector of each atom type (of `[ntypes × nchan]` dimensions). We then save the type embedding vector into `input_dict`, so that they can be fetched later in `embedding net` and `fitting net`.

### 15.4.3 embedding net (descriptor/se\*.py)

In `embedding net`, we shall take local environment  $\mathcal{R}$  as input and output matrix  $\mathcal{G}$ . Functions called in this process by the order is

```
build -> _pass_filter -> _filter -> _filter_lower
```

`_pass_filter`: It will first detect whether an atom type embedding exists, if so, it will apply atom type embedding algorithm and doesn't divide the input by type.

`_filter`: It will call `_filter_lower` function to obtain the result of matrix multiplication ( $\mathcal{G}^T \cdot \mathcal{R}$ ), do further multiplication involved in `Multi()`, and finally output the result of descriptor vector of  $M_1 \times M_2$  dim.

`_filter_lower`: The main function handling input modification. If type embedding exists, it will call `_concat_type_embedding` function to concat the first column of input  $\mathcal{R}$  (the column of  $s_{ij}$ ) with the atom type embedding information. It will decide whether to use the atom type embedding vector of the centric atom according to the value of `type_one_side` (if set True, then we only use the vector of the neighbor atom). The modified input will be put into the `fitting net` to get  $\mathcal{G}$  for further matrix multiplication stage.

### 15.4.4 fitting net (fit/ener.py)

In `fitting net`, it takes the descriptor vector as input, whose dimension is  $[\text{natoms}, M_1 \times M_2]$ . Because we need to involve information on the centric atom in this step, we need to generate a matrix named `atype_embed` (of dim  $[\text{natoms}, \text{nchan}]$ ), in which each row is the type embedding vector of the specific centric atom. The input is sorted by type of centric atom, we also know the number of a particular atom type (stored in `natoms[2+i]`), thus we get the type vector of the centric atom. In the build phase of the fitting net, it will check whether type embedding exists in `input_dict` and fetch them. After that, call `embed_atom_type` function to look up the embedding vector for the type vector of the centric atom to obtain `atype_embed`, and concat input with it (`[input, atype_embed]`). The modified input goes through `fitting net` to get predicted energy.

---

Note: You can't apply the compression method while using atom-type embedding.

---



## CODING CONVENTIONS

### 16.1 Preface

The aim of these coding standards is to help create a codebase with a defined and consistent coding style that every contributor can get easily familiar with. This will enhance code readability as there will be no different coding styles from different contributors and everything will be documented. Also, PR diffs will be smaller because of the unified coding style. Finally, static typing will help in hunting down potential bugs before the code is even run.

Contributed code will not be refused merely because it does not strictly adhere to these conditions; as long as it's internally consistent, clean, and correct, it probably will be accepted. But don't be surprised if the "offending" code gets fiddled with overtime to conform to these conventions.

There are also pre-commit CI checks for python code style which will automatically fix the PR.

### 16.2 Python

#### 16.2.1 Rules

The code must be compatible with the oldest supported version of python which is 3.7

The project follows the generic coding conventions as specified in the [Style Guide for Python Code](#), [Docstring Conventions](#) and [Typing Conventions](#) PEPs, clarified and extended as follows:

- Do not use "\*" imports such as `from module import *`. Instead, list imports explicitly.
- Use 4 spaces per indentation level. No tabs.
- No one-liner compound statements (i.e., no `if x: return:` use two lines).
- Maximum line length is 88 characters as recommended by [black](#) which is less strict than [Docstring Conventions](#) suggests.
- Use "StudlyCaps" for class names.
- Use "lowercase" or "lowercase\_with\_underscores" for function, method, variable names and module names. For short names, joined lowercase may be used (e.g. "tagname"). Choose what is most readable.
- No single-character variable names, except indices in loops that encompass a very small number of lines (`for i in range(5): ...`).
- Avoid lambda expressions. Use named functions instead.
- Avoid functional constructs (filter, map, etc.). Use list comprehensions instead.

- Use "double quotes" for string literals, and `"""triple double quotes"""` for docstring's. Single quotes are OK for something like

```
f"something {'this' if x else 'that'}"
```

- Use f-strings `s = f"{x:.2f}"` instead of old style formatting with `"%f" % x`. string format method `"{x:.2f}".format()` may be used sparsely where it is more convenient than f-strings.

### 16.2.2 Whitespace

Python is not C/C++ so whitespace should be used sparingly to maintain code readability

- Read the Whitespace in Expressions and Statements section of [PEP8](#).
- Avoid [trailing whitespaces](#).
- Do not use excessive whitespace in your expressions and statements.
- You should have blank spaces after commas, colons, and semi-colons if it isn't trailing next to the end of a bracket, brace, or parentheses.
- With any operators you should use space on both sides of the operator.
- Colons for slicing are considered a binary operator, and should not have any spaces between them.
- You should have parentheses with no space, directly next to the function when calling functions `function()`.
- When indexing or slicing the brackets should be directly next to the collection with no space `collection["index"]`.
- Whitespace used to line up variable values is not recommended.
- Make sure you are consistent with the formats you choose when optional choices are available.

### 16.2.3 General advice

- Get rid of as many `break` and `continue` statements as possible.
- Write short functions. All functions should fit within a standard screen.
- Use descriptive variable names.

### 16.2.4 Writing documentation in the code

Here is an example of how to write good docstrings:

<https://github.com/numpy/numpy/blob/master/doc/example.py>

The NumPy docstring documentation can be found [here](#)

## 16.3 C++

The customized Clang Format style is used for C++ code formatting. The style is defined in `.clang-format` file in the root of the repository. The style is based on the Google C++ style with some modifications.

## 16.4 Run scripts to check the code

It's a good idea to install `pre-commit` on your repository:

```
$ pip install pre-commit
$ pre-commit install
```

The scripts will be run automatically before each commit and will fix the code style issues automatically.





## 17.1 CI

### 17.1.1 Test CUDA

**Test CUDA** action runs tests on a self-hosted runner with the NVIDIA card. It is not triggered by every PR. The developer who has the permission to manage the label can apply the label **Test CUDA** to a PR to trigger this action.

## 17.2 CD

GitHub Actions is used to build pre-compiled packages for each commit. See the [Easy install the latest development version](#) section to learn how to install the latest development version.



## 18.1 backend package

### 18.1.1 Submodules

### 18.1.2 backend.dp\_backend module

### 18.1.3 backend.dynamic\_metadata module

`backend.dynamic_metadata.dynamic_metadata(field: str, settings: Optional[Dict[str, object]] = None) → str`

### 18.1.4 backend.find\_tensorflow module

`backend.find_tensorflow.find_tensorflow() → Tuple[Optional[str], List[str]]`

Find TensorFlow library.

Tries to find TensorFlow in the order of:

1. Environment variable `TENSORFLOW_ROOT` if set
2. The current Python environment.
3. user site packages directory if enabled
4. system site packages directory (purelib)
5. add as a requirement (detect `TENSORFLOW_VERSION` or the latest) and let pip install it

Returns

`str`

TensorFlow library path if found.

`list of str`

TensorFlow requirement if not found. Empty if found.

`backend.find_tensorflow.get_tf_requirement(tf_version: str = '') → dict`

Get TensorFlow requirement (CPU) when TF is not installed.

If `tf_version` is not given and the environment variable `TENSORFLOW_VERSION` is set, use it as the requirement.

Parameters

`tf_version`  
[`str`, optional] TF version

Returns

`dict`  
TensorFlow requirement, including cpu and gpu.

`backend.find_tensorflow.get_tf_version(tf_path: Union[str, Path]) → str`

Get TF version from a TF Python library path.

Parameters

`tf_path`  
[`str` or `Path`] TF Python library path

Returns

`str`  
version

### 18.1.5 backend.read\_env module

Read environment variables to configure the build.

`backend.read_env.get_argument_from_env() → Tuple[str, list, list, dict, str]`

Get the arguments from environment variables.

The environment variables are assumed to be not changed during the build.

Returns

`str`  
The minimum required CMake version.

`list of str`  
The CMake arguments.

`list of str`  
The requirements for the build.

`dict`  
The extra scripts to be installed.

`str`  
The TensorFlow version.

`backend.read_env.set_scikit_build_env()`

Set scikit-build environment variables before executing scikit-build.

## 18.2 deepmd package

Root of the deepmd package, exposes all public classes and submodules.

```
class deepmd.DeepEval(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False,
                      auto_batch_size: Union[bool, int, AutoBatchSize] = False, input_map:
                      Optional[dict] = None, neighbor_list=None)
```

Bases: `object`

Common methods for DeepPot, DeepWFC, DeepPolar, ...

Parameters

`model_file`  
[Path] The name of the frozen model file.

`load_prefix`: str  
The prefix in the load computational graph

`default_tf_graph`  
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

`auto_batch_size`  
[bool or int or AutomaticBatchSize, default: False] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

`input_map`  
[dict, optional] The input map for `tf.import_graph_def`. Only work with default tf graph

`neighbor_list`  
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

Attributes

`model_type`  
Get type of model.

`model_version`  
Get version of model.

`sess`  
Get TF session.

### Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval_typeembed()</code>	Evaluate output of type embedding network by using this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

**build\_neighbor\_list**(coords: `ndarray`, cell: `Optional[ndarray]`, atype: `ndarray`, imap: `ndarray`, neighbor\_list)

Make the mesh with neighbor list for a single frame.

Parameters

coords

[`np.ndarray`] The coordinates of atoms. Should be of shape [natoms, 3]

cell

[`Optional[np.ndarray]`] The cell of the system. Should be of shape [3, 3]

atype

[`np.ndarray`] The type of atoms. Should be of shape [natoms]

imap

[`np.ndarray`] The index map of atoms. Should be of shape [natoms]

neighbor\_list

[`ase.neighborlist.NewPrimitiveNeighborList`] ASE neighbor list. The following method or attribute will be used/set: bothways, self\_interaction, update, build, first\_neigh, pair\_second, offset\_vec.

Returns

natoms\_vec

[`np.ndarray`] The number of atoms. This tensor has the length of Ntypes + 2  
natoms[0]: nloc  
natoms[1]: nall  
natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms for nloc

coords

[`np.ndarray`] The coordinates of atoms, including ghost atoms. Should be of shape [nframes, nall, 3]

atype

[`np.ndarray`] The type of atoms, including ghost atoms. Should be of shape [nall]

mesh

[`np.ndarray`] The mesh in nei\_mode=4.

imap

[`np.ndarray`] The index map of atoms. Should be of shape [nall]

ghost\_map

[`np.ndarray`] The index map of ghost atoms. Should be of shape [nghost]

**eval\_typeebd**() → `ndarray`

Evaluate output of type embedding network by using this model.

Returns

`np.ndarray`

The output of type embedding network. The shape is [ntypes, o\_size], where ntypes is the number of types, and o\_size is the number of nodes in the output layer.

Raises

`KeyError`

If the model does not enable type embedding.

See also:

`deepmd.utils.type_embed.TypeEmbedNet`  
The type embedding network.

## Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential('graph.pb')
>>> dp.eval_typeembed()
```

**load\_prefix:** `str`

**make\_natoms\_vec**(atom\_types: `ndarray`, mixed\_type: `bool` = False) → `ndarray`

Make the natom vector used by deepmd-kit.

Parameters

atom\_types

The type of atoms

mixed\_type

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

**natoms**

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < Ntypes+2$ , number of type i atoms

**property model\_type:** `str`

Get type of model.

:type:str

**property model\_version:** `str`

Get version of model.

Returns

`str`

version of model

**static reverse\_map**(vec: `ndarray`, imap: `List[int]`) → `ndarray`

Reverse mapping of a vector according to the index map.

Parameters

vec

Input vector. Be of shape [nframes, natoms, -1]

imap

Index map. Be of shape [natoms]

Returns

**vec\_out**

Reverse mapped vector.

**property sess:** `Session`

Get TF session.

**static sort\_input**(coord: `ndarray`, atom\_type: `ndarray`, sel\_atoms: `Optional[List[int]]` = None, mixed\_type: `bool` = False)

Sort atoms in the system according to their types.

Parameters

coord

The coordinates of atoms. Should be of shape [nframes, natoms, 3]

atom\_type

The type of atoms Should be of shape [natoms]

sel\_atoms

The selected atoms by type

mixed\_type

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

**coord\_out**

The coordinates after sorting

**atom\_type\_out**

The atom types after sorting

**idx\_map**

The index mapping from the input to the output. For example coord\_out = coord[:,idx\_map,:]

**sel\_atom\_type**

Only output if sel\_atoms is not None The sorted selected atom types

**sel\_idx\_map**

Only output if sel\_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

**deepmd.DeepPotential**(model\_file: `Union[str, Path]`, load\_prefix: `str` = 'load', default\_tf\_graph: `bool` = False, input\_map: `Optional[dict]` = None, neighbor\_list=None) → `Union[DeepDipole, DeepGlobalPolar, DeepPolar, DeepPot, DeepDOS, DeepWFC]`

Factory function that will initialize appropriate potential read from model\_file.

Parameters

model\_file

[`str`] The name of the frozen model file.

load\_prefix

[`str`] The prefix in the load computational graph

default\_tf\_graph

[`bool`] If uses the default tf graph, otherwise build a new tf graph for evaluation

input\_map

[`dict`, `optional`] The input map for tf.import\_graph\_def. Only work with default tf graph



`neighbor_list`  
`[ase.neighborlist.NeighborList, optional]` The neighbor list object. If None, then build the native neighbor list.

Returns

`Union[DeepDipole, DeepGlobalPolar, DeepPolar, DeepPot, DeepWFC]`  
 one of the available potentials

Raises

`RuntimeError`  
 if model file does not correspond to any implemented potential

```
class deepmd.DipoleChargeModifier(model_name: str, model_charge_map: List[float],
                                   sys_charge_map: List[float], ewald_h: float = 1, ewald_beta: float
                                   = 1)
```

Bases: *DeepDipole*

Parameters

`model_name`  
 The model file for the DeepDipole model

`model_charge_map`  
 Gives the amount of charge for the wfcc

`sys_charge_map`  
 Gives the amount of charge for the real atoms

`ewald_h`  
 Grid spacing of the reciprocal part of Ewald sum. Unit: Å

`ewald_beta`  
 Splitting parameter of the Ewald sum. Unit: Å<sup>-1</sup>

Attributes

`model_type`  
 Get type of model.

`model_version`  
 Get version of model.

`sess`  
 Get TF session.

## Methods

<i>build_fv_graph()</i>	Build the computational graph for the force and virial inference.
<i>build_neighbor_list</i> (coords, cell, atype, ...)	Make the mesh with neighbor list for a single frame.
<i>eval</i> (coord, box, atype[, eval_fv])	Evaluate the modification.
<i>eval_full</i> (coords, cells, atom_types[, ...])	Evaluate the model with interface similar to the energy model.
<i>eval_typeebd</i> ()	Evaluate output of type embedding network by using this model.
<i>get_dim_aparam</i> ()	Unsupported in this model.
<i>get_dim_fparam</i> ()	Unsupported in this model.
<i>get_ntypes</i> ()	Get the number of atom types of this model.
<i>get_rcut</i> ()	Get the cut-off radius of this model.
<i>get_sel_type</i> ()	Get the selected atom types of this model.
<i>get_type_map</i> ()	Get the type map (element name of the atom types) of this model.
<i>make_natoms_vec</i> (atom_types[, mixed_type])	Make the natom vector used by deepmd-kit.
<i>modify_data</i> (data, data_sys)	Modify data.
<i>reverse_map</i> (vec, imap)	Reverse mapping of a vector according to the index map.
<i>sort_input</i> (coord, atom_type[, sel_atoms, ...])	Sort atoms in the system according their types.

**build\_fv\_graph()** → Tensor

Build the computational graph for the force and virial inference.

**eval**(coord: ndarray, box: ndarray, atype: ndarray, eval\_fv: bool = True) → Tuple[ndarray, ndarray, ndarray]

Evaluate the modification.

Parameters

**coord**  
The coordinates of atoms

**box**  
The simulation region. PBC is assumed

**atype**  
The atom types

**eval\_fv**  
Evaluate force and virial

Returns

**tot\_e**  
The energy modification

**tot\_f**  
The force modification

**tot\_v**  
The virial modification

`modify_data(data: dict, data_sys: DeepmdData) → None`

Modify data.

Parameters

`data`

Internal data of DeepmdData. Be a dict, has the following keys - coord coordinates - box simulation box - type atom types - find\_energy tells if data has energy - find\_force tells if data has force - find\_virial tells if data has virial - energy energy - force force - virial virial

`data_sys`

[DeepmdData] The data system.

## 18.2.1 Subpackages

### deepmd.cluster package

Module that reads node resources, auto detects if running local or on SLURM.

`deepmd.cluster.get_resource() → Tuple[str, List[str], Optional[List[int]]]`

Get local or slurm resources: nodename, nodelist, and gpus.

Returns

`Tuple[str, List[str], Optional[List[int]]]`  
nodename, nodelist, and gpus

### Submodules

#### deepmd.cluster.local module

Get local GPU resources.

`deepmd.cluster.local.get_gpus()`

Get available IDs of GPU cards at local. These IDs are valid when used as the TensorFlow device ID.

Returns

`Optional[List[int]]`  
List of available GPU IDs. Otherwise, None.

`deepmd.cluster.local.get_resource() → Tuple[str, List[str], Optional[List[int]]]`

Get local resources: nodename, nodelist, and gpus.

Returns

`Tuple[str, List[str], Optional[List[int]]]`  
nodename, nodelist, and gpus

## deepmd.cluster.slurm module

Module to get resources on SLURM cluster.

### References

[https://github.com/deepsense-ai/tensorflow\\_on\\_slurm](https://github.com/deepsense-ai/tensorflow_on_slurm) ####

`deepmd.cluster.slurm.get_resource()` → `Tuple[str, List[str], Optional[List[int]]]`

Get SLURM resources: nodename, nodelist, and gpus.

Returns

`Tuple[str, List[str], Optional[List[int]]]`  
nodename, nodelist, and gpus

Raises

`RuntimeError`  
if number of nodes could not be retrieved

`ValueError`  
list of nodes is not of the same length as a number of nodes

`ValueError`  
if current nodename is not found in node list

## deepmd.descriptor package

`class deepmd.descriptor.Descriptor(*args, **kwargs)`

Bases: *PluginVariant*

The abstract class for descriptors. All specific descriptors should be based on this class.

The descriptor  $\mathcal{D}$  describes the environment of an atom, which should be a function of coordinates and types of its neighbour atoms.

### Notes

Only methods and attributes defined in this class are generally public, that can be called by other classes.

### Examples

```
>>> descript = Descriptor(type="se_e2_a", rcut=6., rcut_smth=0.5, sel=[50])
>>> type(descript)
<class 'deepmd.descriptor.se_a.DescriptSeA'>
```

Attributes

`explicit_ntypes`  
Explicit ntypes with type embedding.

## Methods

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Receive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>pass_tensors_from_frz_model</i> (*tensors)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get\_class\_by\_input

**abstract** `build`(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: Dict[str, Any], reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

Parameters

`coord_`  
[tf.Tensor] The coordinate of atoms

`atype_`  
[tf.Tensor] The type of atoms

`natoms`  
[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2  
natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

`box_`  
[tf.Tensor] The box of frames

`mesh`  
[tf.Tensor] For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`  
[dict[str, Any]] Dictionary for additional inputs

reuse  
[`bool`, `optional`] The weights in the networks should be reused when get the variable.

suffix  
[`str`, `optional`] Name suffix to identify this descriptor

Returns

descriptor: `tf.Tensor`  
The output descriptor

## Notes

This method must be implemented, as it's called by other classes.

`build_type_exclude_mask(exclude_types: List[Tuple[int, int]], ntypes: int, sel: List[int], ndescript: int, atype: Tensor, shape0: Tensor) → Tensor`

Build the type exclude mask for the descriptor.

Parameters

exclude\_types  
[`List`[`Tuple`[`int`, `int`]] The list of excluded types, e.g. [(0, 1), (1, 0)] means the interaction between type 0 and type 1 is excluded.

ntypes  
[`int`] The number of types.

sel  
[`List`[`int`]] The list of the number of selected neighbors for each type.

ndescript  
[`int`] The number of descriptors for each atom.

atype  
[`tf.Tensor`] The type of atoms, with the size of shape0.

shape0  
[`tf.Tensor`] The shape of the first dimension of the inputs, which is equal to nsamples \* natoms.

Returns

`tf.Tensor`  
The type exclude mask, with the shape of (shape0, ndescript), and the precision of GLOBAL\_TF\_FLOAT\_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

## Notes

To exclude the interaction between two types, the derivative of energy with respect to distances (or angles) between two atoms should be zero[R08579741114c-1], i.e.

$$\forall i \in \text{type 1}, j \in \text{type 2}, \frac{\partial E}{\partial r_{ij}} = 0$$

When embedding networks between every two types are built, we can just remove that network. But when `type_one_side` is enabled, a network may be built for multiple pairs of types. In this case, we need to build a mask to exclude the interaction between two types.

The mask assumes the descriptors are sorted by neighbor type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

## References

[1]

```
abstract compute_input_stats(data_coord: List[ndarray], data_box: List[ndarray], data_atype:
                             List[ndarray], natoms_vec: List[ndarray], mesh: List[ndarray],
                             input_dict: Dict[str, List[ndarray]], **kwargs) → None
```

Compute the statistics (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
[list[np.ndarray]] The coordinates. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`data_box`  
[list[np.ndarray]] The box. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`data_atype`  
[list[np.ndarray]] The atom types. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`natoms_vec`  
[list[np.ndarray]] The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`mesh`  
[list[np.ndarray]] The mesh for neighbor searching. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`input_dict`  
[dict[str, list[np.ndarray]]] Dictionary for additional input

`**kwargs`  
Additional keyword arguments which may contain `mixed_type` and `real_natoms_vec`.

### Notes

This method must be implemented, as it's called by other classes.

```
enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate: float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1, check_frequency: int = -1, suffix: str = '') → None
```

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

#### Parameters

**min\_nbor\_dist**  
[float] The nearest distance between atoms

**graph**  
[tf.Graph] The graph of the model

**graph\_def**  
[tf.GraphDef] The graph definition of the model

**table\_extrapolate**  
[float, default: 5.] The scale of model extrapolation

**table\_stride\_1**  
[float, default: 0.01] The uniform stride of the first table

**table\_stride\_2**  
[float, default: 0.1] The uniform stride of the second table

**check\_frequency**  
[int, default: -1] The overflow check frequency

**suffix**  
[str, optional] The suffix of the scope

### Notes

This method is called by others when the descriptor supported compression.

```
enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
```

Reveive the mixed precision setting.

#### Parameters

**mixed\_prec**  
The mixed precision setting used in the embedding net

### Notes

This method is called by others when the descriptor supported compression.

```
property explicit_ntypes: bool
```

Explicit ntypes with type embedding.

```
classmethod get_class_by_input(input: dict)
```



**abstract** `get_dim_out()`  $\rightarrow$  `int`

Returns the output dimension of this descriptor.

Returns

`int`  
the output dimension of this descriptor

### Notes

This method must be implemented, as it's called by other classes.

**get\_dim\_rot\_mat\_1()**  $\rightarrow$  `int`

Returns the first dimension of the rotation matrix. The rotation is of shape `dim_1` x 3.

Returns

`int`  
the first dimension of the rotation matrix

**get\_nlist()**  $\rightarrow$  `Tuple[Tensor, Tensor, List[int], List[int]]`

Returns neighbor information.

Returns

`nlist`  
`[tf.Tensor]` Neighbor list

`rij`  
`[tf.Tensor]` The relative distance between the neighbor and the center atom.

`sel_a`  
`[list[int]]` The number of neighbors with full information

`sel_r`  
`[list[int]]` The number of neighbors with only radial information

**abstract** `get_ntypes()`  $\rightarrow$  `int`

Returns the number of atom types.

Returns

`int`  
the number of atom types

### Notes

This method must be implemented, as it's called by other classes.

**abstract** `get_rcut()`  $\rightarrow$  `float`

Returns the cut-off radius.

Returns

`float`  
the cut-off radius

### Notes

This method must be implemented, as it's called by other classes.

**get\_tensor\_names**(suffix: str = "") → Tuple[str]

Get names of tensors.

Parameters

suffix  
[str] The suffix of the scope

Returns

Tuple[str]  
Names of tensors

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: str = "") → None

Init the embedding net variables with the given dict.

Parameters

graph  
[tf.Graph] The input frozen model graph  
graph\_def  
[tf.GraphDef] The input frozen model graph\_def  
suffix  
[str, optional] The suffix of the scope

### Notes

This method is called by others when the descriptor supported initialization from the given variables.

**pass\_tensors\_from\_frz\_model**(\*tensors: Tensor) → None

Pass the descrpt\_reshape tensor as well as descrpt\_deriv tensor from the frz graph\_def.

Parameters

\*tensors  
[tf.Tensor] passed tensors

### Notes

The number of parameters in the method must be equal to the numbers of returns in *get\_tensor\_names()*.

**abstract prod\_force\_virial**(atom\_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]

Compute force and virial.

Parameters

atom\_ener  
[tf.Tensor] The atomic energy

`natoms`  
 [`tf.Tensor`] The number of atoms. This tensor has the length of `Ntypes + 2`  
`natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this  
 processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms

Returns

`force`  
 [`tf.Tensor`] The force on atoms  
  
`virial`  
 [`tf.Tensor`] The total virial  
  
`atom_virial`  
 [`tf.Tensor`] The atomic virial

**static register**(key: `str`) → `Callable`

Register a descriptor plugin.

Parameters

`key`  
 [`str`] the key of a descriptor

Returns

*Descriptor*  
 the registered descriptor

## Examples

```
>>> @Descriptor.register("some_descript")
class SomeDescript(Descriptor):
    pass
```

**abstract classmethod update\_sel**(global\_jdata: `dict`, local\_jdata: `dict`)

Update the selection and perform neighbor statistics.

Parameters

`global_jdata`  
 [`dict`] The global data, containing the training section

`local_jdata`  
 [`dict`] The local data refer to the current class

**class** `deepmd.descriptor.DescriptHybrid`(\*args, \*\*kwargs)

Bases: *Descriptor*

Concatenate a list of descriptors to form a new descriptor.

Parameters

`list`  
 [`list`] Build a descriptor from the concatenation of the list of descriptors.

Attributes

*explicit\_ntypes*  
 Explicit ntypes with type embedding.

## Methods

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.
<i>get_nlist_i</i> (ii)	Get the neighbor information of the ii-th descriptor.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (*tensors)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh

For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input\_dict

Dictionary for additional inputs

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

*descriptor*

The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, mixed\_type: bool = False, real\_natoms\_vec: Optional[list] = None, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data\_coord

The coordinates. Can be generated by deepmd.model.make\_stat\_input

data\_box

The box. Can be generated by deepmd.model.make\_stat\_input

data\_atype

The atom types. Can be generated by deepmd.model.make\_stat\_input

natoms\_vec

The vector for the number of atoms of the system and different types of atoms. Can be generated by deepmd.model.make\_stat\_input

mesh

The mesh for neighbor searching. Can be generated by deepmd.model.make\_stat\_input

input\_dict

Dictionary for additional input

mixed\_type

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

real\_natoms\_vec

If mixed\_type is True, it takes in the real natoms\_vec for each frame.

\*\*kwargs

Additional keyword arguments.

**enable\_compression**(min\_nbor\_dist: float, graph: Graph, graph\_def: GraphDef, table\_extrapolate: float = 5.0, table\_stride\_1: float = 0.01, table\_stride\_2: float = 0.1, check\_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

## Parameters

**min\_nbor\_dist**  
[float] The nearest distance between atoms

**graph**  
[tf.Graph] The graph of the model

**graph\_def**  
[tf.GraphDef] The graph\_def of the model

**table\_extrapolate**  
[float, default: 5.] The scale of model extrapolation

**table\_stride\_1**  
[float, default: 0.01] The uniform stride of the first table

**table\_stride\_2**  
[float, default: 0.1] The uniform stride of the second table

**check\_frequency**  
[int, default: -1] The overflow check frequency

**suffix**  
[str, optional] The suffix of the scope

**enable\_mixed\_precision**(mixed\_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

## Parameters

**mixed\_prec**  
The mixed precision setting used in the embedding net

**property explicit\_ntypes: bool**

Explicit ntypes with type embedding.

**get\_dim\_out()** → int

Returns the output dimension of this descriptor.

**get\_nlist()** → Tuple[Tensor, Tensor, List[int], List[int]]

Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.

## Returns

**nlist**  
Neighbor list

**rij**  
The relative distance between the neighbor and the center atom.

**sel\_a**  
The number of neighbors with full information

**sel\_r**  
The number of neighbors with only radial information

**get\_nlist\_i(ii: int)** → Tuple[Tensor, Tensor, List[int], List[int]]

Get the neighbor information of the ii-th descriptor.

## Parameters

ii  
     [`int`] The index of the descriptor

Returns

**nlist**  
     Neighbor list

**rij**  
     The relative distance between the neighbor and the center atom.

**sel\_a**  
     The number of neighbors with full information

**sel\_r**  
     The number of neighbors with only radial information

**get\_ntypes()** → `int`  
     Returns the number of atom types.

**get\_rcut()** → `float`  
     Returns the cut-off radius.

**get\_tensor\_names**(suffix: `str` = '') → `Tuple[str]`  
     Get names of tensors.

Parameters

    suffix  
         [`str`] The suffix of the scope

Returns

`Tuple[str]`  
         Names of tensors

**init\_variables**(graph: `Graph`, graph\_def: `GraphDef`, suffix: `str` = '') → `None`  
     Init the embedding net variables with the given dict.

Parameters

    graph  
         [`tf.Graph`] The input frozen model graph

    graph\_def  
         [`tf.GraphDef`] The input frozen model graph\_def

    suffix  
         [`str`, `optional`] The suffix of the scope

**merge\_input\_stats**(stat\_dict)  
     Merge the statisitcs computed from compute\_input\_stats to obtain the self.davg and self.dstd.

Parameters

    stat\_dict  
         The dict of statisitcs computed from compute\_input\_stats, including:

        sumr  
             The sum of radial statisitcs.

        suma  
             The sum of relative coord statisitcs.

**sumn**  
The sum of neighbor numbers.

**sumr2**  
The sum of square of radial statisits.

**suma2**  
The sum of square of relative coord statisits.

**pass\_tensors\_from\_frz\_model**(\*tensors: Tensor) → None

Pass the descrpt\_reshape tensor as well as descrpt\_deriv tensor from the frz graph\_def.

Parameters

**\*tensors**  
[[tf.Tensor](#)] passed tensors

**prod\_force\_virial**(atom\_ener: Tensor, natoms: Tensor) → [Tuple](#)[Tensor, Tensor, Tensor]

Compute force and virial.

Parameters

**atom\_ener**  
The atomic energy

**natoms**  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

Returns

**force**  
The force on atoms

**virial**  
The total virial

**atom\_virial**  
The atomic virial

**classmethod update\_sel**(global\_jdata: [dict](#), local\_jdata: [dict](#))

Update the selection and perform neighbor statistics.

Parameters

**global\_jdata**  
[[dict](#)] The global data, containing the training section

**local\_jdata**  
[[dict](#)] The local data refer to the current class

**class** deepmd.descriptor.**DescrptLocFrame**(\*args, \*\*kwargs)

Bases: [Descriptor](#)

Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.

Parameters

**rcut**  
The cut-off radius



`sel_a`  
`[list[str]]` The length of the list should be the same as the number of atom types in the system. `sel_a[i]` gives the selected number of type-*i* neighbors. The full relative coordinates of the neighbors are used by the descriptor.

`sel_r`  
`[list[str]]` The length of the list should be the same as the number of atom types in the system. `sel_r[i]` gives the selected number of type-*i* neighbors. Only relative distance of the neighbors are used by the descriptor. `sel_a[i] + sel_r[i]` is recommended to be larger than the maximally possible number of type-*i* neighbors in the cut-off radius.

`axis_rule`: list[int]  
 The length should be 6 times of the number of types. - `axis_rule[i*6+0]`: class of the atom defining the first axis of type-*i* atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - `axis_rule[i*6+1]`: type of the atom defining the first axis of type-*i* atom. - `axis_rule[i*6+2]`: index of the axis atom defining the first axis. Note that the neighbors with the same class and type are sorted according to their relative distance. - `axis_rule[i*6+3]`: class of the atom defining the second axis of type-*i* atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - `axis_rule[i*6+4]`: type of the atom defining the second axis of type-*i* atom. - `axis_rule[i*6+5]`: index of the axis atom defining the second axis. Note that the neighbors with the same class and type are sorted according to their relative distance.

#### Attributes

**explicit\_ntypes**  
 Explicit ntypes with type embedding.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Receive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>pass_tensors_from_frz_model(*tensors)</code>	Pass the <code>descript_reshape</code> tensor as well as <code>descript_deriv</code> tensor from the <code>frz_graph_def</code> .
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

**input\_dict**  
Dictionary for additional inputs

**reuse**  
The weights in the networks should be reused when get the variable.

**suffix**  
Name suffix to identify this descriptor

Returns

**descriptor**  
The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

**data\_coord**  
The coordinates. Can be generated by `deepmd.model.make_stat_input`

**data\_box**  
The box. Can be generated by `deepmd.model.make_stat_input`

**data\_atype**  
The atom types. Can be generated by `deepmd.model.make_stat_input`

**natoms\_vec**  
The vector for the number of atoms of the system and different types of atoms.  
Can be generated by `deepmd.model.make_stat_input`

**mesh**  
The mesh for neighbor searching. Can be generated by `deepmd.model.make_stat_input`

**input\_dict**  
Dictionary for additional input

**\*\*kwargs**  
Additional keyword arguments.

**get\_dim\_out**() → int

Returns the output dimension of this descriptor.

**get\_nlist**() → Tuple[Tensor, Tensor, List[int], List[int]]

Returns

**nlist**  
Neighbor list

**rij**  
The relative distance between the neighbor and the center atom.

**sel\_a**  
The number of neighbors with full information

**sel\_r**  
The number of neighbors with only radial information

`get_ntypes()` → `int`

Returns the number of atom types.

`get_rcut()` → `float`

Returns the cut-off radius.

`get_rot_mat()` → `Tensor`

Get rotational matrix.

`init_variables(graph: Graph, graph_def: GraphDef, suffix: str = '')` → `None`

Init the embedding net variables with the given dict.

Parameters

`graph`

[`tf.Graph`] The input frozen model graph

`graph_def`

[`tf.GraphDef`] The input frozen model graph\_def

`suffix`

[`str`, `optional`] The suffix of the scope

`prod_force_virial(atom_ener: Tensor, natoms: Tensor)` → `Tuple[Tensor, Tensor, Tensor]`

Compute force and virial.

Parameters

`atom_ener`

The atomic energy

`natoms`

The number of atoms. This tensor has the length of `Ntypes + 2` `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes + 2$ , number of type `i` atoms

Returns

`force`

The force on atoms

`virial`

The total virial

`atom_virial`

The atomic virial

`classmethod update_sel(global_jdata: dict, local_jdata: dict)`

Update the selection and perform neighbor statistics.

Parameters

`global_jdata`

[`dict`] The global data, containing the training section

`local_jdata`

[`dict`] The local data refer to the current class

`class deepmd.descriptor.DescriptSeA(*args, **kwargs)`

Bases: *DescriptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor  $\mathcal{D}^i \in \mathbb{R}^{M_1 \times M_2}$  is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i$$

where  $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$  is the coordinate matrix, and each row of  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{r_{ji}} \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{bmatrix}$$

where  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$  consists of outputs of a embedding network  $\mathcal{N}$  of  $s(r_{ji})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}_{<}^i \in \mathbb{R}^{N \times M_2}$  takes first  $M_2$  columns of  $\mathcal{G}^i$ . The equation of embedding network  $\mathcal{N}$  can be found at [deepmd.utils.network.embedding\\_net\(\)](#).

Parameters

rcut

The cut-off radius  $r_c$

rcut\_smth

From where the environment matrix should be smoothed  $r_s$

sel

`[list[str]] sel[i]` specifies the maximum number of type i atoms in the cut-off radius

neuron

`[list[int]]` Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

axis\_neuron

Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

resnet\_dt

Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

trainable

If the weights of embedding net are trainable.

seed

Random seed for initializing the network parameters.

type\_one\_side

Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

exclude\_types

`[List[List[int]]]` The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

`set_davg_zero`  
Set the shift of embedding net input to zero.

`activation_function`  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

`precision`  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

`uniform_seed`  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

`multi_task`  
If the model has multi fitting nets to train.

## References

[1]

Attributes

**`explicit_ntypes`**  
Explicit ntypes with type embedding.

**`precision`**  
Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Receive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = "") → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6,

pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input\_dict

    Dictionary for additional inputs

reuse

    The weights in the networks should be reused when get the variable.

suffix

    Name suffix to identify this descriptor

Returns

*descriptor*

    The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

    data\_coord

        The coordinates. Can be generated by deepmd.model.make\_stat\_input

    data\_box

        The box. Can be generated by deepmd.model.make\_stat\_input

    data\_atype

        The atom types. Can be generated by deepmd.model.make\_stat\_input

    natoms\_vec

        The vector for the number of atoms of the system and different types of atoms.  
        Can be generated by deepmd.model.make\_stat\_input

    mesh

        The mesh for neighbor searching. Can be generated by  
        deepmd.model.make\_stat\_input

    input\_dict

        Dictionary for additional input

    \*\*kwargs

        Additional keyword arguments.

**enable\_compression**(min\_nbor\_dist: float, graph: Graph, graph\_def: GraphDef, table\_extrapolate: float = 5, table\_stride\_1: float = 0.01, table\_stride\_2: float = 0.1, check\_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

    min\_nbor\_dist

        The nearest distance between atoms

    graph

        [tf.Graph] The graph of the model

    graph\_def

        [tf.GraphDef] The graph\_def of the model



`table_extrapolate`  
 The scale of model extrapolation

`table_stride_1`  
 The uniform stride of the first table

`table_stride_2`  
 The uniform stride of the second table

`check_frequency`  
 The overflow check frequency

`suffix`  
 [`str`, `optional`] The suffix of the scope

`enable_mixed_precision`(mixed\_prec: `Optional[dict]` = None) → `None`  
 Reveive the mixed precision setting.

Parameters

`mixed_prec`  
 The mixed precision setting used in the embedding net

`property explicit_ntypes: bool`  
 Explicit ntypes with type embedding.

`get_dim_out()` → `int`  
 Returns the output dimension of this descriptor.

`get_dim_rot_mat_1()` → `int`  
 Returns the first dimension of the rotation matrix. The rotation is of shape dim\_1 x 3.

`get_nlist()` → `Tuple[Tensor, Tensor, List[int], List[int]]`  
 Returns neighbor information.

Returns

`nlist`  
 Neighbor list

`rij`  
 The relative distance between the neighbor and the center atom.

`sel_a`  
 The number of neighbors with full information

`sel_r`  
 The number of neighbors with only radial information

`get_ntypes()` → `int`  
 Returns the number of atom types.

`get_rcut()` → `float`  
 Returns the cut-off radius.

`get_rot_mat()` → `Tensor`  
 Get rotational matrix.

`init_variables`(graph: `Graph`, graph\_def: `GraphDef`, suffix: `str` = "") → `None`  
 Init the embedding net variables with the given dict.

Parameters

```

graph
    [tf.Graph] The input frozen model graph
graph_def
    [tf.GraphDef] The input frozen model graph_def
suffix
    [str, optional] The suffix of the scope
merge_input_stats(stat_dict)
    Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
    Parameters
        stat_dict
            The dict of statistics computed from compute_input_stats, including:

            sumr
                The sum of radial statistics.

            suma
                The sum of relative coord statistics.

            sumn
                The sum of neighbor numbers.

            sumr2
                The sum of square of radial statistics.

            suma2
                The sum of square of relative coord statistics.

prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]
    Compute force and virial.
    Parameters
        atom_ener
            The atomic energy
        natoms
            The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number
            of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:
            2 ≤ i < Ntypes+2, number of type i atoms
    Returns
        force
            The force on atoms
        virial
            The total virial
        atom_virial
            The atomic virial

class deepmd.descriptor.DescriptSeAEbd(*args, **kwargs)
    Bases: DescriptSeA
    DeepPot-SE descriptor with type embedding approach.
    Parameters

```

rcut  
The cut-off radius

rcut\_smth  
From where the environment matrix should be smoothed

sel  
[`list[str]`] sel[i] specifies the maximum number of type i atoms in the cut-off radius

neuron  
[`list[int]`] Number of neurons in each hidden layers of the embedding net

axis\_neuron  
Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet\_dt  
Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

trainable  
If the weights of embedding net are trainable.

seed  
Random seed for initializing the network parameters.

type\_one\_side  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

type\_nchanl  
Number of channels for type representation

type\_nlayer  
Number of hidden layers for the type embedding net (skip connected).

numb\_aparam  
Number of atomic parameters. If  $>0$  it will be embedded with atom types.

set\_davg\_zero  
Set the shift of embedding net input to zero.

activation\_function  
The activation function in the embedding net. Supported options are {0}

precision  
The precision of the embedding net parameters. Supported options are {1}

exclude\_types  
[`List[List[int]]`] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

Attributes

**explicit\_ntypes**  
Explicit ntypes with type embedding.

**precision**  
Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Receive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from <code>compute_input_stats</code> to obtain the <code>self.davg</code> and <code>self.dstd</code> .
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the <code>descrpt_reshape</code> tensor as well as <code>descrpt_deriv</code> tensor from the <code>frz graph_def</code> .
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = "") → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6,

pbcs is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`

Dictionary for additional inputs

`reuse`

The weights in the networks should be reused when get the variable.

`suffix`

Name suffix to identify this descriptor

Returns

*descriptor*

The output descriptor

**class** deepmd.descriptor.DescriptSeAEbdV2(\*args, \*\*kwargs)

Bases: *DescriptSeA*

A compressible se\_a\_ebd model.

This model is a wrapper for DescriptorSeA, which set stripped\_type\_embedding=True.

Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statisitcs (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statisitcs computed from <code>compute_input_stats</code> to obtain the <code>self.davg</code> and <code>self.dstd</code> .
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the <code>descrpt_reshape</code> tensor as well as <code>descrpt_deriv</code> tensor from the <code>frz graph_def</code> .
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_class_by_input</code>	
---------------------------------	--

```
class deepmd.descriptor.DescriptSeAEf(*args, **kwargs)
```

Bases: *DescriptSe*

Smooth edition descriptor with Ef.

Parameters

`rcut`

The cut-off radius

`rcut_smth`

From where the environment matrix should be smoothed

`sel`

`[list[str]]` `sel[i]` specifies the maxmum number of type `i` atoms in the cut-off radius

`neuron`

`[list[int]]` Number of neurons in each hidden layers of the embedding net

`axis_neuron`

Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

**resnet\_dt**  
Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**  
If the weights of embedding net are trainable.

**seed**  
Random seed for initializing the network parameters.

**type\_one\_side**  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

**exclude\_types**  
[[`List`[[`List`[[`int`]]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

**set\_davg\_zero**  
Set the shift of embedding net input to zero.

**activation\_function**  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

#### Attributes

**explicit\_ntypes**  
Explicit ntypes with type embedding.

**precision**  
Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

Parameters

`coord_`  
The coordinate of atoms

`atype_`  
The type of atoms

`natoms`  
The number of atoms. This tensor has the length of Ntypes + 2 `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < \text{Ntypes} + 2$ , number of type  $i$  atoms

`box_`  
[`tf.Tensor`] The box of the system

`mesh`  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`  
Dictionary for additional inputs. Should have 'efield'.



reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

*descriptor*

The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data\_coord

The coordinates. Can be generated by deepmd.model.make\_stat\_input

data\_box

The box. Can be generated by deepmd.model.make\_stat\_input

data\_atype

The atom types. Can be generated by deepmd.model.make\_stat\_input

natoms\_vec

The vector for the number of atoms of the system and different types of atoms.  
Can be generated by deepmd.model.make\_stat\_input

mesh

The mesh for neighbor searching. Can be generated by  
deepmd.model.make\_stat\_input

input\_dict

Dictionary for additional input

\*\*kwargs

Additional keyword arguments.

**get\_dim\_out()** → int

Returns the output dimension of this descriptor.

**get\_dim\_rot\_mat\_1()** → int

Returns the first dimension of the rotation matrix. The rotation is of shape dim\_1 x 3.

**get\_nlist()** → Tuple[Tensor, Tensor, List[int], List[int]]

Returns neighbor information.

Returns

**nlist**

Neighbor list

**rij**

The relative distance between the neighbor and the center atom.

**sel\_a**

The number of neighbors with full information

**sel\_r**

The number of neighbors with only radial information

**get\_ntypes()** → *int*

Returns the number of atom types.

**get\_rcut()** → *float*

Returns the cut-off radius.

**get\_rot\_mat()** → *Tensor*

Get rotational matrix.

**prod\_force\_virial**(atom\_ener: *Tensor*, natoms: *Tensor*) → *Tuple*[*Tensor*, *Tensor*, *Tensor*]

Compute force and virial.

Parameters

**atom\_ener**

The atomic energy

**natoms**

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

Returns

**force**

The force on atoms

**virial**

The total virial

**atom\_virial**

The atomic virial

**class** deepmd.descriptor.DescriptSeAEfLower(\*args, \*\*kwargs)

Bases: *DescriptSeA*

Helper class for implementing DescriptSeAEf.

Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_, atype\_, natoms, box\_, mesh, input\_dict, suffix="", reuse=None)

Build the computational graph for the descriptor.

Parameters

coord\_

The coordinate of atoms

atype\_

The type of atoms

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

box\_

[[tf.Tensor](#)] The box of the system

mesh

For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

*descriptor*  
 The output descriptor

**compute\_input\_stats**(data\_coord, data\_box, data\_atype, natoms\_vec, mesh, input\_dict, \*\*kwargs)  
 Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
 The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`  
 The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`  
 The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`  
 The vector for the number of atoms of the system and different types of atoms.  
 Can be generated by `deepmd.model.make_stat_input`

`mesh`  
 The mesh for neighbor searching. Can be generated by  
`deepmd.model.make_stat_input`

`input_dict`  
 Dictionary for additional input

`**kwargs`  
 Additional keyword arguments.

**class** `deepmd.descriptor.DescriptSeAMask`(\*args, \*\*kwargs)

Bases: *DescriptSeA*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor  $\mathcal{D}^i \in \mathcal{R}^{M_1 \times M_2}$  is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i$$

where  $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$  is the coordinate matrix, and each row of  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{r_{ji}} \\ \frac{s(r_{ji})x_{ji}}{r_{ji}^2} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}^2} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}^2} \end{bmatrix}$$

where  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$  consists of outputs of a embedding network  $\mathcal{N}$  of  $s(r_{ji})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_{<} \in \mathbb{R}^{N \times M_2}$  takes first  $M_2$  columns of  $\mathcal{G}^i$ . The equation of embedding network  $\mathcal{N}$  can be found at `deepmd.utils.network.embedding_net()`. Specially for descriptor `se_a_mask` is a concise implementation of `se_a`. The difference is that `se_a_mask` only considered a non-pbc system. And accept a mask matrix to indicate the atom  $i$  in frame  $j$  is a real atom or not. (1 means real atom, 0 means ghost atom) Thus `se_a_mask` can accept a variable number of atoms in a frame.

Parameters

- `sel`  
[`list[str]`] `sel[i]` specifies the maximum number of type  $i$  atoms in the neighbor list.
- `neuron`  
[`list[int]`] Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$
- `axis_neuron`  
Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)
- `resnet_dt`  
Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$
- `trainable`  
If the weights of embedding net are trainable.
- `seed`  
Random seed for initializing the network parameters.
- `type_one_side`  
Try to build  $N_{types}$  embedding nets. Otherwise, building  $N_{types}^2$  embedding nets
- `exclude_types`  
[`List[List[int]]`] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.
- `activation_function`  
The activation function in the embedding net. Supported options are {0}
- `precision`  
The precision of the embedding net parameters. Supported options are {1}
- `uniform_seed`  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

## References

[1]

## Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.

## Methods

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cutoff radius.
<i>get_rot_mat</i> ()	Get rotational matrix.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (descript_reshape, ...)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: Dict[str, Any], reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

## Parameters

coord\_

The coordinate of atoms

`atype_`  
 The type of atoms  
`natoms`  
 The number of atoms. This tensor has the length of `Ntypes + 2` `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms  
`box_`  
`[tf.Tensor]` The box of the system  
`mesh`  
 For historical reasons, only the length of the Tensor matters. if size of `mesh == 6`, pbc is assumed. if size of `mesh == 0`, no-pbc is assumed.  
`input_dict`  
 Dictionary for additional inputs  
`reuse`  
 The weights in the networks should be reused when get the variable.  
`suffix`  
 Name suffix to identify this descriptor

Returns

`descriptor`  
 The output descriptor

`compute_input_stats`(`data_coord`: list, `data_box`: list, `data_atype`: list, `natoms_vec`: list, `mesh`: list, `input_dict`: dict, `**kwargs`) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
 The coordinates. Can be generated by `deepmd.model.make_stat_input`  
`data_box`  
 The box. Can be generated by `deepmd.model.make_stat_input`  
`data_atype`  
 The atom types. Can be generated by `deepmd.model.make_stat_input`  
`natoms_vec`  
 The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.make_stat_input`  
`mesh`  
 The mesh for neighbor searching. Can be generated by `deepmd.model.make_stat_input`  
`input_dict`  
 Dictionary for additional input  
`**kwargs`  
 Additional keyword arguments.

`get_rcut`() → float

Returns the cutoff radius.

`prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]`

Compute force and virial.

Parameters

`atom_ener`

The atomic energy

`natoms`

The number of atoms. This tensor has the length of `Ntypes + 2` `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes + 2$ , number of type `i` atoms

Returns

**force**

The force on atoms

**virial**

None for `se_a_mask` op

**atom\_virial**

None for `se_a_mask` op

`classmethod update_sel(global_jdata: dict, local_jdata: dict)`

Update the selection and perform neighbor statistics.

Parameters

`global_jdata`

[dict] The global data, containing the training section

`local_jdata`

[dict] The local data refer to the current class

`class deepmd.descriptor.DescriptSeAtten(*args, **kwargs)`

Bases: *DescriptSeA*

Smooth version descriptor with attention.

Parameters

`rcut`

The cut-off radius  $r_c$

`rcut_smth`

From where the environment matrix should be smoothed  $r_s$

`sel`

[list[str]] `sel[i]` specifies the maximum number of type `i` atoms in the cut-off radius

`neuron`

[list[int]] Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

`axis_neuron`

Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

`resnet_dt`

Time-step `dt` in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`trainable`

If the weights of embedding net are trainable.



**seed**  
Random seed for initializing the network parameters.

**type\_one\_side**  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

**exclude\_types**  
[List[List[int]]] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

**set\_davg\_zero**  
Set the shift of embedding net input to zero.

**activation\_function**  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

**attn**  
The length of hidden vector during scale-dot attention computation.

**attn\_layer**  
The number of layers in attention mechanism.

**attn\_dotr**  
Whether to dot the relative coordinates on the attention weights as a gated scheme.

**attn\_mask**  
Whether to mask the diagonal in the attention weights.

**multi\_task**  
If the model has multi fitting nets to train.

**stripped\_type\_embedding**  
Whether to strip the type embedding into a separated embedding network. Default value will be True in `se_atten_v2` descriptor.

**smooth\_type\_embdding**  
When using stripped type embedding, whether to dot smooth factor on the network output of type embedding to keep the network smooth, instead of setting `set_davg_zero` to be True. Default value will be True in `se_atten_v2` descriptor.

**Raises**

**ValueError**  
if `ntypes` is 0.

**Attributes**

**explicit\_ntypes**  
Explicit `ntypes` with type embedding.

**precision**  
Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the attention descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get\_class\_by\_input

**build**(coord\_ : Tensor, atype\_ : Tensor, natoms: Tensor, box\_ : Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = "") → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

`mesh`  
 For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

`descriptor`  
 The output descriptor

`build_type_exclude_mask(exclude_types: List[Tuple[int, int]], ntypes: int, sel: List[int], ndescript: int, atype: Tensor, shape0: Tensor, nei_type_vec: Tensor) → Tensor`

Build the type exclude mask for the attention descriptor.

Parameters

`exclude_types`  
`[List[Tuple[int, int]]]` The list of excluded types, e.g. [(0, 1), (1, 0)] means the interaction between type 0 and type 1 is excluded.

`ntypes`  
`[int]` The number of types.

`sel`  
`[List[int]]` The list of the number of selected neighbors for each type.

`ndescript`  
`[int]` The number of descriptors for each atom.

`atype`  
`[tf.Tensor]` The type of atoms, with the size of shape0.

`shape0`  
`[tf.Tensor]` The shape of the first dimension of the inputs, which is equal to nsamples \* natoms.

`nei_type_vec`  
`[tf.Tensor]` The type of neighbors, with the size of (shape0, nnei).

Returns

`tf.Tensor`  
 The type exclude mask, with the shape of (shape0, ndescript), and the precision of GLOBAL\_TF\_FLOAT\_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

See also:

`deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask`

## Notes

This method has the similar way to build the type exclude mask as `deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask()`. The mathematical expression has been explained in that method. The difference is that the attention descriptor has provided the type of the neighbors (`idx_j`) that is not in order, so we use it from an extra input.

```
compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list,
                    input_dict: dict, mixed_type: bool = False, real_natoms_vec: Optional[list] =
                    None, **kwargs) → None
```

Compute the statistics (avg and std) of the training data. The input will be normalized by the statistics.

### Parameters

`data_coord`

The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`

The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`

The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`

The vector for the number of atoms of the system and different types of atoms. If `mixed_type` is True, this para is blank. See `real_natoms_vec`.

`mesh`

The mesh for neighbor searching. Can be generated by `deepmd.model.make_stat_input`

`input_dict`

Dictionary for additional input

`mixed_type`

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

`real_natoms_vec`

If `mixed_type` is True, it takes in the real `natoms_vec` for each frame.

`**kwargs`

Additional keyword arguments.

```
enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate:
                    float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1,
                    check_frequency: int = -1, suffix: str = '') → None
```

Reveive the statistics (distance, max\_nbor\_size and env\_mat\_range) of the training data.

### Parameters

`min_nbor_dist`

The nearest distance between atoms

`graph`

[`tf.Graph`] The graph of the model

`graph_def`

[`tf.GraphDef`] The graph\_def of the model

```

    table_extrapolate
        The scale of model extrapolation
    table_stride_1
        The uniform stride of the first table
    table_stride_2
        The uniform stride of the second table
    check_frequency
        The overflow check frequency
    suffix
        [str, optional] The suffix of the scope
property explicit_ntypes: bool
    Explicit ntypes with type embedding.
init_variables(graph: Graph, graph_def: GraphDef, suffix: str = '') → None
    Init the embedding net variables with the given dict.
    Parameters
        graph
            [tf.Graph] The input frozen model graph
        graph_def
            [tf.GraphDef] The input frozen model graph_def
        suffix
            [str, optional] The suffix of the scope
classmethod update_sel(global_jdata: dict, local_jdata: dict)
    Update the selection and perform neighbor statistics.
    Parameters
        global_jdata
            [dict] The global data, containing the training section
        local_jdata
            [dict] The local data refer to the current class
class deepmd.descriptor.DescriptSeAttenV2(*args, **kwargs)
    Bases: DescriptSeAtten
    Smooth version 2.0 descriptor with attention.
    Parameters
        rcut
            The cut-off radius  $r_c$ 
        rcut_smth
            From where the environment matrix should be smoothed  $r_s$ 
        sel
            [list[str]] sel[i] specifies the maxmum number of type i atoms in the cut-off radius
        neuron
            [list[int]] Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$ 

```

**axis\_neuron**  
Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

**resnet\_dt**  
Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**  
If the weights of embedding net are trainable.

**seed**  
Random seed for initializing the network parameters.

**type\_one\_side**  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

**exclude\_types**  
[[`List`[[`List`[[`int`]]]] The excluded pairs of types which have no interaction with each other. For example, [[0, 1]] means no interaction between type 0 and type 1.

**set\_davg\_zero**  
Set the shift of embedding net input to zero.

**activation\_function**  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

**attn**  
The length of hidden vector during scale-dot attention computation.

**attn\_layer**  
The number of layers in attention mechanism.

**attn\_dotr**  
Whether to dot the relative coordinates on the attention weights as a gated scheme.

**attn\_mask**  
Whether to mask the diagonal in the attention weights.

**multi\_task**  
If the model has multi fitting nets to train.

#### Attributes

**explicit\_ntypes**  
Explicit ntypes with type embedding.

**precision**  
Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the attention descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from <code>compute_input_stats</code> to obtain the <code>self.davg</code> and <code>self.dstd</code> .
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the <code>descrpt_reshape</code> tensor as well as <code>descrpt_deriv</code> tensor from the <code>frz graph_def</code> .
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get\_class\_by\_input

```
class deepmd.descriptor.DescriptSeR(*args, **kwargs)
```

Bases: *DescriptSe*

DeepPot-SE constructed from radial information of atomic configurations.

The embedding takes the distance between atoms as input.

Parameters

`rcut`

The cut-off radius

`rcut_smth`

From where the environment matrix should be smoothed

`sel`

`[list[str]] sel[i]` specifies the maxmum number of type `i` atoms in the cut-off radius

`neuron`

`[list[int]]` Number of neurons in each hidden layers of the embedding net

`resnet_dt`

Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`trainable`

If the weights of embedding net are trainable.

`seed`

Random seed for initializing the network parameters.

`type_one_side`

Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

`exclude_types`

`[List[List[int]]]` The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

`activation_function`

The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

`precision`

The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

`uniform_seed`

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**`explicit_ntypes`**

Explicit ntypes with type embedding.

**`precision`**

Precision of filter network.



## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Receive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = "") → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`  
Dictionary for additional inputs

`reuse`  
The weights in the networks should be reused when get the variable.

`suffix`  
Name suffix to identify this descriptor

Returns

*descriptor*  
The output descriptor

**compute\_input\_stats**(data\_coord, data\_box, data\_atype, natoms\_vec, mesh, input\_dict, \*\*kwargs)  
Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`  
The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`  
The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`  
The vector for the number of atoms of the system and different types of atoms.  
Can be generated by `deepmd.model.make_stat_input`

`mesh`  
The mesh for neighbor searching. Can be generated by  
`deepmd.model.make_stat_input`

`input_dict`  
Dictionary for additional input

`**kwargs`  
Additional keyword arguments.

**enable\_compression**(min\_nbor\_dist: float, graph: Graph, graph\_def: GraphDef, table\_extrapolate: float = 5, table\_stride\_1: float = 0.01, table\_stride\_2: float = 0.1, check\_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

`min_nbor_dist`  
The nearest distance between atoms

`graph`  
[`tf.Graph`] The graph of the model

`graph_def`  
[`tf.GraphDef`] The graph\_def of the model

`table_extrapolate`  
The scale of model extrapolation

`table_stride_1`  
 The uniform stride of the first table

`table_stride_2`  
 The uniform stride of the second table

`check_frequency`  
 The overflow check frequency

`suffix`  
`[str, optional]` The suffix of the scope

**`get_dim_out()`**  
 Returns the output dimension of this descriptor.

**`get_nlist()`**  
 Returns neighbor information.  
 Returns

**`nlist`**  
 Neighbor list

**`rij`**  
 The relative distance between the neighbor and the center atom.

**`sel_a`**  
 The number of neighbors with full information

**`sel_r`**  
 The number of neighbors with only radial information

**`get_ntypes()`**  
 Returns the number of atom types.

**`get_rcut()`**  
 Returns the cut-off radius.

**`merge_input_stats(stat_dict)`**  
 Merge the statisitcs computed from `compute_input_stats` to obtain the `self.davg` and `self.dstd`.  
 Parameters

`stat_dict`  
 The dict of statisitcs computed from `compute_input_stats`, including:

`sumr`  
 The sum of radial statisitcs.

`sumn`  
 The sum of neighbor numbers.

`sumr2`  
 The sum of square of radial statisitcs.

**`prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]`**  
 Compute force and virial.  
 Parameters

`atom_ener`  
 The atomic energy

**natoms**

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

Returns

**force**

The force on atoms

**virial**

The total virial

**atom\_virial**

The atomic virial

**class deepmd.descriptor.DescriptSeT(\*args, \*\*kwargs)**

Bases: *DescriptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations.

The embedding takes angles between two neighboring atoms as input.

Parameters

**rcut**

The cut-off radius

**rcut\_smth**

From where the environment matrix should be smoothed

**sel**

[*list[str]*] sel[i] specifies the maximum number of type i atoms in the cut-off radius

**neuron**

[*list[int]*] Number of neurons in each hidden layers of the embedding net

**resnet\_dt**

Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**

If the weights of embedding net are trainable.

**seed**

Random seed for initializing the network parameters.

**set\_davg\_zero**

Set the shift of embedding net input to zero.

**activation\_function**

The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**

The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.

**Methods**

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (descript_reshape, ...)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz_graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

**Parameters****coord\_**

The coordinate of atoms

**atype\_**

The type of atoms

**natoms**

The number of atoms. This tensor has the length of Ntypes + 2  
 natoms[0]: number of local atoms  
 natoms[1]: total number of atoms held by this processor  
 natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[`tf.Tensor`] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input\_dict  
Dictionary for additional inputs

reuse  
The weights in the networks should be reused when get the variable.

suffix  
Name suffix to identify this descriptor

Returns

*descriptor*  
The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data\_coord  
The coordinates. Can be generated by `deepmd.model.make_stat_input`

data\_box  
The box. Can be generated by `deepmd.model.make_stat_input`

data\_atype  
The atom types. Can be generated by `deepmd.model.make_stat_input`

natoms\_vec  
The vector for the number of atoms of the system and different types of atoms.  
Can be generated by `deepmd.model.make_stat_input`

mesh  
The mesh for neighbor searching. Can be generated by `deepmd.model.make_stat_input`

input\_dict  
Dictionary for additional input

\*\*kwargs  
Additional keyword arguments.

**enable\_compression**(min\_nbor\_dist: float, graph: Graph, graph\_def: GraphDef, table\_extrapolate: float = 5, table\_stride\_1: float = 0.01, table\_stride\_2: float = 0.1, check\_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

min\_nbor\_dist  
The nearest distance between atoms

graph  
[`tf.Graph`] The graph of the model

**graph\_def**  
 [tf.GraphDef] The graph\_def of the model

**table\_extrapolate**  
 The scale of model extrapolation

**table\_stride\_1**  
 The uniform stride of the first table

**table\_stride\_2**  
 The uniform stride of the second table

**check\_frequency**  
 The overflow check frequency

**suffix**  
 [str, optional] The suffix of the scope

**get\_dim\_out()** → int  
 Returns the output dimension of this descriptor.

**get\_nlist()** → Tuple[Tensor, Tensor, List[int], List[int]]  
 Returns neighbor information.  
 Returns
 

- nlist**  
Neighbor list
- rij**  
The relative distance between the neighbor and the center atom.
- sel\_a**  
The number of neighbors with full information
- sel\_r**  
The number of neighbors with only radial information

**get\_ntypes()** → int  
 Returns the number of atom types.

**get\_rcut()** → float  
 Returns the cut-off radius.

**merge\_input\_stats(stat\_dict)**  
 Merge the statisitcs computed from compute\_input\_stats to obtain the self.davg and self.dstd.  
 Parameters
 

- stat\_dict**  
The dict of statisitcs computed from compute\_input\_stats, including:
  - sumr**  
The sum of radial statisitcs.
  - suma**  
The sum of relative coord statisitcs.
  - sumn**  
The sum of neighbor numbers.

**sumr2**  
The sum of square of radial statisits.

**suma2**  
The sum of square of relative coord statisits.

**prod\_force\_virial**(atom\_ener: Tensor, natoms: Tensor) → [Tuple](#)[Tensor, Tensor, Tensor]

Compute force and virial.

Parameters

**atom\_ener**  
The atomic energy

**natoms**  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

Returns

**force**  
The force on atoms

**virial**  
The total virial

**atom\_virial**  
The atomic virial

## Submodules

### `deepmd.descriptor.descriptor` module

`class deepmd.descriptor.descriptor.Descriptor(*args, **kwargs)`

Bases: [PluginVariant](#)

The abstract class for descriptors. All specific descriptors should be based on this class.

The descriptor  $\mathcal{D}$  describes the environment of an atom, which should be a function of coordinates and types of its neighbour atoms.

## Notes

Only methods and attributes defined in this class are generally public, that can be called by other classes.



## Examples

```
>>> descript = Descriptor(type="se_e2_a", rcut=6., rcut_smth=0.5, sel=[50])
>>> type(descript)
<class 'deepmd.descriptor.se_a.DescriptSeA'>
```

## Attributes

### `explicit_ntypes`

Explicit ntypes with type embedding.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>pass_tensors_from_frz_model(*tensors)</code>	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**abstract** `build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box_: Tensor, mesh: Tensor, input_dict: Dict[str, Any], reuse: Optional[bool] = None, suffix: str = '')` → Tensor

Build the computational graph for the descriptor.

## Parameters

`coord_`  
[`tf.Tensor`] The coordinate of atoms

`atype_`  
[`tf.Tensor`] The type of atoms

`natoms`  
 [`tf.Tensor`] The number of atoms. This tensor has the length of `Ntypes + 2`  
`natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this  
 processor `natoms[i]`:  $2 \leq i < Ntypes + 2$ , number of type `i` atoms  
  
`box_`  
 [`tf.Tensor`] The box of frames  
  
`mesh`  
 [`tf.Tensor`] For historical reasons, only the length of the Tensor matters. if size of  
`mesh == 6`, pbc is assumed. if size of `mesh == 0`, no-pbc is assumed.  
  
`input_dict`  
 [`dict[str, Any]`] Dictionary for additional inputs  
  
`reuse`  
 [`bool, optional`] The weights in the networks should be reused when get the vari-  
 able.  
  
`suffix`  
 [`str, optional`] Name suffix to identify this descriptor  
  
 Returns  
  
 descriptor: `tf.Tensor`  
 The output descriptor

## Notes

This method must be implemented, as it's called by other classes.

`build_type_exclude_mask(exclude_types: List[Tuple[int, int]], ntypes: int, sel: List[int], ndescript:`  
`int, atype: Tensor, shape0: Tensor) → Tensor`

Build the type exclude mask for the descriptor.

### Parameters

`exclude_types`  
 [`List[Tuple[int, int]]`] The list of excluded types, e.g. `[(0, 1), (1, 0)]` means the  
 interaction between type 0 and type 1 is excluded.  
  
`ntypes`  
 [`int`] The number of types.  
  
`sel`  
 [`List[int]`] The list of the number of selected neighbors for each type.  
  
`ndescript`  
 [`int`] The number of descriptors for each atom.  
  
`atype`  
 [`tf.Tensor`] The type of atoms, with the size of `shape0`.  
  
`shape0`  
 [`tf.Tensor`] The shape of the first dimension of the inputs, which is equal to `nsam-`  
`ples * natoms`.

### Returns

`tf.Tensor`  
 The type exclude mask, with the shape of `(shape0, ndescript)`, and the precision of

GLOBAL\_TF\_FLOAT\_PRECISION. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

## Notes

To exclude the interaction between two types, the derivative of energy with respect to distances (or angles) between two atoms should be zero[Rafclae60e195-1], i.e.

$$\forall i \in \text{type 1}, j \in \text{type 2}, \frac{\partial E}{\partial r_{ij}} = 0$$

When embedding networks between every two types are built, we can just remove that network. But when type\_one\_side is enabled, a network may be built for multiple pairs of types. In this case, we need to build a mask to exclude the interaction between two types.

The mask assumes the descriptors are sorted by neighbor type with the fixed number of given sel and each neighbor has the same number of descriptors (for example 4).

## References

[1]

```
abstract compute_input_stats(data_coord: List[ndarray], data_box: List[ndarray], data_atype:
                             List[ndarray], natoms_vec: List[ndarray], mesh: List[ndarray],
                             input_dict: Dict[str, List[ndarray]], **kwargs) → None
```

Compute the statistics (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
[list[np.ndarray]] The coordinates. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`data_box`  
[list[np.ndarray]] The box. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`data_atype`  
[list[np.ndarray]] The atom types. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`natoms_vec`  
[list[np.ndarray]] The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`mesh`  
[list[np.ndarray]] The mesh for neighbor searching. Can be generated by `deepmd.model.model_stat.make_stat_input()`

`input_dict`  
[dict[str, list[np.ndarray]]] Dictionary for additional input

`**kwargs`  
Additional keyword arguments which may contain `mixed_type` and `real_natoms_vec`.

### Notes

This method must be implemented, as it's called by other classes.

```
enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate:
                    float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1,
                    check_frequency: int = -1, suffix: str = '') → None
```

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

#### Parameters

**min\_nbor\_dist**  
[float] The nearest distance between atoms

**graph**  
[tf.Graph] The graph of the model

**graph\_def**  
[tf.GraphDef] The graph definition of the model

**table\_extrapolate**  
[float, default: 5.] The scale of model extrapolation

**table\_stride\_1**  
[float, default: 0.01] The uniform stride of the first table

**table\_stride\_2**  
[float, default: 0.1] The uniform stride of the second table

**check\_frequency**  
[int, default: -1] The overflow check frequency

**suffix**  
[str, optional] The suffix of the scope

### Notes

This method is called by others when the descriptor supported compression.

```
enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
```

Reveive the mixed precision setting.

#### Parameters

**mixed\_prec**  
The mixed precision setting used in the embedding net

### Notes

This method is called by others when the descriptor supported compression.

```
property explicit_ntypes: bool
```

Explicit ntypes with type embedding.

```
classmethod get_class_by_input(input: dict)
```

**abstract** `get_dim_out()`  $\rightarrow$  `int`

Returns the output dimension of this descriptor.

Returns

`int`  
the output dimension of this descriptor

### Notes

This method must be implemented, as it's called by other classes.

**get\_dim\_rot\_mat\_1()**  $\rightarrow$  `int`

Returns the first dimension of the rotation matrix. The rotation is of shape `dim_1` x 3.

Returns

`int`  
the first dimension of the rotation matrix

**get\_nlist()**  $\rightarrow$  `Tuple[Tensor, Tensor, List[int], List[int]]`

Returns neighbor information.

Returns

`nlist`  
`[tf.Tensor]` Neighbor list

`rij`  
`[tf.Tensor]` The relative distance between the neighbor and the center atom.

`sel_a`  
`[list[int]]` The number of neighbors with full information

`sel_r`  
`[list[int]]` The number of neighbors with only radial information

**abstract** `get_ntypes()`  $\rightarrow$  `int`

Returns the number of atom types.

Returns

`int`  
the number of atom types

### Notes

This method must be implemented, as it's called by other classes.

**abstract** `get_rcut()`  $\rightarrow$  `float`

Returns the cut-off radius.

Returns

`float`  
the cut-off radius

### Notes

This method must be implemented, as it's called by other classes.

**get\_tensor\_names**(suffix: str = "") → Tuple[str]

Get names of tensors.

Parameters

suffix  
[str] The suffix of the scope

Returns

Tuple[str]  
Names of tensors

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: str = "") → None

Init the embedding net variables with the given dict.

Parameters

graph  
[tf.Graph] The input frozen model graph  
graph\_def  
[tf.GraphDef] The input frozen model graph\_def  
suffix  
[str, optional] The suffix of the scope

### Notes

This method is called by others when the descriptor supported initialization from the given variables.

**pass\_tensors\_from\_frz\_model**(\*tensors: Tensor) → None

Pass the descrpt\_reshape tensor as well as descrpt\_deriv tensor from the frz graph\_def.

Parameters

\*tensors  
[tf.Tensor] passed tensors

### Notes

The number of parameters in the method must be equal to the numbers of returns in `get_tensor_names()`.

**abstract prod\_force\_virial**(atom\_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]

Compute force and virial.

Parameters

atom\_ener  
[tf.Tensor] The atomic energy

natoms  
 [tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2  
 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this  
 processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

Returns

force  
 [tf.Tensor] The force on atoms  
 virial  
 [tf.Tensor] The total virial  
 atom\_virial  
 [tf.Tensor] The atomic virial

**static register**(key: str) → Callable

Register a descriptor plugin.

Parameters

key  
 [str] the key of a descriptor

Returns

*Descriptor*  
 the registered descriptor

## Examples

```
>>> @Descriptor.register("some_descript")
class SomeDescript(Descriptor):
    pass
```

**abstract classmethod update\_sel**(global\_jdata: dict, local\_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global\_jdata  
 [dict] The global data, containing the training section

local\_jdata  
 [dict] The local data refer to the current class

## deepmd.descriptor.hybrid module

**class** deepmd.descriptor.hybrid.DescriptHybrid(\*args, \*\*kwargs)

Bases: *Descriptor*

Concat a list of descriptors to form a new descriptor.

Parameters

list  
 [list] Build a descriptor from the concatenation of the list of descriptors.

Attributes

*explicit\_ntypes*

Explicit ntypes with type embedding.

**Methods**

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.
<i>get_nlist_i</i> (ii)	Get the neighbor information of the ii-th descriptor.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (*tensors)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz_graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get\_class\_by\_input

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_

The coordinate of atoms

atype\_

The type of atoms

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number



of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  
 $2 \leq i < \text{Ntypes} + 2$ , number of type  $i$  atoms

`box_`  
`[tf.Tensor]` The box of the system

`mesh`  
 For historical reasons, only the length of the Tensor matters. if size of `mesh == 6`,  
 pbc is assumed. if size of `mesh == 0`, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

**descriptor**  
 The output descriptor

**compute\_input\_stats**(`data_coord`: list, `data_box`: list, `data_atype`: list, `natoms_vec`: list, `mesh`: list,  
`input_dict`: dict, `mixed_type`: bool = False, `real_natoms_vec`: Optional[list] =  
 None, **\*\*kwargs**) → None

Compute the statistics (avg and std) of the training data. The input will be normalized by the  
 statistics.

Parameters

`data_coord`  
 The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`  
 The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`  
 The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`  
 The vector for the number of atoms of the system and different types of atoms.  
 Can be generated by `deepmd.model.make_stat_input`

`mesh`  
 The mesh for neighbor searching. Can be generated by  
`deepmd.model.make_stat_input`

`input_dict`  
 Dictionary for additional input

`mixed_type`  
 Whether to perform the `mixed_type` mode. If True, the input data has the  
`mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a sys-  
 tem may have different `natoms_vec`(s), with the same `nloc`.

`real_natoms_vec`  
 If `mixed_type` is True, it takes in the real `natoms_vec` for each frame.

**\*\*kwargs**  
 Additional keyword arguments.

```
enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate:
    float = 5.0, table_stride_1: float = 0.01, table_stride_2: float = 0.1,
    check_frequency: int = -1, suffix: str = "") → None
```

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

```
min_nbor_dist
    [float] The nearest distance between atoms

graph
    [tf.Graph] The graph of the model

graph_def
    [tf.GraphDef] The graph_def of the model

table_extrapolate
    [float, default: 5.] The scale of model extrapolation

table_stride_1
    [float, default: 0.01] The uniform stride of the first table

table_stride_2
    [float, default: 0.1] The uniform stride of the second table

check_frequency
    [int, default: -1] The overflow check frequency

suffix
    [str, optional] The suffix of the scope
```

```
enable_mixed_precision(mixed_prec: Optional[dict] = None) → None
```

Reveive the mixed precision setting.

Parameters

```
mixed_prec
    The mixed precision setting used in the embedding net
```

```
property explicit_ntypes: bool
```

Explicit ntypes with type embedding.

```
get_dim_out() → int
```

Returns the output dimension of this descriptor.

```
get_nlist() → Tuple[Tensor, Tensor, List[int], List[int]]
```

Get the neighbor information of the descriptor, returns the nlist of the descriptor with the largest cut-off radius.

Returns

```
nlist
    Neighbor list

rij
    The relative distance between the neighbor and the center atom.

sel_a
    The number of neighbors with full information

sel_r
    The number of neighbors with only radial information
```

`get_nlist_i(ii: int) → Tuple[Tensor, Tensor, List[int], List[int]]`

Get the neighbor information of the ii-th descriptor.

Parameters

`ii`  
[int] The index of the descriptor

Returns

`nlist`  
Neighbor list

`rij`  
The relative distance between the neighbor and the center atom.

`sel_a`  
The number of neighbors with full information

`sel_r`  
The number of neighbors with only radial information

`get_ntypes() → int`

Returns the number of atom types.

`get_rcut() → float`

Returns the cut-off radius.

`get_tensor_names(suffix: str = '') → Tuple[str]`

Get names of tensors.

Parameters

`suffix`  
[str] The suffix of the scope

Returns

`Tuple[str]`  
Names of tensors

`init_variables(graph: Graph, graph_def: GraphDef, suffix: str = '') → None`

Init the embedding net variables with the given dict.

Parameters

`graph`  
[tf.Graph] The input frozen model graph

`graph_def`  
[tf.GraphDef] The input frozen model graph\_def

`suffix`  
[str, optional] The suffix of the scope

`merge_input_stats(stat_dict)`

Merge the statistics computed from compute\_input\_stats to obtain the self.davg and self.dstd.

Parameters

`stat_dict`  
The dict of statistics computed from compute\_input\_stats, including:

**sumr**  
The sum of radial statisitscs.

**suma**  
The sum of relative coord statisitscs.

**sumn**  
The sum of neighbor numbers.

**sumr2**  
The sum of square of radial statisitscs.

**suma2**  
The sum of square of relative coord statisitscs.

**pass\_tensors\_from\_frz\_model**(\*tensors: Tensor) → None

Pass the descrpt\_reshape tensor as well as descrpt\_deriv tensor from the frz graph\_def.

Parameters

**\*tensors**  
[[tf.Tensor](#)] passed tensors

**prod\_force\_virial**(atom\_ener: Tensor, natoms: Tensor) → [Tuple](#)[Tensor, Tensor, Tensor]

Compute force and virial.

Parameters

**atom\_ener**  
The atomic energy

**natoms**  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 <= i < Ntypes+2, number of type i atoms

Returns

**force**  
The force on atoms

**virial**  
The total virial

**atom\_virial**  
The atomic virial

**classmethod update\_sel**(global\_jdata: [dict](#), local\_jdata: [dict](#))

Update the selection and perform neighbor statistics.

Parameters

**global\_jdata**  
[[dict](#)] The global data, containing the training section

**local\_jdata**  
[[dict](#)] The local data refer to the current class

**deepmd.descriptor.loc\_frame module**

```
class deepmd.descriptor.loc_frame.DescriptLocFrame(*args, **kwargs)
```

Bases: *Descriptor*

Defines a local frame at each atom, and the compute the descriptor as local coordinates under this frame.

**Parameters**

**rcut**

The cut-off radius

**sel\_a**

[list[str]] The length of the list should be the same as the number of atom types in the system. sel\_a[i] gives the selected number of type-i neighbors. The full relative coordinates of the neighbors are used by the descriptor.

**sel\_r**

[list[str]] The length of the list should be the same as the number of atom types in the system. sel\_r[i] gives the selected number of type-i neighbors. Only relative distance of the neighbors are used by the descriptor. sel\_a[i] + sel\_r[i] is recommended to be larger than the maximally possible number of type-i neighbors in the cut-off radius.

**axis\_rule**: list[int]

The length should be 6 times of the number of types. - axis\_rule[i\*6+0]: class of the atom defining the first axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - axis\_rule[i\*6+1]: type of the atom defining the first axis of type-i atom. - axis\_rule[i\*6+2]: index of the axis atom defining the first axis. Note that the neighbors with the same class and type are sorted according to their relative distance. - axis\_rule[i\*6+3]: class of the atom defining the second axis of type-i atom. 0 for neighbors with full coordinates and 1 for neighbors only with relative distance. - axis\_rule[i\*6+4]: type of the atom defining the second axis of type-i atom. - axis\_rule[i\*6+5]: index of the axis atom defining the second axis. Note that the neighbors with the same class and type are sorted according to their relative distance.

**Attributes**

**explicit\_ntypes**

Explicit ntypes with type embedding.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Receive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>pass_tensors_from_frz_model(*tensors)</code>	Pass the <code>descript_reshape</code> tensor as well as <code>descript_deriv</code> tensor from the <code>frz_graph_def</code> .
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

**input\_dict**  
Dictionary for additional inputs

**reuse**  
The weights in the networks should be reused when get the variable.

**suffix**  
Name suffix to identify this descriptor

Returns

**descriptor**  
The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

**data\_coord**  
The coordinates. Can be generated by `deepmd.model.make_stat_input`

**data\_box**  
The box. Can be generated by `deepmd.model.make_stat_input`

**data\_atype**  
The atom types. Can be generated by `deepmd.model.make_stat_input`

**natoms\_vec**  
The vector for the number of atoms of the system and different types of atoms.  
Can be generated by `deepmd.model.make_stat_input`

**mesh**  
The mesh for neighbor searching. Can be generated by `deepmd.model.make_stat_input`

**input\_dict**  
Dictionary for additional input

**\*\*kwargs**  
Additional keyword arguments.

**get\_dim\_out**() → int

Returns the output dimension of this descriptor.

**get\_nlist**() → Tuple[Tensor, Tensor, List[int], List[int]]

Returns

**nlist**  
Neighbor list

**rij**  
The relative distance between the neighbor and the center atom.

**sel\_a**  
The number of neighbors with full information

**sel\_r**  
The number of neighbors with only radial information

**get\_ntypes()** → `int`

Returns the number of atom types.

**get\_rcut()** → `float`

Returns the cut-off radius.

**get\_rot\_mat()** → `Tensor`

Get rotational matrix.

**init\_variables**(graph: `Graph`, graph\_def: `GraphDef`, suffix: `str` = "") → `None`

Init the embedding net variables with the given dict.

Parameters

graph

[`tf.Graph`] The input frozen model graph

graph\_def

[`tf.GraphDef`] The input frozen model graph\_def

suffix

[`str`, `optional`] The suffix of the scope

**prod\_force\_virial**(atom\_ener: `Tensor`, natoms: `Tensor`) → `Tuple`[`Tensor`, `Tensor`, `Tensor`]

Compute force and virial.

Parameters

atom\_ener

The atomic energy

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

Returns

**force**

The force on atoms

**virial**

The total virial

**atom\_virial**

The atomic virial

**classmethod update\_sel**(global\_jdata: `dict`, local\_jdata: `dict`)

Update the selection and perform neighbor statistics.

Parameters

global\_jdata

[`dict`] The global data, containing the training section

local\_jdata

[`dict`] The local data refer to the current class



**deepmd.descriptor.se module**

```
class deepmd.descriptor.se.DescriptSe(*args, **kwargs)
```

Bases: *Descriptor*

A base class for smooth version of descriptors.

**Notes**

All of these descriptors have an environmental matrix and an embedding network (*deepmd.utils.network.embedding\_net()*), so they can share some similiar methods without defining them twice.

**Attributes**

embedding\_net\_variables  
     [dict] initial embedding network variables

descript\_reshape  
     [tf.Tensor] the reshaped descriptor

descript\_deriv  
     [tf.Tensor] the descriptor derivative

rij  
     [tf.Tensor] distances between two atoms

nlist  
     [tf.Tensor] the neighbor list

**Methods**

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statisitcs (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>pass_tensors_from_frz_model(descript_reshape, ...)</code>	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**get\_tensor\_names**(suffix: str = '') → Tuple[str]

Get names of tensors.

Parameters

suffix  
[str] The suffix of the scope

Returns

Tuple[str]  
Names of tensors

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: str = '') → None

Init the embedding net variables with the given dict.

Parameters

graph  
[tf.Graph] The input frozen model graph  
graph\_def  
[tf.GraphDef] The input frozen model graph\_def  
suffix  
[str, optional] The suffix of the scope

**pass\_tensors\_from\_frz\_model**(descript\_reshape: Tensor, descript\_deriv: Tensor, rij: Tensor, nlist: Tensor)

Pass the descript\_reshape tensor as well as descript\_deriv tensor from the frz graph\_def.

Parameters

descript\_reshape  
The passed descript\_reshape tensor  
descript\_deriv  
The passed descript\_deriv tensor  
rij  
The passed rij tensor  
nlist  
The passed nlist tensor

**property precision: DType**

Precision of filter network.

**classmethod update\_sel**(global\_jdata: dict, local\_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global\_jdata  
[dict] The global data, containing the training section  
local\_jdata  
[dict] The local data refer to the current class

## deepmd.descriptor.se\_a module

```
class deepmd.descriptor.se_a.DescriptSeA(*args, **kwargs)
```

Bases: *DescriptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor  $\mathcal{D}^i \in \mathbb{R}^{M_1 \times M_2}$  is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i$$

where  $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$  is the coordinate matrix, and each row of  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{r_{ji}} \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{bmatrix}$$

where  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$  consists of outputs of a embedding network  $\mathcal{N}$  of  $s(r_{ji})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}_{<}^i \in \mathbb{R}^{N \times M_2}$  takes first  $M_2$  columns of  $\mathcal{G}^i$ . The equation of embedding network  $\mathcal{N}$  can be found at `deepmd.utils.network.embedding_net()`.

Parameters

rcut

The cut-off radius  $r_c$

rcut\_smth

From where the environment matrix should be smoothed  $r_s$

sel

[list[str]] sel[i] specifies the maximum number of type i atoms in the cut-off radius

neuron

[list[int]] Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

axis\_neuron

Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

resnet\_dt

Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

trainable

If the weights of embedding net are trainable.

`seed`  
Random seed for initializing the network parameters.

`type_one_side`  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

`exclude_types`  
[`List`[`List`[`int`]]] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

`set_davg_zero`  
Set the shift of embedding net input to zero.

`activation_function`  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

`precision`  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

`uniform_seed`  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

`multi_task`  
If the model has multi fitting nets to train.

## References

[1]

Attributes

`explicit_ntypes`

Explicit ntypes with type embedding.

`precision`

Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get\_class\_by\_input

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = "") → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6,

pbc is assumed. if size of mesh == 0, no-pbc is assumed.

input\_dict

    Dictionary for additional inputs

reuse

    The weights in the networks should be reused when get the variable.

suffix

    Name suffix to identify this descriptor

Returns

**descriptor**

        The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list,  
                        input\_dict: dict, \*\*kwargs) → None

Compute the statistics (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

    data\_coord

        The coordinates. Can be generated by deepmd.model.make\_stat\_input

    data\_box

        The box. Can be generated by deepmd.model.make\_stat\_input

    data\_atype

        The atom types. Can be generated by deepmd.model.make\_stat\_input

    natoms\_vec

        The vector for the number of atoms of the system and different types of atoms.  
        Can be generated by deepmd.model.make\_stat\_input

    mesh

        The mesh for neighbor searching. Can be generated by  
        deepmd.model.make\_stat\_input

    input\_dict

        Dictionary for additional input

    \*\*kwargs

        Additional keyword arguments.

**enable\_compression**(min\_nbor\_dist: float, graph: Graph, graph\_def: GraphDef, table\_extrapolate:  
                        float = 5, table\_stride\_1: float = 0.01, table\_stride\_2: float = 0.1,  
                        check\_frequency: int = -1, suffix: str = '') → None

Reveive the statistics (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

    min\_nbor\_dist

        The nearest distance between atoms

    graph

        [tf.Graph] The graph of the model

    graph\_def

        [tf.GraphDef] The graph\_def of the model

`table_extrapolate`  
 The scale of model extrapolation

`table_stride_1`  
 The uniform stride of the first table

`table_stride_2`  
 The uniform stride of the second table

`check_frequency`  
 The overflow check frequency

`suffix`  
 [`str`, `optional`] The suffix of the scope

`enable_mixed_precision`(mixed\_prec: `Optional[dict]` = None) → `None`  
 Reveal the mixed precision setting.

Parameters

`mixed_prec`  
 The mixed precision setting used in the embedding net

`property explicit_ntypes: bool`  
 Explicit ntypes with type embedding.

`get_dim_out()` → `int`  
 Returns the output dimension of this descriptor.

`get_dim_rot_mat_1()` → `int`  
 Returns the first dimension of the rotation matrix. The rotation is of shape dim\_1 x 3.

`get_nlist()` → `Tuple[Tensor, Tensor, List[int], List[int]]`  
 Returns neighbor information.

Returns

`nlist`  
 Neighbor list

`rij`  
 The relative distance between the neighbor and the center atom.

`sel_a`  
 The number of neighbors with full information

`sel_r`  
 The number of neighbors with only radial information

`get_ntypes()` → `int`  
 Returns the number of atom types.

`get_rcut()` → `float`  
 Returns the cut-off radius.

`get_rot_mat()` → `Tensor`  
 Get rotational matrix.

`init_variables`(graph: `Graph`, graph\_def: `GraphDef`, suffix: `str` = "") → `None`  
 Init the embedding net variables with the given dict.

Parameters

**graph**  
[`tf.Graph`] The input frozen model graph

**graph\_def**  
[`tf.GraphDef`] The input frozen model graph\_def

**suffix**  
[`str`, optional] The suffix of the scope

**merge\_input\_stats**(stat\_dict)

Merge the statistics computed from compute\_input\_stats to obtain the self.davg and self.dstd.

Parameters

**stat\_dict**  
The dict of statistics computed from compute\_input\_stats, including:

**sumr**  
The sum of radial statistics.

**suma**  
The sum of relative coord statistics.

**sumn**  
The sum of neighbor numbers.

**sumr2**  
The sum of square of radial statistics.

**suma2**  
The sum of square of relative coord statistics.

**prod\_force\_virial**(atom\_ener: Tensor, natoms: Tensor) → `Tuple`[Tensor, Tensor, Tensor]

Compute force and virial.

Parameters

**atom\_ener**  
The atomic energy

**natoms**  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

Returns

**force**  
The force on atoms

**virial**  
The total virial

**atom\_virial**  
The atomic virial



**deepmd.descriptor.se\_a\_ebd module**

```
class deepmd.descriptor.se_a_ebd.DescriptSeAEbd(*args, **kwargs)
```

Bases: *DescriptSeA*

DeepPot-SE descriptor with type embedding approach.

Parameters

rcut

The cut-off radius

rcut\_smth

From where the environment matrix should be smoothed

sel

`[list[str]]` sel[i] specifies the maximum number of type i atoms in the cut-off radius

neuron

`[list[int]]` Number of neurons in each hidden layers of the embedding net

axis\_neuron

Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

resnet\_dt

Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

trainable

If the weights of embedding net are trainable.

seed

Random seed for initializing the network parameters.

type\_one\_side

Try to build  $N_{types}$  embedding nets. Otherwise, building  $N_{types}^2$  embedding nets

type\_nchanl

Number of channels for type representation

type\_nlayer

Number of hidden layers for the type embedding net (skip connected).

numb\_aparam

Number of atomic parameters. If  $>0$  it will be embedded with atom types.

set\_davg\_zero

Set the shift of embedding net input to zero.

activation\_function

The activation function in the embedding net. Supported options are {0}

precision

The precision of the embedding net parameters. Supported options are {1}

exclude\_types

`[List[List[int]]]` The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.

**Methods**

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_rot_mat</i> ()	Get rotational matrix.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (descript_reshape, ...)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

**Parameters****coord\_**

The coordinate of atoms

**atype\_**

The type of atoms

**natoms**

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number

of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  
 $2 \leq i < Ntypes+2$ , number of type  $i$  atoms

`box_`  
`[tf.Tensor]` The box of the system

`mesh`  
 For historical reasons, only the length of the Tensor matters. if size of `mesh == 6`,  
 pbc is assumed. if size of `mesh == 0`, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

**descriptor**  
 The output descriptor

#### `deepmd.descriptor.se_a_ebd_v2` module

`class deepmd.descriptor.se_a_ebd_v2.DescriptSeAEbdV2(*args, **kwargs)`

Bases: *DescriptSeA*

A compressible `se_a_ebd` model.

This model is a warpper for `DescriptorSeA`, which set `stripped_type_embedding=True`.

Attributes

**explicit\_ntypes**  
 Explicit ntypes with type embedding.

**precision**  
 Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from <code>compute_input_stats</code> to obtain the <code>self.davg</code> and <code>self.dstd</code> .
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the <code>descrpt_reshape</code> tensor as well as <code>descrpt_deriv</code> tensor from the <code>frz graph_def</code> .
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_class_by_input</code>	
---------------------------------	--

## deepmd.descriptor.se\_a\_ef module

```
class deepmd.descriptor.se_a_ef.DescriptSeAEf(*args, **kwargs)
```

Bases: *DescriptSe*

Smooth edition descriptor with Ef.

Parameters

`rcut`

The cut-off radius

`rcut_smth`

From where the environment matrix should be smoothed

`sel`

`[list[str]]` `sel[i]` specifies the maximum number of type `i` atoms in the cut-off radius

`neuron`

`[list[int]]` Number of neurons in each hidden layers of the embedding net

**axis\_neuron**  
 Number of the axis neuron (number of columns of the sub-matrix of the embedding matrix)

**resnet\_dt**  
 Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**  
 If the weights of embedding net are trainable.

**seed**  
 Random seed for initializing the network parameters.

**type\_one\_side**  
 Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

**exclude\_types**  
`[List[List[int]]]` The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

**set\_davg\_zero**  
 Set the shift of embedding net input to zero.

**activation\_function**  
 The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
 The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
 Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

#### Attributes

**explicit\_ntypes**  
 Explicit ntypes with type embedding.

**precision**  
 Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

Parameters

`coord_`  
The coordinate of atoms

`atype_`  
The type of atoms

`natoms`  
The number of atoms. This tensor has the length of Ntypes + 2 `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < \text{Ntypes} + 2$ , number of type  $i$  atoms

`box_`  
[`tf.Tensor`] The box of the system

`mesh`  
For historical reasons, only the length of the Tensor matters. if size of `mesh` == 6, pbc is assumed. if size of `mesh` == 0, no-pbc is assumed.

`input_dict`  
Dictionary for additional inputs. Should have 'efield'.

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

**descriptor**

The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

data\_coord

The coordinates. Can be generated by deepmd.model.make\_stat\_input

data\_box

The box. Can be generated by deepmd.model.make\_stat\_input

data\_atype

The atom types. Can be generated by deepmd.model.make\_stat\_input

natoms\_vec

The vector for the number of atoms of the system and different types of atoms.  
Can be generated by deepmd.model.make\_stat\_input

mesh

The mesh for neighbor searching. Can be generated by  
deepmd.model.make\_stat\_input

input\_dict

Dictionary for additional input

\*\*kwargs

Additional keyword arguments.

**get\_dim\_out()** → int

Returns the output dimension of this descriptor.

**get\_dim\_rot\_mat\_1()** → int

Returns the first dimension of the rotation matrix. The rotation is of shape dim\_1 x 3.

**get\_nlist()** → Tuple[Tensor, Tensor, List[int], List[int]]

Returns neighbor information.

Returns

**nlist**

Neighbor list

**rij**

The relative distance between the neighbor and the center atom.

**sel\_a**

The number of neighbors with full information

**sel\_r**

The number of neighbors with only radial information

**get\_ntypes()** → *int*

Returns the number of atom types.

**get\_rcut()** → *float*

Returns the cut-off radius.

**get\_rot\_mat()** → *Tensor*

Get rotational matrix.

**prod\_force\_virial**(atom\_ener: *Tensor*, natoms: *Tensor*) → *Tuple*[*Tensor*, *Tensor*, *Tensor*]

Compute force and virial.

Parameters

**atom\_ener**

The atomic energy

**natoms**

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

Returns

**force**

The force on atoms

**virial**

The total virial

**atom\_virial**

The atomic virial

**class** deepmd.descriptor.se\_a\_ef.DescriptSeAEfLower(\*args, \*\*kwargs)

Bases: *DescriptSeA*

Helper class for implementing DescriptSeAEf.

Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.



## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statisitcs (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statisitcs (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statisitcs computed from compute_input_stats to obtain the self.davg and self.dstd.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get\_class\_by\_input

**build**(coord\_, atype\_, natoms, box\_, mesh, input\_dict, suffix="", reuse=None)

Build the computational graph for the descriptor.

Parameters

coord\_

The coordinate of atoms

atype\_

The type of atoms

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

box\_

[[tf.Tensor](#)] The box of the system

mesh

For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

**descriptor**  
 The output descriptor

**compute\_input\_stats**(data\_coord, data\_box, data\_atype, natoms\_vec, mesh, input\_dict, \*\*kwargs)  
 Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
 The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`  
 The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`  
 The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`  
 The vector for the number of atoms of the system and different types of atoms.  
 Can be generated by `deepmd.model.make_stat_input`

`mesh`  
 The mesh for neighbor searching. Can be generated by  
`deepmd.model.make_stat_input`

`input_dict`  
 Dictionary for additional input

`**kwargs`  
 Additional keyword arguments.

## deepmd.descriptor.se\_a\_mask module

**class** `deepmd.descriptor.se_a_mask.DescriptSeAMask`(\*args, \*\*kwargs)

Bases: *DescriptSeA*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor  $\mathcal{D}^i \in \mathbb{R}^{M_1 \times M_2}$  is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i$$

where  $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$  is the coordinate matrix, and each row of  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{r_{ji}} \\ \frac{s(r_{ji})x_{ji}}{r_{ji}^2} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}^2} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}^2} \end{bmatrix}$$

where  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$  consists of outputs of a embedding network  $\mathcal{N}$  of  $s(r_{ji})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_{<} \in \mathbb{R}^{N \times M_2}$  takes first  $M_2$  columns of  $\mathcal{G}^i$ . The equation of embedding network  $\mathcal{N}$  can be found at `deepmd.utils.network.embedding_net()`. Specially for descriptor `se_a_mask` is a concise implementation of `se_a`. The difference is that `se_a_mask` only considered a non-pbc system. And accept a mask matrix to indicate the atom  $i$  in frame  $j$  is a real atom or not. (1 means real atom, 0 means ghost atom) Thus `se_a_mask` can accept a variable number of atoms in a frame.

Parameters

- `sel`  
[`list[str]`] `sel[i]` specifies the maximum number of type  $i$  atoms in the neighbor list.
- `neuron`  
[`list[int]`] Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$
- `axis_neuron`  
Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)
- `resnet_dt`  
Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$
- `trainable`  
If the weights of embedding net are trainable.
- `seed`  
Random seed for initializing the network parameters.
- `type_one_side`  
Try to build  $N_{types}$  embedding nets. Otherwise, building  $N_{types}^2$  embedding nets
- `exclude_types`  
[`List[List[int]]`] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.
- `activation_function`  
The activation function in the embedding net. Supported options are {0}
- `precision`  
The precision of the embedding net parameters. Supported options are {1}
- `uniform_seed`  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

## References

[1]

## Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.

## Methods

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cutoff radius.
<i>get_rot_mat</i> ()	Get rotational matrix.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (descript_reshape, ...)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: Dict[str, Any], reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

## Parameters

coord\_

The coordinate of atoms

**atype\_**  
 The type of atoms  
**natoms**  
 The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms  
**box\_**  
 [`tf.Tensor`] The box of the system  
**mesh**  
 For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.  
**input\_dict**  
 Dictionary for additional inputs  
**reuse**  
 The weights in the networks should be reused when get the variable.  
**suffix**  
 Name suffix to identify this descriptor

Returns

**descriptor**  
 The output descriptor

**compute\_input\_stats**(data\_coord: list, data\_box: list, data\_atype: list, natoms\_vec: list, mesh: list, input\_dict: dict, \*\*kwargs) → None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

**data\_coord**  
 The coordinates. Can be generated by `deepmd.model.make_stat_input`  
**data\_box**  
 The box. Can be generated by `deepmd.model.make_stat_input`  
**data\_atype**  
 The atom types. Can be generated by `deepmd.model.make_stat_input`  
**natoms\_vec**  
 The vector for the number of atoms of the system and different types of atoms. Can be generated by `deepmd.model.make_stat_input`  
**mesh**  
 The mesh for neighbor searching. Can be generated by `deepmd.model.make_stat_input`  
**input\_dict**  
 Dictionary for additional input  
**\*\*kwargs**  
 Additional keyword arguments.

**get\_rcut**() → float

Returns the cutoff radius.

**prod\_force\_virial**(atom\_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]

Compute force and virial.

Parameters

atom\_ener

The atomic energy

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

Returns

**force**

The force on atoms

**virial**

None for se\_a\_mask op

**atom\_virial**

None for se\_a\_mask op

**classmethod update\_sel**(global\_jdata: dict, local\_jdata: dict)

Update the selection and perform neighbor statistics.

Parameters

global\_jdata

[dict] The global data, containing the training section

local\_jdata

[dict] The local data refer to the current class

### deepmd.descriptor.se\_atten module

**class** deepmd.descriptor.se\_atten.DescriptSeAtten(\*args, \*\*kwargs)

Bases: *DescriptSeA*

Smooth version descriptor with attention.

Parameters

rcut

The cut-off radius  $r_c$

rcut\_smth

From where the environment matrix should be smoothed  $r_s$

sel

[list[str]] sel[i] specifies the maximum number of type i atoms in the cut-off radius

neuron

[list[int]] Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

axis\_neuron

Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

resnet\_dt

Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**  
If the weights of embedding net are trainable.

**seed**  
Random seed for initializing the network parameters.

**type\_one\_side**  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

**exclude\_types**  
`[List[List[int]]]` The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

**set\_davg\_zero**  
Set the shift of embedding net input to zero.

**activation\_function**  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

**attn**  
The length of hidden vector during scale-dot attention computation.

**attn\_layer**  
The number of layers in attention mechanism.

**attn\_dotr**  
Whether to dot the relative coordinates on the attention weights as a gated scheme.

**attn\_mask**  
Whether to mask the diagonal in the attention weights.

**multi\_task**  
If the model has multi fitting nets to train.

**stripped\_type\_embedding**  
Whether to strip the type embedding into a separated embedding network. Default value will be True in `se_atten_v2` descriptor.

**smooth\_type\_embdding**  
When using stripped type embedding, whether to dot smooth factor on the network output of type embedding to keep the network smooth, instead of setting `set_davg_zero` to be True. Default value will be True in `se_atten_v2` descriptor.

**Raises**

**ValueError**  
if `ntypes` is 0.

**Attributes**

**explicit\_ntypes**  
Explicit `ntypes` with type embedding.

**precision**

Precision of filter network.

**Methods**

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the attention descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_rot_mat</i> ()	Get rotational matrix.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (descript_reshape, ...)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

**Parameters**

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms



`box_`  
`[tf.Tensor]` The box of the system

`mesh`  
 For historical reasons, only the length of the Tensor matters. if size of `mesh == 6`, pbc is assumed. if size of `mesh == 0`, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

**`descriptor`**  
 The output descriptor

**`build_type_exclude_mask`**(`exclude_types`: `List[Tuple[int, int]]`, `ntypes`: `int`, `sel`: `List[int]`, `ndescript`: `int`, `atype`: `Tensor`, `shape0`: `Tensor`, `nei_type_vec`: `Tensor`) → `Tensor`

Build the type exclude mask for the attention descriptor.

Parameters

`exclude_types`  
`[List[Tuple[int, int]]]` The list of excluded types, e.g. `[(0, 1), (1, 0)]` means the interaction between type 0 and type 1 is excluded.

`ntypes`  
`[int]` The number of types.

`sel`  
`[List[int]]` The list of the number of selected neighbors for each type.

`ndescript`  
`[int]` The number of descriptors for each atom.

`atype`  
`[tf.Tensor]` The type of atoms, with the size of `shape0`.

`shape0`  
`[tf.Tensor]` The shape of the first dimension of the inputs, which is equal to `nsamples * natoms`.

`nei_type_vec`  
`[tf.Tensor]` The type of neighbors, with the size of `(shape0, nnei)`.

Returns

**`tf.Tensor`**  
 The type exclude mask, with the shape of `(shape0, ndescript)`, and the precision of `GLOBAL_TF_FLOAT_PRECISION`. The mask has the value of 1 if the interaction between two types is not excluded, and 0 otherwise.

See also:

`deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask`

## Notes

This method has the similar way to build the type exclude mask as `deepmd.descriptor.descriptor.Descriptor.build_type_exclude_mask()`. The mathematical expression has been explained in that method. The difference is that the attention descriptor has provided the type of the neighbors (`idx_j`) that is not in order, so we use it from an extra input.

```
compute_input_stats(data_coord: list, data_box: list, data_atype: list, natoms_vec: list, mesh: list,
                    input_dict: dict, mixed_type: bool = False, real_natoms_vec: Optional[list] =
                    None, **kwargs) → None
```

Compute the statistics (avg and std) of the training data. The input will be normalized by the statistics.

### Parameters

`data_coord`

The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`

The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`

The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`

The vector for the number of atoms of the system and different types of atoms. If `mixed_type` is True, this para is blank. See `real_natoms_vec`.

`mesh`

The mesh for neighbor searching. Can be generated by `deepmd.model.make_stat_input`

`input_dict`

Dictionary for additional input

`mixed_type`

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

`real_natoms_vec`

If `mixed_type` is True, it takes in the real `natoms_vec` for each frame.

`**kwargs`

Additional keyword arguments.

```
enable_compression(min_nbor_dist: float, graph: Graph, graph_def: GraphDef, table_extrapolate:
                    float = 5, table_stride_1: float = 0.01, table_stride_2: float = 0.1,
                    check_frequency: int = -1, suffix: str = '') → None
```

Reveive the statistics (distance, max\_nbor\_size and env\_mat\_range) of the training data.

### Parameters

`min_nbor_dist`

The nearest distance between atoms

`graph`

[`tf.Graph`] The graph of the model

`graph_def`

[`tf.GraphDef`] The `graph_def` of the model

```

    table_extrapolate
        The scale of model extrapolation
    table_stride_1
        The uniform stride of the first table
    table_stride_2
        The uniform stride of the second table
    check_frequency
        The overflow check frequency
    suffix
        [str, optional] The suffix of the scope
property explicit_ntypes: bool
    Explicit ntypes with type embedding.
init_variables(graph: Graph, graph_def: GraphDef, suffix: str = "") → None
    Init the embedding net variables with the given dict.
Parameters
    graph
        [tf.Graph] The input frozen model graph
    graph_def
        [tf.GraphDef] The input frozen model graph_def
    suffix
        [str, optional] The suffix of the scope
classmethod update_sel(global_jdata: dict, local_jdata: dict)
    Update the selection and perform neighbor statistics.
Parameters
    global_jdata
        [dict] The global data, containing the training section
    local_jdata
        [dict] The local data refer to the current class

```

### deepmd.descriptor.se\_atten\_v2 module

```

class deepmd.descriptor.se_atten_v2.DescriptSeAttenV2(*args, **kwargs)
    Bases: DescriptSeAtten
    Smooth version 2.0 descriptor with attention.
Parameters
    rcut
        The cut-off radius  $r_c$ 
    rcut_smth
        From where the environment matrix should be smoothed  $r_s$ 
    sel
        [list[str]] sel[i] specifies the maximum number of type i atoms in the cut-off radius

```

**neuron**  
[`list[int]`] Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

**axis\_neuron**  
Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

**resnet\_dt**  
Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**  
If the weights of embedding net are trainable.

**seed**  
Random seed for initializing the network parameters.

**type\_one\_side**  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

**exclude\_types**  
[`List[List[int]]`] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

**set\_davg\_zero**  
Set the shift of embedding net input to zero.

**activation\_function**  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

**attn**  
The length of hidden vector during scale-dot attention computation.

**attn\_layer**  
The number of layers in attention mechanism.

**attn\_dotr**  
Whether to dot the relative coordinates on the attention weights as a gated scheme.

**attn\_mask**  
Whether to mask the diagonal in the attention weights.

**multi\_task**  
If the model has multi fitting nets to train.

Attributes

**explicit\_ntypes**  
Explicit ntypes with type embedding.

**precision**  
Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the attention descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_rot_mat()</code>	Get rotational matrix.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from <code>compute_input_stats</code> to obtain the <code>self.davg</code> and <code>self.dstd</code> .
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the <code>descrpt_reshape</code> tensor as well as <code>descrpt_deriv</code> tensor from the <code>frz_graph_def</code> .
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get\_class\_by\_input

## deepmd.descriptor.se\_r module

`class deepmd.descriptor.se_r.DescriptSeR(*args, **kwargs)`

Bases: *DescriptSe*

DeepPot-SE constructed from radial information of atomic configurations.

The embedding takes the distance between atoms as input.

Parameters

`rcut`

The cut-off radius

`rcut_smth`

From where the environment matrix should be smoothed

`sel`

`[list[str]] sel[i]` specifies the maximum number of type `i` atoms in the cut-off radius

**neuron**  
[`list[int]`] Number of neurons in each hidden layers of the embedding net

**resnet\_dt**  
Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**  
If the weights of embedding net are trainable.

**seed**  
Random seed for initializing the network parameters.

**type\_one\_side**  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

**exclude\_types**  
[`List[List[int]]`] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

**activation\_function**  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**explicit\_ntypes**  
Explicit ntypes with type embedding.

**precision**  
Precision of filter network.

## Methods

<code>build(coord_, atype_, natoms, box_, mesh, ...)</code>	Build the computational graph for the descriptor.
<code>build_type_exclude_mask(exclude_types, ...)</code>	Build the type exclude mask for the descriptor.
<code>compute_input_stats(data_coord, data_box, ...)</code>	Compute the statistics (avg and std) of the training data.
<code>enable_compression(min_nbor_dist, graph, ...)</code>	Receive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<code>enable_mixed_precision([mixed_prec])</code>	Receive the mixed precision setting.
<code>get_dim_out()</code>	Returns the output dimension of this descriptor.
<code>get_dim_rot_mat_1()</code>	Returns the first dimension of the rotation matrix.
<code>get_nlist()</code>	Returns neighbor information.
<code>get_ntypes()</code>	Returns the number of atom types.
<code>get_rcut()</code>	Returns the cut-off radius.
<code>get_tensor_names([suffix])</code>	Get names of tensors.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the embedding net variables with the given dict.
<code>merge_input_stats(stat_dict)</code>	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<code>pass_tensors_from_frz_model(descrpt_reshape, ...)</code>	Pass the descrpt_reshape tensor as well as descrpt_deriv tensor from the frz graph_def.
<code>prod_force_virial(atom_ener, natoms)</code>	Compute force and virial.
<code>register(key)</code>	Register a descriptor plugin.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = "") → Tensor

Build the computational graph for the descriptor.

Parameters

coord\_  
The coordinate of atoms

atype\_  
The type of atoms

natoms  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box\_  
[tf.Tensor] The box of the system

mesh  
For historical reasons, only the length of the Tensor matters. if size of mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

**descriptor**  
 The output descriptor

**compute\_input\_stats**(data\_coord, data\_box, data\_atype, natoms\_vec, mesh, input\_dict, \*\*kwargs)  
 Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
 The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`  
 The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`  
 The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`  
 The vector for the number of atoms of the system and different types of atoms.  
 Can be generated by `deepmd.model.make_stat_input`

`mesh`  
 The mesh for neighbor searching. Can be generated by  
`deepmd.model.make_stat_input`

`input_dict`  
 Dictionary for additional input

`**kwargs`  
 Additional keyword arguments.

**enable\_compression**(min\_nbor\_dist: float, graph: Graph, graph\_def: GraphDef, table\_extrapolate:  
 float = 5, table\_stride\_1: float = 0.01, table\_stride\_2: float = 0.1,  
 check\_frequency: int = -1, suffix: str = '') → None

Reveive the statisitcs (distance, max\_nbor\_size and env\_mat\_range) of the training data.

Parameters

`min_nbor_dist`  
 The nearest distance between atoms

`graph`  
 [tf.Graph] The graph of the model

`graph_def`  
 [tf.GraphDef] The graph\_def of the model

`table_extrapolate`  
 The scale of model extrapolation



`table_stride_1`  
 The uniform stride of the first table  
`table_stride_2`  
 The uniform stride of the second table  
`check_frequency`  
 The overflow check frequency  
`suffix`  
 [`str`, `optional`] The suffix of the scope  
**`get_dim_out()`**  
 Returns the output dimension of this descriptor.  
**`get_nlist()`**  
 Returns neighbor information.  
 Returns  
     **`nlist`**  
     Neighbor list  
     **`rij`**  
     The relative distance between the neighbor and the center atom.  
     **`sel_a`**  
     The number of neighbors with full information  
     **`sel_r`**  
     The number of neighbors with only radial information  
**`get_ntypes()`**  
 Returns the number of atom types.  
**`get_rcut()`**  
 Returns the cut-off radius.  
**`merge_input_stats(stat_dict)`**  
 Merge the statisitcs computed from `compute_input_stats` to obtain the `self.davg` and `self.dstd`.  
 Parameters  
     `stat_dict`  
     The dict of statisitcs computed from `compute_input_stats`, including:  
         **`sumr`**  
         The sum of radial statisitcs.  
         **`sumn`**  
         The sum of neighbor numbers.  
         **`sumr2`**  
         The sum of square of radial statisitcs.  
**`prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]`**  
 Compute force and virial.  
 Parameters  
     `atom_ener`  
     The atomic energy

**natoms**  
The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  $2 \leq i < \text{Ntypes} + 2$ , number of type i atoms

Returns

**force**  
The force on atoms

**virial**  
The total virial

**atom\_virial**  
The atomic virial

### deepmd.descriptor.se\_t module

**class** deepmd.descriptor.se\_t.DescriptSeT(\*args, \*\*kwargs)

Bases: *DescriptSe*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations.

The embedding takes angles between two neighboring atoms as input.

Parameters

**rcut**  
The cut-off radius

**rcut\_smth**  
From where the environment matrix should be smoothed

**sel**  
[*list*[*str*]] sel[i] specifies the maximum number of type i atoms in the cut-off radius

**neuron**  
[*list*[*int*]] Number of neurons in each hidden layers of the embedding net

**resnet\_dt**  
Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**trainable**  
If the weights of embedding net are trainable.

**seed**  
Random seed for initializing the network parameters.

**set\_davg\_zero**  
Set the shift of embedding net input to zero.

**activation\_function**  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

## Attributes

**explicit\_ntypes**

Explicit ntypes with type embedding.

**precision**

Precision of filter network.

## Methods

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the computational graph for the descriptor.
<i>build_type_exclude_mask</i> (exclude_types, ...)	Build the type exclude mask for the descriptor.
<i>compute_input_stats</i> (data_coord, data_box, ...)	Compute the statistics (avg and std) of the training data.
<i>enable_compression</i> (min_nbor_dist, graph, ...)	Reveive the statistics (distance, max_nbor_size and env_mat_range) of the training data.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_dim_out</i> ()	Returns the output dimension of this descriptor.
<i>get_dim_rot_mat_1</i> ()	Returns the first dimension of the rotation matrix.
<i>get_nlist</i> ()	Returns neighbor information.
<i>get_ntypes</i> ()	Returns the number of atom types.
<i>get_rcut</i> ()	Returns the cut-off radius.
<i>get_tensor_names</i> ([suffix])	Get names of tensors.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the embedding net variables with the given dict.
<i>merge_input_stats</i> (stat_dict)	Merge the statistics computed from compute_input_stats to obtain the self.davg and self.dstd.
<i>pass_tensors_from_frz_model</i> (descript_reshape, ...)	Pass the descript_reshape tensor as well as descript_deriv tensor from the frz_graph_def.
<i>prod_force_virial</i> (atom_ener, natoms)	Compute force and virial.
<i>register</i> (key)	Register a descriptor plugin.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_class_by_input	
--------------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for the descriptor.

## Parameters

coord\_

The coordinate of atoms

atype\_

The type of atoms

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number

of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  
 $2 \leq i < Ntypes+2$ , number of type  $i$  atoms

`box_`  
`[tf.Tensor]` The box of the system

`mesh`  
 For historical reasons, only the length of the Tensor matters. if size of `mesh == 6`,  
 pbc is assumed. if size of `mesh == 0`, no-pbc is assumed.

`input_dict`  
 Dictionary for additional inputs

`reuse`  
 The weights in the networks should be reused when get the variable.

`suffix`  
 Name suffix to identify this descriptor

Returns

**descriptor**  
 The output descriptor

**compute\_input\_stats**(`data_coord`: list, `data_box`: list, `data_atype`: list, `natoms_vec`: list, `mesh`: list,  
`input_dict`: dict, **\*\*kwargs**)  $\rightarrow$  None

Compute the statisitcs (avg and std) of the training data. The input will be normalized by the statistics.

Parameters

`data_coord`  
 The coordinates. Can be generated by `deepmd.model.make_stat_input`

`data_box`  
 The box. Can be generated by `deepmd.model.make_stat_input`

`data_atype`  
 The atom types. Can be generated by `deepmd.model.make_stat_input`

`natoms_vec`  
 The vector for the number of atoms of the system and different types of atoms.  
 Can be generated by `deepmd.model.make_stat_input`

`mesh`  
 The mesh for neighbor searching. Can be generated by  
`deepmd.model.make_stat_input`

`input_dict`  
 Dictionary for additional input

**\*\*kwargs**  
 Additional keyword arguments.

**enable\_compression**(`min_nbor_dist`: float, `graph`: Graph, `graph_def`: GraphDef, `table_extrapolate`:  
float = 5, `table_stride_1`: float = 0.01, `table_stride_2`: float = 0.1,  
`check_frequency`: int = -1, `suffix`: str = '')  $\rightarrow$  None

Reveive the statisitcs (distance, `max_nbor_size` and `env_mat_range`) of the training data.

Parameters

`min_nbor_dist`  
 The nearest distance between atoms

`graph`  
`[tf.Graph]` The graph of the model

`graph_def`  
`[tf.GraphDef]` The graph\_def of the model

`table_extrapolate`  
 The scale of model extrapolation

`table_stride_1`  
 The uniform stride of the first table

`table_stride_2`  
 The uniform stride of the second table

`check_frequency`  
 The overflow check frequency

`suffix`  
`[str, optional]` The suffix of the scope

`get_dim_out() → int`  
 Returns the output dimension of this descriptor.

`get_nlist() → Tuple[Tensor, Tensor, List[int], List[int]]`  
 Returns neighbor information.  
 Returns
 

- `nlist`  
Neighbor list
- `rij`  
The relative distance between the neighbor and the center atom.
- `sel_a`  
The number of neighbors with full information
- `sel_r`  
The number of neighbors with only radial information

`get_ntypes() → int`  
 Returns the number of atom types.

`get_rcut() → float`  
 Returns the cut-off radius.

`merge_input_stats(stat_dict)`  
 Merge the statisitcs computed from compute\_input\_stats to obtain the self.davg and self.dstd.  
 Parameters
 

- `stat_dict`  
The dict of statisitcs computed from compute\_input\_stats, including:
  - `sumr`  
The sum of radial statisitcs.
  - `suma`  
The sum of relative coord statisitcs.

`sumn`  
The sum of neighbor numbers.

`sumr2`  
The sum of square of radial statisits.

`suma2`  
The sum of square of relative coord statisits.

`prod_force_virial(atom_ener: Tensor, natoms: Tensor) → Tuple[Tensor, Tensor, Tensor]`

Compute force and virial.

Parameters

`atom_ener`  
The atomic energy

`natoms`  
The number of atoms. This tensor has the length of Ntypes + 2  
`natoms[0]`: number of local atoms  
`natoms[1]`: total number of atoms held by this processor  
`natoms[i]`:  $2 \leq i < \text{Ntypes} + 2$ , number of type  $i$  atoms

Returns

`force`  
The force on atoms

`virial`  
The total virial

`atom_virial`  
The atomic virial

## deepmd.entrypoints package

Submodule that contains all the DeePMD-Kit entry point scripts.

`deepmd.entrypoints.compress(*, input: str, output: str, extrapolate: int, step: float, frequency: str, checkpoint_folder: str, training_script: str, mpi_log: str, log_path: Optional[str], log_level: int, **kwargs)`

Compress model.

The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. The first table takes the step parameter as the domain's uniform step size, while the second table takes  $10 * \text{step}$  as it's uniform step size. The range of the first table is automatically detected by the code, while the second table ranges from the first table's upper boundary(`upper`) to the `extrapolate(parameter) * upper`.

Parameters

`input`  
[`str`] frozen model file to compress

`output`  
[`str`] compressed model filename

`extrapolate`  
[`int`] scale of model extrapolation

`step`  
[`float`] uniform step size of the tabulation's first table

```

frequency
    [str] frequency of tabulation overflow check
checkpoint_folder
    [str] trining checkpoint folder for freezing
training_script
    [str] training script of the input frozen model
mpi_log
    [str] mpi logging mode for training
log_path
    [Optional[str]] if speccified log will be written to this file
log_level
    [int] logging level
**kwargs
    additional arguments
deepmd.entrypoints.convert(*, FROM: str, input_model: str, output_model: str, **kwargs)
deepmd.entrypoints.doc_train_input(*, out_type: str = 'rst', **kwargs)
    Print out trining input arguments to console.
deepmd.entrypoints.freeze(*, checkpoint_folder: str, output: str, node_names: Optional[str] = None,
                           nvnmmd_weight: Optional[str] = None, united_model: bool = False,
                           **kwargs)
    Freeze the graph in supplied folder.
    Parameters
        checkpoint_folder
            [str] location of the folder with model
        output
            [str] output file name
        node_names
            [Optional[str], optional] names of nodes to output, by default None
        nvnmmd_weight
            [Optional[str], optional] nvnmmd weight file
        united_model
            [bool] when in multi-task mode, freeze all nodes into one unit model
        **kwargs
            other arguments
deepmd.entrypoints.make_model_devi(*, models: list, system: str, set_prefix: str, output: str, frequency:
                                   int, real_error: bool = False, atomic: bool = False, relative:
                                   Optional[float] = None, relative_v: Optional[float] = None,
                                   **kwargs)
    Make model deviation calculation.
    Parameters
        models
            [list] A list of paths of models to use for making model deviation

```

system  
    [str] The path of system to make model deviation calculation

set\_prefix  
    [str] The set prefix of the system

output  
    [str] The output file for model deviation results

frequency  
    [int] The number of steps that elapse between writing coordinates in a trajectory by a MD engine (such as Gromacs / Lammps). This paramter is used to determine the index in the output file.

real\_error  
    [bool, default: False] If True, calculate the RMS real error instead of model deviation.

atomic  
    [bool, default: False] If True, calculate the force model deviation of each atom.

relative  
    [float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

relative\_v  
    [float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

\*\*kwargs  
    Arbitrary keyword arguments.

```
deepmd.entrypoints.neighbor_stat(*, system: str, rcut: float, type_map: List[str], one_type: bool = False, **kwargs)
```

Calculate neighbor statistics.

Parameters

system  
    [str] system to stat

rcut  
    [float] cutoff radius

type\_map  
    [list[str]] type map

one\_type  
    [bool, optional, default=False] treat all types as a single type

\*\*kwargs  
    additional arguments



## Examples

```
>>> neighbor_stat(system='.', rcut=6., type_map=["C", "H", "O", "N", "P", "S", "Mg", "Na", "HW
↳", "OW", "mNa", "mCl", "mC", "mH", "mMg", "mN", "mO", "mP"])
min_nbor_dist: 0.6599510670195264
max_nbor_size: [23, 26, 19, 16, 2, 2, 1, 1, 72, 37, 5, 0, 31, 29, 1, 21, 20, 5]
```

`deepmd.entrypoints.start_dpgui(*, port: int, bind_all: bool, **kwargs)`

Host DP-GUI server.

Parameters

port

[int] The port to serve DP-GUI on.

bind\_all

[bool] Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

\*\*kwargs

additional arguments

Raises

`ModuleNotFoundError`

The dpgui package is not installed

`deepmd.entrypoints.test(*, model: str, system: str, datafile: str, set_prefix: str, numb_test: int, rand_seed: Optional[int], shuffle_test: bool, detail_file: str, atomic: bool, **kwargs)`

Test model predictions.

Parameters

model

[str] path where model is stored

system

[str] system directory

datafile

[str] the path to the list of systems to test

set\_prefix

[str] string prefix of set

numb\_test

[int] munber of tests to do. 0 means all data.

rand\_seed

[Optional[int]] seed for random generator

shuffle\_test

[bool] whether to shuffle tests

detail\_file

[Optional[str]] file where test details will be output

atomic

[bool] whether per atom quantities should be computed

```
    **kwargs
        additional arguments

Raises

    RuntimeError
        if no valid system was found

deepmd.entrypoints.train_dp(*, INPUT: str, init_model: Optional[str], restart: Optional[str], output:
    str, init_frz_model: str, mpi_log: str, log_level: int, log_path:
    Optional[str], is_compress: bool = False, skip_neighbor_stat: bool =
    False, finetune: Optional[str] = None, **kwargs)
```

Run DeePMD model training.

Parameters

```
INPUT
    [str] json/yaml control file

init_model
    [Optional[str]] path prefix of checkpoint files or None

restart
    [Optional[str]] path prefix of checkpoint files or None

output
    [str] path for dump file with arguments

init_frz_model
    [str] path to frozen model or None

mpi_log
    [str] mpi logging mode

log_level
    [int] logging level defined by int 0-3

log_path
    [Optional[str]] logging file path or None if logs are to be output only to stdout

is_compress
    [bool] indicates whether in the model compress mode

skip_neighbor_stat
    [bool, default=False] skip checking neighbor statistics

finetune
    [Optional[str]] path to pretrained model or None

**kwargs
    additional arguments
```

Raises

```
    RuntimeError
        if distributed training job name is wrong
```

```
deepmd.entrypoints.transfer(*, old_model: str, raw_model: str, output: str, **kwargs)
```

Transfer operation from old from graph to new prepared raw graph.

Parameters

```
old_model
    [str] frozen old graph model
```

```

raw_model
    [str] new model that will accept ops from old model

output
    [str] new model with transfered parameters will be saved to this location

**kwargs
    additional arguments

```

## Submodules

### deepmd.entrypoints.compress module

Compress a model, which including tabulating the embedding-net.

```

deepmd.entrypoints.compress.compress(*, input: str, output: str, extrapolate: int, step: float,
                                     frequency: str, checkpoint_folder: str, training_script: str,
                                     mpi_log: str, log_path: Optional[str], log_level: int, **kwargs)

```

Compress model.

The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. The first table takes the step parameter as the domain's uniform step size, while the second table takes 10 \* step as its uniform step size. The range of the first table is automatically detected by the code, while the second table ranges from the first table's upper boundary(upper) to the extrapolate(parameter) \* upper.

Parameters

```

input
    [str] frozen model file to compress

output
    [str] compressed model filename

extrapolate
    [int] scale of model extrapolation

step
    [float] uniform step size of the tabulation's first table

frequency
    [str] frequency of tabulation overflow check

checkpoint_folder
    [str] training checkpoint folder for freezing

training_script
    [str] training script of the input frozen model

mpi_log
    [str] mpi logging mode for training

log_path
    [Optional[str]] if specified log will be written to this file

log_level
    [int] logging level

**kwargs
    additional arguments

```

**deepmd.entrypoints.convert module**

```
deepmd.entrypoints.convert.convert(*, FROM: str, input_model: str, output_model: str, **kwargs)
```

**deepmd.entrypoints.doc module**

```
deepmd.entrypoints.doc.doc_train_input(*, out_type: str = 'rst', **kwargs)
```

Print out training input arguments to console.

**deepmd.entrypoints.freeze module**

Script for freezing TF trained graph so it can be used with LAMMPS and i-PI.

**References**

<https://blog.metaflow.fr/tensorflow-how-to-freeze-a-model-and-serve-it-with-a-python-api-d4f3596b3adc>

```
deepmd.entrypoints.freeze.freeze(*, checkpoint_folder: str, output: str, node_names: Optional[str] =
                                None, nvnmd_weight: Optional[str] = None, united_model: bool =
                                False, **kwargs)
```

Freeze the graph in supplied folder.

Parameters

checkpoint\_folder

[str] location of the folder with model

output

[str] output file name

node\_names

[Optional[str], optional] names of nodes to output, by default None

nvnmd\_weight

[Optional[str], optional] nvnmd weight file

united\_model

[bool] when in multi-task mode, freeze all nodes into one unit model

\*\*kwargs

other arguments

**deepmd.entrypoints.gui module**

```
deepmd.entrypoints.gui.start_dpGui(*, port: int, bind_all: bool, **kwargs)
```

Host DP-GUI server.

Parameters

port

[int] The port to serve DP-GUI on.

`bind_all`  
`[bool]` Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

`**kwargs`  
 additional arguments

Raises

`ModuleNotFoundError`  
 The dpgui package is not installed

### deepmd.entrypoints.ipi module

Use dp\_ipi inside the Python package.

`deepmd.entrypoints.ipi.dp_ipi()`  
`dp_ipi`.

### deepmd.entrypoints.main module

DeePMD-Kit entry point module.

`deepmd.entrypoints.main.get_ll(log_level: str) → int`

Convert string to python logging level.

Parameters

`log_level`  
`[str]` allowed input values are: DEBUG, INFO, WARNING, ERROR, 3, 2, 1, 0

Returns

`int`  
 one of python logging module log levels - 10, 20, 30 or 40

`deepmd.entrypoints.main.main(args: Optional[Union[List[str], Namespace]] = None)`

DeePMD-Kit entry point.

Parameters

`args`  
`[List[str] or argparse.Namespace, optional]` list of command line arguments, used to avoid calling from the subprocess, as it is quite slow to import tensorflow; if Namespace is given, it will be used directly

Raises

`RuntimeError`  
 if no command was input

`deepmd.entrypoints.main.main_parser() → ArgumentParser`

DeePMD-Kit commandline options argument parser.

Returns

`argparse.ArgumentParser`  
 main parser of DeePMD-kit

`deepmd.entrypoints.main.parse_args(args: Optional[List[str]] = None) → Namespace`

Parse arguments and convert argument strings to objects.

Parameters

`args`

`[List[str]]` list of command line arguments, main purpose is testing default option

`None` takes arguments from `sys.argv`

Returns

`argparse.Namespace`

the populated namespace

### `deepmd.entrypoints.neighbor_stat` module

`deepmd.entrypoints.neighbor_stat.neighbor_stat(*, system: str, rcut: float, type_map: List[str], one_type: bool = False, **kwargs)`

Calculate neighbor statistics.

Parameters

`system`

`[str]` system to stat

`rcut`

`[float]` cutoff radius

`type_map`

`[list[str]]` type map

`one_type`

`[bool, optional, default=False]` treat all types as a single type

`**kwargs`

additional arguments

### Examples

```
>>> neighbor_stat(system='.', rcut=6., type_map=["C", "H", "O", "N", "P", "S", "Mg", "Na", "HW", "OW", "mNa", "mCl", "mC", "mH", "mMg", "mN", "mO", "mP"])
min_nbor_dist: 0.6599510670195264
max_nbor_size: [23, 26, 19, 16, 2, 2, 1, 1, 72, 37, 5, 0, 31, 29, 1, 21, 20, 5]
```

### `deepmd.entrypoints.test` module

Test trained DeePMD model.

`deepmd.entrypoints.test.test(*, model: str, system: str, datafile: str, set_prefix: str, numb_test: int, rand_seed: Optional[int], shuffle_test: bool, detail_file: str, atomic: bool, **kwargs)`

Test model predictions.

Parameters

model  
     [str] path where model is stored  
 system  
     [str] system directory  
 datafile  
     [str] the path to the list of systems to test  
 set\_prefix  
     [str] string prefix of set  
 numb\_test  
     [int] number of tests to do. 0 means all data.  
 rand\_seed  
     [Optional[int]] seed for random generator  
 shuffle\_test  
     [bool] whether to shuffle tests  
 detail\_file  
     [Optional[str]] file where test details will be output  
 atomic  
     [bool] whether per atom quantities should be computed  
 \*\*kwargs  
     additional arguments  
 Raises  
     RuntimeError  
         if no valid system was found

### deepmd.entrypoints.train module

DeePMD training entrypoint script.

Can handle local or distributed training.

```

deepmd.entrypoints.train.train(*, INPUT: str, init_model: Optional[str], restart: Optional[str], output:
    str, init_frz_model: str, mpi_log: str, log_level: int, log_path:
    Optional[str], is_compress: bool = False, skip_neighbor_stat: bool =
    False, finetune: Optional[str] = None, **kwargs)
  
```

Run DeePMD model training.

Parameters

INPUT  
     [str] json/yaml control file  
 init\_model  
     [Optional[str]] path prefix of checkpoint files or None  
 restart  
     [Optional[str]] path prefix of checkpoint files or None  
 output  
     [str] path for dump file with arguments

`init_frz_model`  
    [`str`] path to frozen model or None

`mpi_log`  
    [`str`] mpi logging mode

`log_level`  
    [`int`] logging level defined by int 0-3

`log_path`  
    [`Optional[str]`] logging file path or None if logs are to be output only to stdout

`is_compress`  
    [`bool`] indicates whether in the model compress mode

`skip_neighbor_stat`  
    [`bool`, default=False] skip checking neighbor statistics

`finetune`  
    [`Optional[str]`] path to pretrained model or None

`**kwargs`  
    additional arguments

Raises

`RuntimeError`  
        if distributed training job name is wrong

### **deepmd.entrypoints.transfer module**

Module used for transferring parameters between models.

`deepmd.entrypoints.transfer.transfer(*, old_model: str, raw_model: str, output: str, **kwargs)`

Transfer operation from old from graph to new prepared raw graph.

Parameters

`old_model`  
    [`str`] frozen old graph model

`raw_model`  
    [`str`] new model that will accept ops from old model

`output`  
    [`str`] new model with transfered parameters will be saved to this location

`**kwargs`  
    additional arguments



**deepmd.fit package**

```
class deepmd.fit.DOSFitting(*args, **kwargs)
```

Bases: *Fitting*

Fitting the density of states (DOS) of the system. The energy should be shifted by the fermi level.

Parameters

descript

The descriptor  $\mathcal{D}$

neuron

Number of neurons  $N$  in each hidden layer of the fitting net

resnet\_dt

Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

numb\_fparam

Number of frame parameter

numb\_aparam

Number of atomic parameter

! numb\_dos (added)

Number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

rcond

The condition number for the regression of atomic energy.

trainable

If the weights of fitting net are trainable. Suppose that we have  $N_l$  hidden layers in the fitting net, this list is of length  $N_l + 1$ , specifying if the hidden layers and the output layer are trainable.

seed

Random seed for initializing the network parameters.

activation\_function

The activation function  $\phi$  in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

precision

The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform\_seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

layer\_name

[list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use\_aparam\_as\_mask: bool, optional

If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

Attributes

**precision**

Precision of fitting network.

**Methods**

<i>build</i> (inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics.
<i>compute_output_stats</i> (all_stat[, mixed_type])	Compute the output statistics.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_numa_aparam</i> ()	Get the number of atomic parameters.
<i>get_numa_dos</i> ()	Get the number of gridpoints in energy space.
<i>get_numa_fparam</i> ()	Get the number of frame parameters.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(inputs: Tensor, natoms: Tensor, input\_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for fitting net.

Parameters

inputs

The input descriptor

input\_dict

Additional dict for inputs. if numb\_fparam > 0, should have input\_dict['fparam']  
if numb\_aparam > 0, should have input\_dict['aparam']

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i &lt; Ntypes+2, number of type i atoms

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

**ener**

The system energy

**compute\_input\_stats**(all\_stat: dict, protection: float = 0.01) → None

Compute the input statistics.

Parameters

all\_stat

if numb\_fparam &gt; 0 must have all\_stat['fparam'] if numb\_aparam &gt; 0 must have all\_stat['aparam'] can be prepared by model.make\_stat\_input

protection

Divided-by-zero protection

**compute\_output\_stats**(all\_stat: dict, mixed\_type: bool = False) → None

Compute the output statistics.

Parameters

all\_stat  
must have the following components: all\_stat['dos'] of shape n\_sys x n\_batch x n\_frame x numb\_dos can be prepared by model.make\_stat\_input

mixed\_type  
Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

**enable\_mixed\_precision**(mixed\_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

mixed\_prec  
The mixed precision setting used in the embedding net

**get\_loss**(loss: dict, lr) → Loss

Get the loss function.

Parameters

loss  
[dict] the loss dict

lr  
[LearningRateExp] the learning rate

Returns

Loss  
the loss function

**get\_numb\_aparam**() → int

Get the number of atomic parameters.

**get\_numb\_dos**() → int

Get the number of gridpoints in energy space.

**get\_numb\_fparam**() → int

Get the number of frame parameters.

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: str = "") → None

Init the fitting net variables with the given dict.

Parameters

graph  
[tf.Graph] The input frozen model graph

graph\_def  
[tf.GraphDef] The input frozen model graph\_def

suffix  
[str] suffix to name scope

```
class deepmd.fit.DipoleFittingSeA(*args, **kwargs)
```

Bases: *Fitting*

Fit the atomic dipole with descriptor se\_a.

Parameters

descript

[`tf.Tensor`] The descriptor

neuron

[`List[int]`] Number of neurons in each hidden layer of the fitting net

resnet\_dt

[`bool`] Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

sel\_type

[`List[int]`] The atom types selected to have an atomic dipole prediction. If is None, all atoms are selected.

seed

[`int`] Random seed for initializing the network parameters.

activation\_function

[`str`] The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

precision

[`str`] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform\_seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**precision**

Precision of fitting network.

## Methods

<i>build</i> (input_d, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_out_size</i> ()	Get the output size.
<i>get_sel_type</i> ()	Get selected type.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(input\_d: Tensor, rot\_mat: Tensor, natoms: Tensor, input\_dict: `Optional[dict]` = None, reuse: `Optional[bool]` = None, suffix: `str` = '') → Tensor

Build the computational graph for fitting net.

Parameters

input\_d

The input descriptor

`rot_mat`  
The rotation matrix from the descriptor.

`natoms`  
The number of atoms. This tensor has the length of `Ntypes + 2` `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms

`input_dict`  
Additional dict for inputs.

`reuse`  
The weights in the networks should be reused when get the variable.

`suffix`  
Name suffix to identify this descriptor

Returns

*dipole*  
The atomic dipole.

`enable_mixed_precision(mixed_prec: Optional[dict] = None) → None`

Reveive the mixed precision setting.

Parameters

`mixed_prec`  
The mixed precision setting used in the embedding net

`get_loss(loss: dict, lr) → Loss`

Get the loss function.

Parameters

`loss`  
[dict] the loss dict

`lr`  
[LearningRateExp] the learning rate

Returns

**Loss**  
the loss function

`get_out_size() → int`

Get the output size. Should be 3.

`get_sel_type() → int`

Get selected type.

`init_variables(graph: Graph, graph_def: GraphDef, suffix: str = '') → None`

Init the fitting net variables with the given dict.

Parameters

`graph`  
[tf.Graph] The input frozen model graph

`graph_def`  
[tf.GraphDef] The input frozen model graph\_def

suffix  
[`str`] suffix to name scope

`class deepmd.fit.EnerFitting(*args, **kwargs)`

Bases: *Fitting*

Fitting the energy of the system. The force and the virial can also be trained.

The potential energy  $E$  is a fitting network function of the descriptor  $\mathcal{D}$ :

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \dots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first  $n$  hidden layers  $\mathcal{L}^{(0)}, \dots, \mathcal{L}^{(n-1)}$  are given by

$$y = \mathcal{L}(x; w, b) = \phi(x^T w + b)$$

where  $x \in \mathbb{R}^{N_1}$  is the input vector and  $y \in \mathbb{R}^{N_2}$  is the output vector.  $w \in \mathbb{R}^{N_1 \times N_2}$  and  $b \in \mathbb{R}^{N_2}$  are weights and biases, respectively, both of which are trainable if `trainable[i]` is True.  $\phi$  is the activation function.

The output layer  $\mathcal{L}^{(n)}$  is given by

$$y = \mathcal{L}^{(n)}(x; w, b) = x^T w + b$$

where  $x \in \mathbb{R}^{N_{n-1}}$  is the input vector and  $y \in \mathbb{R}$  is the output scalar.  $w \in \mathbb{R}^{N_{n-1}}$  and  $b \in \mathbb{R}$  are weights and bias, respectively, both of which are trainable if `trainable[n]` is True.

Parameters

`descript`

The descriptor  $\mathcal{D}$

`neuron`

Number of neurons  $N$  in each hidden layer of the fitting net

`resnet_dt`

Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`numb_fparam`

Number of frame parameter

`numb_aparam`

Number of atomic parameter

`rcond`

The condition number for the regression of atomic energy.

`tot_ener_zero`

Force the total energy to zero. Useful for the charge fitting.

`trainable`

If the weights of fitting net are trainable. Suppose that we have  $N_l$  hidden layers in the fitting net, this list is of length  $N_l + 1$ , specifying if the hidden layers and the output layer are trainable.

`seed`

Random seed for initializing the network parameters.

`atom_ener`

Specifying atomic energy contribution in vacuum. The `set_davg_zero` key in the descriptor should be set.

activation\_function

The activation function  $\phi$  in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

precision

The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform\_seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

layer\_name

[list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use\_aparam\_as\_mask: bool, optional

If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

Attributes

**precision**

Precision of fitting network.

## Methods

<i>build</i> (inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
<i>change_energy_bias</i> (data, frozen_model, ...)	Change the energy bias according to the input data and the pretrained model.
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics.
<i>compute_output_stats</i> (all_stat[, mixed_type])	Compute the output statistics.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_numb_aparam</i> ()	Get the number of atomic parameters.
<i>get_numb_fparam</i> ()	Get the number of frame parameters.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(inputs: Tensor, natoms: Tensor, input\_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for fitting net.

Parameters

inputs

The input descriptor

input\_dict

Additional dict for inputs. if numb\_fparam > 0, should have input\_dict['fparam']  
if numb\_aparam > 0, should have input\_dict['aparam']

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number

of local atoms natoms[1]: total number of atoms held by this processor natoms[i]:  
 $2 \leq i < Ntypes+2$ , number of type i atoms

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

*ener*

The system energy

**change\_energy\_bias**(data, frozen\_model, origin\_type\_map, full\_type\_map, bias\_shift='delta',  
 ntest=10) → *None*

Change the energy bias according to the input data and the pretrained model.

Parameters

data

[*DeepmdDataSystem*] The training data.

frozen\_model

[*str*] The path file of frozen model.

origin\_type\_map

[*list*] The original type\_map in dataset, they are targets to change the energy bias.

full\_type\_map

[*str*] The full type\_map in pretrained model

bias\_shift

[*str*] The mode for changing energy bias : ['delta', 'statistic'] 'delta' : perform  
 predictions on energies of target dataset,

and do least square on the errors to obtain the target shift as bias.

'statistic' : directly use the statistic energy bias in the target dataset.

ntest

[*int*] The number of test samples in a system to change the energy bias.

**compute\_input\_stats**(all\_stat: *dict*, protection: *float* = 0.01) → *None*

Compute the input statistics.

Parameters

all\_stat

if numb\_fparam > 0 must have all\_stat['fparam'] if numb\_aparam > 0 must have  
 all\_stat['aparam'] can be prepared by model.make\_stat\_input

protection

Divided-by-zero protection

**compute\_output\_stats**(all\_stat: *dict*, mixed\_type: *bool* = False) → *None*

Compute the output statistics.

Parameters

all\_stat

must have the following components: all\_stat['energy'] of shape n\_sys x n\_batch  
 x n\_frame can be prepared by model.make\_stat\_input



`mixed_type`

Whether to perform the `mixed_type` mode. If True, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

`enable_mixed_precision(mixed_prec: Optional[dict] = None) → None`

Reveive the mixed precision setting.

Parameters

`mixed_prec`

The mixed precision setting used in the embedding net

`get_loss(loss: dict, lr) → Loss`

Get the loss function.

Parameters

`loss`

[dict] The loss function parameters.

`lr`

[LearningRateExp] The learning rate.

Returns

**Loss**

The loss function.

`get_numb_aparam() → int`

Get the number of atomic parameters.

`get_numb_fparam() → int`

Get the number of frame parameters.

`init_variables(graph: Graph, graph_def: GraphDef, suffix: str = "") → None`

Init the fitting net variables with the given dict.

Parameters

`graph`

[tf.Graph] The input frozen model graph

`graph_def`

[tf.GraphDef] The input frozen model graph\_def

`suffix`

[str] suffix to name scope

`class deepmd.fit.Fitting(*args, **kwargs)`

Bases: *PluginVariant*

Attributes

*precision*

Precision of fitting network.

## Methods

<code>get_loss(loss, lr)</code>	Get the loss function.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the fitting net variables with the given dict.
<code>register(key)</code>	Register a Fitting plugin.

**abstract** `get_loss(loss: dict, lr) → Loss`

Get the loss function.

Parameters

loss  
[dict] the loss dict

lr  
[LearningRateExp] the learning rate

Returns

**Loss**  
the loss function

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph  
[tf.Graph] The input frozen model graph

graph\_def  
[tf.GraphDef] The input frozen model graph\_def

suffix  
[str] suffix to name scope

## Notes

This method is called by others when the fitting supported initialization from the given variables.

**property precision: DType**

Precision of fitting network.

**static register**(key: str) → Callable

Register a Fitting plugin.

Parameters

key  
[str] the key of a Fitting

Returns

**Fitting**  
the registered Fitting

## Examples

```
>>> @Fitting.register("some_fitting")
class SomeFitting(Fitting):
    pass
```

```
class deepmd.fit.GlobalPolarFittingSeA(descrpt: Tensor, neuron: List[int] = [120, 120, 120],
                                       resnet_dt: bool = True, sel_type: Optional[List[int]] = None,
                                       fit_diag: bool = True, scale: Optional[List[float]] = None,
                                       diag_shift: Optional[List[float]] = None, seed: Optional[int]
                                       = None, activation_function: str = 'tanh', precision: str =
                                       'default')
```

Bases: `object`

Fit the system polarizability with descriptor `se_a`.

Parameters

`descrpt`

`[tf.Tensor]` The descriptor

`neuron`

`[List[int]]` Number of neurons in each hidden layer of the fitting net

`resnet_dt`

`[bool]` Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`sel_type`

`[List[int]]` The atom types selected to have an atomic polarizability prediction

`fit_diag`

`[bool]` Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

`scale`

`[List[float]]` The output of the fitting net (polarizability matrix) for type  $i$  atom will be scaled by `scale[i]`

`diag_shift`

`[List[float]]` The diagonal part of the polarizability matrix of type  $i$  will be shifted by `diag_shift[i]`. The shift operation is carried out after scale.

`seed`

`[int]` Random seed for initializing the network parameters.

`activation_function`

`[str]` The activation function in the embedding net. Supported options are "relu", "relu6", "softplus", "sigmoid", "tanh", "gelu", "gelu\_tf", "None", "none".

`precision`

`[str]` The precision of the embedding net parameters. Supported options are "default", "float16", "float32", "float64", "bfloat16".

## Methods

<i>build</i> (input_d, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_out_size</i> ()	Get the output size.
<i>get_sel_type</i> ()	Get selected atom types.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.

**build**(input\_d, rot\_mat, natoms, input\_dict: `Optional[dict] = None`, reuse=None, suffix='') → Tensor

Build the computational graph for fitting net.

Parameters

input\_d

The input descriptor

rot\_mat

The rotation matrix from the descriptor.

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

input\_dict

Additional dict for inputs.

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

*polar*

The system polarizability

**enable\_mixed\_precision**(mixed\_prec: `Optional[dict] = None`) → `None`

Reveive the mixed precision setting.

Parameters

mixed\_prec

The mixed precision setting used in the embedding net

**get\_loss**(loss: `dict`, lr) → `Loss`

Get the loss function.

Parameters

loss

[`dict`] the loss dict

lr

[`LearningRateExp`] the learning rate

Returns

```

    Loss
        the loss function

get_out_size() → int
    Get the output size. Should be 9.

get_sel_type() → int
    Get selected atom types.

init_variables(graph: Graph, graph_def: GraphDef, suffix: str = '') → None
    Init the fitting net variables with the given dict.

Parameters
    graph
        [tf.Graph] The input frozen model graph
    graph_def
        [tf.GraphDef] The input frozen model graph_def
    suffix
        [str] suffix to name scope

class deepmd.fit.PolarFittingSeA(*args, **kwargs)
    Bases: Fitting
    Fit the atomic polarizability with descriptor se_a.

Parameters
    descript
        [tf.Tensor] The descriptor
    neuron
        [List[int]] Number of neurons in each hidden layer of the fitting net
    resnet_dt
        [bool] Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$ 
    sel_type
        [List[int]] The atom types selected to have an atomic polarizability prediction. If
        is None, all atoms are selected.
    fit_diag
        [bool] Fit the diagonal part of the rotational invariant polarizability matrix, which
        will be converted to normal polarizability matrix by contracting with the rotation
        matrix.
    scale
        [List[float]] The output of the fitting net (polarizability matrix) for type i atom
        will be scaled by scale[i]
    diag_shift
        [List[float]] The diagonal part of the polarizability matrix of type i will be shifted
        by diag_shift[i]. The shift operation is carried out after scale.
    seed
        [int] Random seed for initializing the network parameters.
    activation_function
        [str] The activation function in the embedding net. Supported options are "relu",
        "relu6", "softplus", "sigmoid", "tanh", "gelu", "gelu_tf", "None", "none".

```

**precision**  
 [**str**] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
 Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

#### Attributes

**precision**  
 Precision of fitting network.

## Methods

<i>build</i> (input_d, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_out_size</i> ()	Get the output size.
<i>get_sel_type</i> ()	Get selected atom types.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(input\_d: Tensor, rot\_mat: Tensor, natoms: Tensor, input\_dict: **Optional**[dict] = None, reuse: **Optional**[bool] = None, suffix: str = "")

Build the computational graph for fitting net.

#### Parameters

**input\_d**  
 The input descriptor

**rot\_mat**  
 The rotation matrix from the descriptor.

**natoms**  
 The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

**input\_dict**  
 Additional dict for inputs.

**reuse**  
 The weights in the networks should be reused when get the variable.

**suffix**  
 Name suffix to identify this descriptor

#### Returns

**atomic\_polar**  
 The atomic polarizability

**compute\_input\_stats**(all\_stat, protection=0.01)  
 Compute the input statistics.

Parameters

`all_stat`  
Dictionary of inputs. can be prepared by `model.make_stat_input`

`protection`  
Divided-by-zero protection

**`enable_mixed_precision`**(mixed\_prec: `Optional[dict] = None`) → `None`

Reveive the mixed precision setting.

Parameters

`mixed_prec`  
The mixed precision setting used in the embedding net

**`get_loss`**(loss: `dict`, lr) → `Loss`

Get the loss function.

**`get_out_size`**() → `int`

Get the output size. Should be 9.

**`get_sel_type`**() → `List[int]`

Get selected atom types.

**`init_variables`**(graph: `Graph`, graph\_def: `GraphDef`, suffix: `str` = "") → `None`

Init the fitting net variables with the given dict.

Parameters

`graph`  
[`tf.Graph`] The input frozen model graph

`graph_def`  
[`tf.GraphDef`] The input frozen model graph\_def

`suffix`  
[`str`] suffix to name scope

## Submodules

### deepmd.fit.dipole module

**`class deepmd.fit.dipole.DipoleFittingSeA`**(\*args, \*\*kwargs)

Bases: *Fitting*

Fit the atomic dipole with descriptor se\_a.

Parameters

`descript`  
[`tf.Tensor`] The descripttor

`neuron`  
[`List[int]`] Number of neurons in each hidden layer of the fitting net

`resnet_dt`  
[`bool`] Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

**sel\_type**  
 [List[int]] The atom types selected to have an atomic dipole prediction. If is None, all atoms are selected.

**seed**  
 [int] Random seed for initializing the network parameters.

**activation\_function**  
 [str] The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**  
 [str] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**  
 Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

#### Attributes

**precision**  
 Precision of fitting network.

#### Methods

<i>build</i> (input_d, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_out_size</i> ()	Get the output size.
<i>get_sel_type</i> ()	Get selected type.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(input\_d: Tensor, rot\_mat: Tensor, natoms: Tensor, input\_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for fitting net.

#### Parameters

**input\_d**  
 The input descriptor

**rot\_mat**  
 The rotation matrix from the descriptor.

**natoms**  
 The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

**input\_dict**  
 Additional dict for inputs.

**reuse**  
 The weights in the networks should be reused when get the variable.



suffix  
Name suffix to identify this descriptor

Returns

**dipole**  
The atomic dipole.

**enable\_mixed\_precision**(mixed\_prec: `Optional[dict] = None`) → `None`

Reveive the mixed precision setting.

Parameters

mixed\_prec  
The mixed precision setting used in the embedding net

**get\_loss**(loss: `dict`, lr) → `Loss`

Get the loss function.

Parameters

loss  
[`dict`] the loss dict

lr  
[`LearningRateExp`] the learning rate

Returns

**Loss**  
the loss function

**get\_out\_size**() → `int`

Get the output size. Should be 3.

**get\_sel\_type**() → `int`

Get selected type.

**init\_variables**(graph: `Graph`, graph\_def: `GraphDef`, suffix: `str` = "") → `None`

Init the fitting net variables with the given dict.

Parameters

graph  
[`tf.Graph`] The input frozen model graph

graph\_def  
[`tf.GraphDef`] The input frozen model graph\_def

suffix  
[`str`] suffix to name scope

## deepmd.fit.dos module

```
class deepmd.fit.dos.DOSFitting(*args, **kwargs)
```

Bases: *Fitting*

Fitting the density of states (DOS) of the system. The energy should be shifted by the fermi level.

Parameters

descript

The descriptor  $\mathcal{D}$

neuron

Number of neurons  $N$  in each hidden layer of the fitting net

resnet\_dt

Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

numb\_fparam

Number of frame parameter

numb\_aparam

Number of atomic parameter

! numb\_dos (added)

Number of gridpoints on which the DOS is evaluated (NEDOS in VASP)

rcond

The condition number for the regression of atomic energy.

trainable

If the weights of fitting net are trainable. Suppose that we have  $N_l$  hidden layers in the fitting net, this list is of length  $N_l + 1$ , specifying if the hidden layers and the output layer are trainable.

seed

Random seed for initializing the network parameters.

activation\_function

The activation function  $\phi$  in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

precision

The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform\_seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

layer\_name

[list[Optional[str]], optional] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

use\_aparam\_as\_mask: bool, optional

If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

Attributes

**precision**  
Precision of fitting network.

## Methods

<i>build</i> (inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics.
<i>compute_output_stats</i> (all_stat[, mixed_type])	Compute the output statistics.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_numb_aparam</i> ()	Get the number of atomic parameters.
<i>get_numb_dos</i> ()	Get the number of gridpoints in energy space.
<i>get_numb_fparam</i> ()	Get the number of frame parameters.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(inputs: Tensor, natoms: Tensor, input\_dict: Optional[dict] = None, reuse: Optional[bool] = None, suffix: str = '') → Tensor

Build the computational graph for fitting net.

Parameters

**inputs**  
The input descriptor

**input\_dict**  
Additional dict for inputs. if numb\_fparam > 0, should have input\_dict['fparam']  
if numb\_aparam > 0, should have input\_dict['aparam']

**natoms**  
The number of atoms. This tensor has the length of Ntypes + 2  
natoms[0]: number of local atoms  
natoms[1]: total number of atoms held by this processor  
natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

**reuse**  
The weights in the networks should be reused when get the variable.

**suffix**  
Name suffix to identify this descriptor

Returns

**ener**  
The system energy

**compute\_input\_stats**(all\_stat: dict, protection: float = 0.01) → None

Compute the input statistics.

Parameters

**all\_stat**  
if numb\_fparam > 0 must have all\_stat['fparam']  
if numb\_aparam > 0 must have all\_stat['aparam']  
can be prepared by model.make\_stat\_input

**protection**  
Divided-by-zero protection

**compute\_output\_stats**(all\_stat: dict, mixed\_type: bool = False) → None

Compute the output statistics.

Parameters

all\_stat

must have the following components: all\_stat['dos'] of shape n\_sys x n\_batch x n\_frame x numb\_dos can be prepared by model.make\_stat\_input

mixed\_type

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

**enable\_mixed\_precision**(mixed\_prec: Optional[dict] = None) → None

Reveive the mixed precision setting.

Parameters

mixed\_prec

The mixed precision setting used in the embedding net

**get\_loss**(loss: dict, lr) → Loss

Get the loss function.

Parameters

loss

[dict] the loss dict

lr

[LearningRateExp] the learning rate

Returns

Loss

the loss function

**get\_numb\_aparam**() → int

Get the number of atomic parameters.

**get\_numb\_dos**() → int

Get the number of gridpoints in energy space.

**get\_numb\_fparam**() → int

Get the number of frame parameters.

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph

[tf.Graph] The input frozen model graph

graph\_def

[tf.GraphDef] The input frozen model graph\_def

suffix

[str] suffix to name scope

**deepmd.fit.ener module**

```
class deepmd.fit.ener.EnerFitting(*args, **kwargs)
```

Bases: *Fitting*

Fitting the energy of the system. The force and the virial can also be trained.

The potential energy  $E$  is a fitting network function of the descriptor  $\mathcal{D}$ :

$$E(\mathcal{D}) = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \dots \circ \mathcal{L}^{(1)} \circ \mathcal{L}^{(0)}$$

The first  $n$  hidden layers  $\mathcal{L}^{(0)}, \dots, \mathcal{L}^{(n-1)}$  are given by

$$y = \mathcal{L}(x; w, b) = \phi(x^T w + b)$$

where  $x \in \mathbb{R}^{N_1}$  is the input vector and  $y \in \mathbb{R}^{N_2}$  is the output vector.  $w \in \mathbb{R}^{N_1 \times N_2}$  and  $b \in \mathbb{R}^{N_2}$  are weights and biases, respectively, both of which are trainable if `trainable[i]` is True.  $\phi$  is the activation function.

The output layer  $\mathcal{L}^{(n)}$  is given by

$$y = \mathcal{L}^{(n)}(x; w, b) = x^T w + b$$

where  $x \in \mathbb{R}^{N_{n-1}}$  is the input vector and  $y \in \mathbb{R}$  is the output scalar.  $w \in \mathbb{R}^{N_{n-1}}$  and  $b \in \mathbb{R}$  are weights and bias, respectively, both of which are trainable if `trainable[n]` is True.

Parameters

`descript`

The descriptor  $\mathcal{D}$

`neuron`

Number of neurons  $N$  in each hidden layer of the fitting net

`resnet_dt`

Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`numb_fparam`

Number of frame parameter

`numb_aparam`

Number of atomic parameter

`rcond`

The condition number for the regression of atomic energy.

`tot_ener_zero`

Force the total energy to zero. Useful for the charge fitting.

`trainable`

If the weights of fitting net are trainable. Suppose that we have  $N_l$  hidden layers in the fitting net, this list is of length  $N_l + 1$ , specifying if the hidden layers and the output layer are trainable.

`seed`

Random seed for initializing the network parameters.

`atom_ener`

Specifying atomic energy contribution in vacuum. The `set_davg_zero` key in the descriptor should be set.

**activation\_function**

The activation function  $\phi$  in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

**precision**

The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

**uniform\_seed**

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

**layer\_name**

[[list](#)[[Optional](#)[[str](#)]], [optional](#)] The name of the each layer. If two layers, either in the same fitting or different fittings, have the same name, they will share the same neural network parameters.

**use\_aparam\_as\_mask: bool, optional**

If True, the atomic parameters will be used as a mask that determines the atom is real/virtual. And the aparam will not be used as the atomic parameters for embedding.

**Attributes****precision**

Precision of fitting network.

**Methods**

<i>build</i> (inputs, natoms[, input_dict, reuse, ...])	Build the computational graph for fitting net.
<i>change_energy_bias</i> (data, frozen_model, ...)	Change the energy bias according to the input data and the pretrained model.
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics.
<i>compute_output_stats</i> (all_stat[, mixed_type])	Compute the output statistics.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_numb_aparam</i> ()	Get the number of atomic parameters.
<i>get_numb_fparam</i> ()	Get the number of frame parameters.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(inputs: Tensor, natoms: Tensor, input\_dict: [Optional](#)[dict] = None, reuse: [Optional](#)[bool] = None, suffix: [str](#) = '') → Tensor

Build the computational graph for fitting net.

**Parameters****inputs**

The input descriptor

**input\_dict**

Additional dict for inputs. if numb\_fparam > 0, should have input\_dict['fparam']  
if numb\_aparam > 0, should have input\_dict['aparam']

**natoms**

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number

of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  
 $2 \leq i < Ntypes+2$ , number of type  $i$  atoms

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

**ener**

The system energy

**change\_energy\_bias**(data, frozen\_model, origin\_type\_map, full\_type\_map, bias\_shift='delta',  
 ntest=10) → `None`

Change the energy bias according to the input data and the pretrained model.

Parameters

data

[`DeepmdDataSystem`] The training data.

frozen\_model

[`str`] The path file of frozen model.

origin\_type\_map

[`list`] The original type\_map in dataset, they are targets to change the energy bias.

full\_type\_map

[`str`] The full type\_map in pretrained model

bias\_shift

[`str`] The mode for changing energy bias : ['delta', 'statistic'] 'delta' : perform  
 predictions on energies of target dataset,

and do least square on the errors to obtain the target shift as bias.

'statistic' : directly use the statistic energy bias in the target dataset.

ntest

[`int`] The number of test samples in a system to change the energy bias.

**compute\_input\_stats**(all\_stat: `dict`, protection: `float` = 0.01) → `None`

Compute the input statistics.

Parameters

all\_stat

if `numb_fparam` > 0 must have `all_stat['fparam']` if `numb_aparam` > 0 must have  
`all_stat['aparam']` can be prepared by `model.make_stat_input`

protection

Divided-by-zero protection

**compute\_output\_stats**(all\_stat: `dict`, mixed\_type: `bool` = False) → `None`

Compute the output statistics.

Parameters

all\_stat

must have the following components: `all_stat['energy']` of shape `n_sys x n_batch`  
`x n_frame` can be prepared by `model.make_stat_input`

`mixed_type`

Whether to perform the `mixed_type` mode. If True, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

`enable_mixed_precision(mixed_prec: Optional[dict] = None) → None`

Reveive the mixed precision setting.

Parameters

`mixed_prec`

The mixed precision setting used in the embedding net

`get_loss(loss: dict, lr) → Loss`

Get the loss function.

Parameters

`loss`

[dict] The loss function parameters.

`lr`

[LearningRateExp] The learning rate.

Returns

**Loss**

The loss function.

`get_numb_aparam() → int`

Get the number of atomic parameters.

`get_numb_fparam() → int`

Get the number of frame parameters.

`init_variables(graph: Graph, graph_def: GraphDef, suffix: str = "") → None`

Init the fitting net variables with the given dict.

Parameters

`graph`

[tf.Graph] The input frozen model graph

`graph_def`

[tf.GraphDef] The input frozen model graph\_def

`suffix`

[str] suffix to name scope

## deepmd.fit.fitting module

`class deepmd.fit.fitting.Fitting(*args, **kwargs)`

Bases: *PluginVariant*

Attributes

*precision*

Precision of fitting network.



## Methods

<code>get_loss(loss, lr)</code>	Get the loss function.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the fitting net variables with the given dict.
<code>register(key)</code>	Register a Fitting plugin.

**abstract** `get_loss(loss: dict, lr) → Loss`

Get the loss function.

Parameters

loss  
[dict] the loss dict

lr  
[LearningRateExp] the learning rate

Returns

**Loss**  
the loss function

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: str = '') → None

Init the fitting net variables with the given dict.

Parameters

graph  
[tf.Graph] The input frozen model graph

graph\_def  
[tf.GraphDef] The input frozen model graph\_def

suffix  
[str] suffix to name scope

## Notes

This method is called by others when the fitting supported initialization from the given variables.

**property precision: DType**

Precision of fitting network.

**static register**(key: str) → Callable

Register a Fitting plugin.

Parameters

key  
[str] the key of a Fitting

Returns

**Fitting**  
the registered Fitting

## Examples

```
>>> @Fitting.register("some_fitting")
class SomeFitting(Fitting):
    pass
```

## deepmd.fit.polar module

```
class deepmd.fit.polar.GlobalPolarFittingSeA(descrpt: Tensor, neuron: List[int] = [120, 120, 120],
                                             resnet_dt: bool = True, sel_type: Optional[List[int]]
                                             = None, fit_diag: bool = True, scale:
                                             Optional[List[float]] = None, diag_shift:
                                             Optional[List[float]] = None, seed: Optional[int] =
                                             None, activation_function: str = 'tanh', precision: str
                                             = 'default')
```

Bases: `object`

Fit the system polarizability with descriptor `se_a`.

Parameters

`descrpt`  
`[tf.Tensor]` The descriptor

`neuron`  
`[List[int]]` Number of neurons in each hidden layer of the fitting net

`resnet_dt`  
`[bool]` Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`sel_type`  
`[List[int]]` The atom types selected to have an atomic polarizability prediction

`fit_diag`  
`[bool]` Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.

`scale`  
`[List[float]]` The output of the fitting net (polarizability matrix) for type  $i$  atom will be scaled by `scale[i]`

`diag_shift`  
`[List[float]]` The diagonal part of the polarizability matrix of type  $i$  will be shifted by `diag_shift[i]`. The shift operation is carried out after scale.

`seed`  
`[int]` Random seed for initializing the network parameters.

`activation_function`  
`[str]` The activation function in the embedding net. Supported options are "relu", "relu6", "softplus", "sigmoid", "tanh", "gelu", "gelu\_tf", "None", "none".

`precision`  
`[str]` The precision of the embedding net parameters. Supported options are "default", "float16", "float32", "float64", "bfloat16".

## Methods

<code>build(input_d, rot_mat, natoms[, ...])</code>	Build the computational graph for fitting net.
<code>enable_mixed_precision([mixed_prec])</code>	Reveive the mixed precision setting.
<code>get_loss(loss, lr)</code>	Get the loss function.
<code>get_out_size()</code>	Get the output size.
<code>get_sel_type()</code>	Get selected atom types.
<code>init_variables(graph, graph_def[, suffix])</code>	Init the fitting net variables with the given dict.

**build**(input\_d, rot\_mat, natoms, input\_dict: `Optional[dict] = None`, reuse=None, suffix='') → Tensor

Build the computational graph for fitting net.

Parameters

input\_d

The input descriptor

rot\_mat

The rotation matrix from the descriptor.

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

input\_dict

Additional dict for inputs.

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

**polar**

The system polarizability

**enable\_mixed\_precision**(mixed\_prec: `Optional[dict] = None`) → `None`

Reveive the mixed precision setting.

Parameters

mixed\_prec

The mixed precision setting used in the embedding net

**get\_loss**(loss: `dict`, lr) → `Loss`

Get the loss function.

Parameters

loss

[`dict`] the loss dict

lr

[`LearningRateExp`] the learning rate

Returns

**Loss**  
the loss function

**get\_out\_size()** → `int`  
Get the output size. Should be 9.

**get\_sel\_type()** → `int`  
Get selected atom types.

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix: `str` = "") → `None`  
Init the fitting net variables with the given dict.

Parameters

- graph  
[`tf.Graph`] The input frozen model graph
- graph\_def  
[`tf.GraphDef`] The input frozen model graph\_def
- suffix  
[`str`] suffix to name scope

**class** deepmd.fit.polar.PolarFittingSeA(\*args, \*\*kwargs)  
Bases: *Fitting*  
Fit the atomic polarizability with descriptor se\_a.

Parameters

- descript  
[`tf.Tensor`] The descriptor
- neuron  
[`List[int]`] Number of neurons in each hidden layer of the fitting net
- resnet\_dt  
[`bool`] Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$
- sel\_type  
[`List[int]`] The atom types selected to have an atomic polarizability prediction. If is None, all atoms are selected.
- fit\_diag  
[`bool`] Fit the diagonal part of the rotational invariant polarizability matrix, which will be converted to normal polarizability matrix by contracting with the rotation matrix.
- scale  
[`List[float]`] The output of the fitting net (polarizability matrix) for type i atom will be scaled by scale[i]
- diag\_shift  
[`List[float]`] The diagonal part of the polarizability matrix of type i will be shifted by diag\_shift[i]. The shift operation is carried out after scale.
- seed  
[`int`] Random seed for initializing the network parameters.
- activation\_function  
[`str`] The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

precision

[[str](#)] The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

uniform\_seed

Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

Attributes

**precision**

Precision of fitting network.

## Methods

<i>build</i> (input_d, rot_mat, natoms[, ...])	Build the computational graph for fitting net.
<i>compute_input_stats</i> (all_stat[, protection])	Compute the input statistics.
<i>enable_mixed_precision</i> ([mixed_prec])	Reveive the mixed precision setting.
<i>get_loss</i> (loss, lr)	Get the loss function.
<i>get_out_size</i> ()	Get the output size.
<i>get_sel_type</i> ()	Get selected atom types.
<i>init_variables</i> (graph, graph_def[, suffix])	Init the fitting net variables with the given dict.
<i>register</i> (key)	Register a Fitting plugin.

**build**(input\_d: Tensor, rot\_mat: Tensor, natoms: Tensor, input\_dict: [Optional](#)[dict] = None, reuse: [Optional](#)[bool] = None, suffix: [str](#) = "")

Build the computational graph for fitting net.

Parameters

input\_d

The input descriptor

rot\_mat

The rotation matrix from the descriptor.

natoms

The number of atoms. This tensor has the length of Ntypes + 2 natoms[0]: number of local atoms natoms[1]: total number of atoms held by this processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

input\_dict

Additional dict for inputs.

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

**atomic\_polar**

The atomic polarizability

**compute\_input\_stats**(all\_stat, protection=0.01)

Compute the input statistics.

Parameters

`all_stat`  
Dictionary of inputs. can be prepared by `model.make_stat_input`

`protection`  
Divided-by-zero protection

`enable_mixed_precision(mixed_prec: Optional[dict] = None) → None`

Reveive the mixed precision setting.

Parameters

`mixed_prec`  
The mixed precision setting used in the embedding net

`get_loss(loss: dict, lr) → Loss`

Get the loss function.

`get_out_size() → int`

Get the output size. Should be 9.

`get_sel_type() → List[int]`

Get selected atom types.

`init_variables(graph: Graph, graph_def: GraphDef, suffix: str = "") → None`

Init the fitting net variables with the given dict.

Parameters

`graph`  
[`tf.Graph`] The input frozen model graph

`graph_def`  
[`tf.GraphDef`] The input frozen model graph\_def

`suffix`  
[`str`] suffix to name scope

## deepmd.infer package

Submodule containing all the implemented potentials.

```
class deepmd.infer.DeepDOS(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False,
                             auto_batch_size: Union[bool, int, AutoBatchSize] = True, input_map:
                             Optional[dict] = None)
```

Bases: *DeepEval*

Constructor.

Parameters

`model_file`  
[`Path`] The name of the frozen model file.

`load_prefix: str`  
The prefix in the load computational graph

`default_tf_graph`  
[`bool`] If uses the default tf graph, otherwise build a new tf graph for evaluation

**auto\_batch\_size**  
 [bool or int or AutomaticBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

**input\_map**  
 [dict, optional] The input map for tf.import\_graph\_def. Only work with default tf graph

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

#### Attributes

**model\_type**  
 Get type of model.

**model\_version**  
 Get version of model.

**sess**  
 Get TF session.

#### Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the dos, atom_dos by using this model.
<code>eval_descriptor(coords, cells, atom_types[, ...])</code>	Evaluate descriptors by using this DP.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Get the number (dimension) of atomic parameters of this DP.
<code>get_dim_fparam()</code>	Get the number (dimension) of frame parameters of this DP.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_numb_dos()</code>	Get the length of DOS output of this DP model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Unsupported in this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

**eval**(coords: ndarray, cells: ndarray, atom\_types: List[int], atomic: bool = False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, mixed\_type: bool = False) → Tuple[ndarray, ...]

Evaluate the dos, atom\_dos by using this model.

Parameters

`coords`

The coordinates of atoms. The array should be of size `nframes x natoms x 3`

`cells`

The cell of the region. If `None` then non-PBC is assumed, otherwise using PBC. The array should be of size `nframes x 9`

`atom_types`

The atom types The list should contain `natoms` ints

`atomic`

Calculate the atomic energy and virial

`fparam`

The frame parameter. The array can be of size : - `nframes x dim_fparam`. - `dim_fparam`. Then all frames are assumed to be provided with the same `fparam`.

`aparam`

The atomic parameter The array can be of size : - `nframes x natoms x dim_aparam`. - `natoms x dim_aparam`. Then all frames are assumed to be provided with the same `aparam`. - `dim_aparam`. Then all frames and atoms are provided with the same `aparam`.

`mixed_type`

Whether to perform the `mixed_type` mode. If `True`, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

Returns

`dos`

The electron density of state.

`atom_dos`

The atom-sited density of state. Only returned when `atomic == True`

`eval_descriptor`(`coords`: `ndarray`, `cells`: `ndarray`, `atom_types`: `List[int]`, `fparam`: `Optional[ndarray]` = `None`, `aparam`: `Optional[ndarray]` = `None`, `efield`: `Optional[ndarray]` = `None`, `mixed_type`: `bool` = `False`) → array

Evaluate descriptors by using this DP.

Parameters

`coords`

The coordinates of atoms. The array should be of size `nframes x natoms x 3`

`cells`

The cell of the region. If `None` then non-PBC is assumed, otherwise using PBC. The array should be of size `nframes x 9`

`atom_types`

The atom types The list should contain `natoms` ints

`fparam`

The frame parameter. The array can be of size : - `nframes x dim_fparam`. - `dim_fparam`. Then all frames are assumed to be provided with the same `fparam`.

`aparam`

The atomic parameter The array can be of size : - `nframes x natoms x dim_aparam`. - `natoms x dim_aparam`. Then all frames are assumed to be provided with the same `aparam`. - `dim_aparam`. Then all frames and atoms are provided with the same `aparam`.



`efield`

The external field on atoms. The array should be of size `nframes x natoms x 3`

`mixed_type`

Whether to perform the `mixed_type` mode. If `True`, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

Returns

**descriptor**

Descriptors.

`get_dim_aparam()` → `int`

Get the number (dimension) of atomic parameters of this DP.

`get_dim_fparam()` → `int`

Get the number (dimension) of frame parameters of this DP.

`get_ntypes()` → `int`

Get the number of atom types of this model.

`get_numb_dos()` → `int`

Get the length of DOS output of this DP model.

`get_rcut()` → `float`

Get the cut-off radius of this model.

`get_sel_type()` → `List[int]`

Unsupported in this model.

`get_type_map()` → `List[str]`

Get the type map (element name of the atom types) of this model.

`load_prefix:` `str`

`class deepmd.infer.DeepDipole(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False, input_map: Optional[dict] = None, neighbor_list=None)`

Bases: *DeepTensor*

Constructor.

Parameters

`model_file`

[`Path`] The name of the frozen model file.

`load_prefix:` `str`

The prefix in the load computational graph

`default_tf_graph`

[`bool`] If uses the default tf graph, otherwise build a new tf graph for evaluation

`input_map`

[`dict`, `optional`] The input map for `tf.import_graph_def`. Only work with default tf graph

`neighbor_list`

[`ase.neighborlist.NeighborList`, `optional`] The neighbor list object. If `None`, then build the native neighbor list.

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

#### Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.

#### Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`get_dim_aparam() → int`  
Unsupported in this model.

`get_dim_fparam() → int`  
Unsupported in this model.

```
class deepmd.infer.DeepEval(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool =
    False, auto_batch_size: Union[bool, int, AutoBatchSize] = False,
    input_map: Optional[dict] = None, neighbor_list=None)
```

Bases: `object`

Common methods for DeepPot, DeepWFC, DeepPolar, ...

#### Parameters

**model\_file**  
[Path] The name of the frozen model file.

`load_prefix`: str  
The prefix in the load computational graph

`default_tf_graph`  
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

`auto_batch_size`  
[bool or int or AutomaticBatchSize, default: False] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

`input_map`  
[dict, optional] The input map for `tf.import_graph_def`. Only work with default tf graph

`neighbor_list`  
[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

#### Attributes

`model_type`  
Get type of model.

`model_version`  
Get version of model.

`sess`  
Get TF session.

#### Methods

<code>build_neighbor_list</code> (coords, cell, atype, ...)	Make the mesh with neighbor list for a single frame.
<code>eval_typeebd</code> ()	Evaluate output of type embedding network by using this model.
<code>make_natoms_vec</code> (atom_types[, mixed_type])	Make the natom vector used by deepmd-kit.
<code>reverse_map</code> (vec, imap)	Reverse mapping of a vector according to the index map.
<code>sort_input</code> (coord, atom_type[, sel_atoms, ...])	Sort atoms in the system according their types.

**build\_neighbor\_list**(coords: ndarray, cell: Optional[ndarray], atype: ndarray, imap: ndarray, neighbor\_list)

Make the mesh with neighbor list for a single frame.

#### Parameters

`coords`  
[np.ndarray] The coordinates of atoms. Should be of shape [natoms, 3]

`cell`  
[Optional[np.ndarray]] The cell of the system. Should be of shape [3, 3]

`atype`  
[np.ndarray] The type of atoms. Should be of shape [natoms]

`imap`  
[np.ndarray] The index map of atoms. Should be of shape [natoms]

`neighbor_list`  
`[ase.neighborlist.NewPrimitiveNeighborList]` ASE neighbor list. The following method or attribute will be used/set: `bothways`, `self_interaction`, `update`, `build`, `first_neigh`, `pair_second`, `offset_vec`.

Returns

`natoms_vec`  
`[np.ndarray]` The number of atoms. This tensor has the length of `Ntypes + 2`  
`natoms[0]`: `nloc` `natoms[1]`: `nall` `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms for `nloc`

`coords`  
`[np.ndarray]` The coordinates of atoms, including ghost atoms. Should be of shape `[nframes, nall, 3]`

`atype`  
`[np.ndarray]` The type of atoms, including ghost atoms. Should be of shape `[nall]`

`mesh`  
`[np.ndarray]` The mesh in `nei_mode=4`.

`imap`  
`[np.ndarray]` The index map of atoms. Should be of shape `[nall]`

`ghost_map`  
`[np.ndarray]` The index map of ghost atoms. Should be of shape `[nghost]`

`eval_typeebd()` → `ndarray`

Evaluate output of type embedding network by using this model.

Returns

`np.ndarray`  
The output of type embedding network. The shape is `[ntypes, o_size]`, where `ntypes` is the number of types, and `o_size` is the number of nodes in the output layer.

Raises

`KeyError`  
If the model does not enable type embedding.

See also:

`deepmd.utils.type_embed.TypeEmbedNet`

The type embedding network.

## Examples

Get the output of type embedding network of `graph.pb`:

```
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential('graph.pb')
>>> dp.eval_typeebd()
```

`load_prefix:` `str`

`make_natoms_vec(atom_types: ndarray, mixed_type: bool = False)` → `ndarray`

Make the natom vector used by deepmd-kit.

## Parameters

`atom_types`

The type of atoms

`mixed_type`Whether to perform the `mixed_type` mode. If True, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

## Returns

`natoms`The number of atoms. This tensor has the length of `Ntypes + 2` `natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms**property** `model_type: str`

Get type of model.

:type:str

**property** `model_version: str`

Get version of model.

## Returns

`str`

version of model

**static** `reverse_map(vec: ndarray, imap: List[int]) → ndarray`

Reverse mapping of a vector according to the index map.

## Parameters

`vec`Input vector. Be of shape `[nframes, natoms, -1]``imap`Index map. Be of shape `[natoms]`

## Returns

`vec_out`

Reverse mapped vector.

**property** `sess: Session`

Get TF session.

**static** `sort_input(coord: ndarray, atom_type: ndarray, sel_atoms: Optional[List[int]] = None, mixed_type: bool = False)`

Sort atoms in the system according their types.

## Parameters

`coord`The coordinates of atoms. Should be of shape `[nframes, natoms, 3]``atom_type`The type of atoms Should be of shape `[natoms]``sel_atoms`

The selected atoms by type

**mixed\_type**

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

**coord\_out**

The coordinates after sorting

**atom\_type\_out**

The atom types after sorting

**idx\_map**

The index mapping from the input to the output. For example coord\_out = coord[:,idx\_map,:]

**sel\_atom\_type**

Only output if sel\_atoms is not None The sorted selected atom types

**sel\_idx\_map**

Only output if sel\_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

```
class deepmd.infer.DeepGlobalPolar(model_file: str, load_prefix: str = 'load', default_tf_graph: bool
                                   = False, neighbor_list=None)
```

Bases: *DeepTensor*

Constructor.

Parameters

**model\_file**

[str] The name of the frozen model file.

**load\_prefix: str**

The prefix in the load computational graph

**default\_tf\_graph**

[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

**neighbor\_list**

[ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

Attributes

**model\_type**

Get type of model.

**model\_version**

Get version of model.

**sess**

Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

**eval** (coords: `ndarray`, cells: `ndarray`, atom\_types: `List[int]`, atomic: `bool` = False, fparam: `Optional[ndarray]` = None, aparam: `Optional[ndarray]` = None, efield: `Optional[ndarray]` = None) → `ndarray`

Evaluate the model.

Parameters

coords

The coordinates of atoms. The array should be of size nframes x natoms x 3

cells

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom\_types

The atom types The list should contain natoms ints

atomic

Not used in this model

fparam

Not used in this model

aparam

Not used in this model

efield

Not used in this model

Returns

**tensor**

The returned tensor If atomic == False then of size nframes x variable\_dof else of size nframes x natoms x variable\_dof

`get_dim_aparam() → int`

Unsupported in this model.

`get_dim_fparam() → int`

Unsupported in this model.

```
class deepmd.infer.DeepPolar(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool =
                             False, input_map: Optional[dict] = None, neighbor_list=None)
```

Bases: *DeepTensor*

Constructor.

Parameters

`model_file`

[`Path`] The name of the frozen model file.

`load_prefix: str`

The prefix in the load computational graph

`default_tf_graph`

[`bool`] If uses the default tf graph, otherwise build a new tf graph for evaluation

`input_map`

[`dict`, `optional`] The input map for `tf.import_graph_def`. Only work with default tf graph

`neighbor_list`

[`ase.neighborlist.NeighborList`, `optional`] The neighbor list object. If None, then build the native neighbor list.

Warning: For developers: DeepTensor initializer must be called at the end after `self.tensors` are modified because it uses the data in `self.tensors` dict. Do not change the order!

Attributes

`model_type`

Get type of model.

`model_version`

Get version of model.

`sess`

Get TF session.



## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`get_dim_aparam() → int`

Unsupported in this model.

`get_dim_fparam() → int`

Unsupported in this model.

```
class deepmd.infer.DeepPot(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False,
                           auto_batch_size: Union[bool, int, AutoBatchSize] = True, input_map:
                           Optional[dict] = None, neighbor_list=None)
```

Bases: *DeepEval*

Constructor.

Parameters

`model_file`

[Path] The name of the frozen model file.

`load_prefix: str`

The prefix in the load computational graph

`default_tf_graph`

[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

`auto_batch_size`

[bool or int or AutomaticBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

`input_map`

[dict, optional] The input map for tf.import\_graph\_def. Only work with default tf graph

`neighbor_list`

[ase.neighborlist.NewPrimitiveNeighborList, optional] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

## Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot('graph.pb')
>>> coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1,0,1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where e, f and v are predicted energy, force and virial of the system, respectively.

### Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the energy, force and virial by using this DP.
<code>eval_descriptor(coords, cells, atom_types[, ...])</code>	Evaluate descriptors by using this DP.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_descriptor_type()</code>	Get the descriptor type of this model.
<code>get_dim_aparam()</code>	Get the number (dimension) of atomic parameters of this DP.
<code>get_dim_fparam()</code>	Get the number (dimension) of frame parameters of this DP.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_ntypes_spin()</code>	Get the number of spin atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Unsupported in this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

**eval**(coords: ndarray, cells: ndarray, atom\_types: List[int], atomic: bool = False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed\_type: bool = False) → Tuple[ndarray, ...]

Evaluate the energy, force and virial by using this DP.

Parameters

coords

The coordinates of atoms. The array should be of size nframes x natoms x 3

cells

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom\_types

The atom types The list should contain natoms ints

atomic

Calculate the atomic energy and virial

fparam

The frame parameter. The array can be of size : - nframes x dim\_fparam. - dim\_fparam. Then all frames are assumed to be provided with the same fparam.

aparam

The atomic parameter The array can be of size : - nframes x natoms x dim\_aparam. - natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. - dim\_aparam. Then all frames and atoms are provided with the same aparam.

efield

The external field on atoms. The array should be of size nframes x natoms x 3

mixed\_type

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

**energy**

The system energy.

**force**

The force on each atom

**virial**

The virial

**atom\_energy**

The atomic energy. Only returned when atomic == True

**atom\_virial**

The atomic virial. Only returned when atomic == True

**eval\_descriptor**(coords: ndarray, cells: ndarray, atom\_types: List[int], fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed\_type: bool = False) → array

Evaluate descriptors by using this DP.

Parameters

`coords`

The coordinates of atoms. The array should be of size `nframes x natoms x 3`

`cells`

The cell of the region. If `None` then non-PBC is assumed, otherwise using PBC. The array should be of size `nframes x 9`

`atom_types`

The atom types The list should contain `natoms` ints

`fparam`

The frame parameter. The array can be of size : - `nframes x dim_fparam`. - `dim_fparam`. Then all frames are assumed to be provided with the same `fparam`.

`aparam`

The atomic parameter The array can be of size : - `nframes x natoms x dim_aparam`. - `natoms x dim_aparam`. Then all frames are assumed to be provided with the same `aparam`. - `dim_aparam`. Then all frames and atoms are provided with the same `aparam`.

`efield`

The external field on atoms. The array should be of size `nframes x natoms x 3`

`mixed_type`

Whether to perform the `mixed_type` mode. If `True`, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

Returns

**`descriptor`**

Descriptors.

**`get_descriptor_type()`** → `List[int]`

Get the descriptor type of this model.

**`get_dim_aparam()`** → `int`

Get the number (dimension) of atomic parameters of this DP.

**`get_dim_fparam()`** → `int`

Get the number (dimension) of frame parameters of this DP.

**`get_ntypes()`** → `int`

Get the number of atom types of this model.

**`get_ntypes_spin()`**

Get the number of spin atom types of this model.

**`get_rcut()`** → `float`

Get the cut-off radius of this model.

**`get_sel_type()`** → `List[int]`

Unsupported in this model.

**`get_type_map()`** → `List[str]`

Get the type map (element name of the atom types) of this model.

**`load_prefix:`** `str`

```
deepmd.infer.DeepPotential(model_file: Union[str, Path], load_prefix: str = 'load', default_tf_graph:
    bool = False, input_map: Optional[dict] = None, neighbor_list=None) →
    Union[DeepDipole, DeepGlobalPolar, DeepPolar, DeepPot, DeepDOS,
    DeepWFC]
```

Factory function that will initialize appropriate potential read from `model_file`.

Parameters

`model_file`  
`[str]` The name of the frozen model file.

`load_prefix`  
`[str]` The prefix in the load computational graph

`default_tf_graph`  
`[bool]` If uses the default tf graph, otherwise build a new tf graph for evaluation

`input_map`  
`[dict, optional]` The input map for `tf.import_graph_def`. Only work with default tf graph

`neighbor_list`  
`[ase.neighborlist.NeighborList, optional]` The neighbor list object. If None, then build the native neighbor list.

Returns

`Union[DeepDipole, DeepGlobalPolar, DeepPolar, DeepPot, DeepWFC]`  
 one of the available potentials

Raises

`RuntimeError`  
 if model file does not correspond to any implemented potential

```
class deepmd.infer.DeepWFC(model_file: Path, load_prefix: str = 'load', default_tf_graph: bool = False,
    input_map: Optional[dict] = None)
```

Bases: `DeepTensor`

Constructor.

Parameters

`model_file`  
`[Path]` The name of the frozen model file.

`load_prefix: str`  
 The prefix in the load computational graph

`default_tf_graph`  
`[bool]` If uses the default tf graph, otherwise build a new tf graph for evaluation

`input_map`  
`[dict, optional]` The input map for `tf.import_graph_def`. Only work with default tf graph

Warning: For developers: `DeepTensor` initializer must be called at the end after `self.tensors` are modified because it uses the data in `self.tensors` dict. Do not change the order!

Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`get_dim_aparam() → int`  
Unsupported in this model.

`get_dim_fparam() → int`  
Unsupported in this model.

```
class deepmd.infer.DipoleChargeModifier(model_name: str, model_charge_map: List[float],
                                         sys_charge_map: List[float], ewald_h: float = 1,
                                         ewald_beta: float = 1)
```

Bases: *DeepDipole*

Parameters

**model\_name**  
The model file for the DeepDipole model

**model\_charge\_map**  
Gives the amount of charge for the wfcc

**sys\_charge\_map**  
Gives the amount of charge for the real atoms

**ewald\_h**  
Grid spacing of the reciprocal part of Ewald sum. Unit: Å

`ewald_beta`  
Splitting parameter of the Ewald sum. Unit:  $\text{\AA}^{-1}$

#### Attributes

`model_type`  
Get type of model.

`model_version`  
Get version of model.

`sess`  
Get TF session.

#### Methods

<code>build_fv_graph()</code>	Build the computational graph for the force and virial inference.
<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coord, box, atype[, eval_fv])</code>	Evaluate the modification.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>modify_data(data, data_sys)</code>	Modify data.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`build_fv_graph()` → Tensor

Build the computational graph for the force and virial inference.

`eval(coord: ndarray, box: ndarray, atype: ndarray, eval_fv: bool = True)` → Tuple[ndarray, ndarray, ndarray]

Evaluate the modification.

#### Parameters

`coord`  
The coordinates of atoms

`box`  
The simulation region. PBC is assumed

`atype`  
The atom types

`eval_fv`  
Evaluate force and virial

Returns

`tot_e`  
The energy modification

`tot_f`  
The force modification

`tot_v`  
The virial modification

`modify_data`(data: dict, data\_sys: DeepmdData) → None

Modify data.

Parameters

`data`  
Internal data of DeepmdData. Be a dict, has the following keys - coord coordinates - box simulation box - type atom types - find\_energy tells if data has energy - find\_force tells if data has force - find\_virial tells if data has virial - energy energy - force force - virial virial

`data_sys`  
[DeepmdData] The data system.

`class deepmd.infer.EwaldRecp`(hh, beta)

Bases: `object`

Evaluate the reciprocal part of the Ewald sum.

## Methods

---

<code>eval</code> (coord, charge, box)	Evaluate.
--	-----------

---

`eval`(coord: ndarray, charge: ndarray, box: ndarray) → Tuple[ndarray, ndarray, ndarray]

Evaluate.

Parameters

`coord`  
The coordinates of atoms

`charge`  
The atomic charge

`box`  
The simulation region. PBC is assumed

Returns

`e`  
The energy

`f`  
The force

`v`  
The virial



```
deepmd.infer.calc_model_devi(coord, box, atype, models, fname=None, frequency=1,
                             mixed_type=False, fparam: Optional[ndarray] = None, aparam:
                             Optional[ndarray] = None, real_data: Optional[dict] = None, atomic:
                             bool = False, relative: Optional[float] = None, relative_v:
                             Optional[float] = None)
```

Python interface to calculate model deviation.

Parameters

**coord**  
[`numpy.ndarray`, n\_frames x n\_atoms x 3] Coordinates of system to calculate

**box**  
[`numpy.ndarray` or `None`, n\_frames x 3 x 3] Box to specify periodic boundary condition. If `None`, no pbc will be used

**atype**  
[`numpy.ndarray`, n\_atoms x 1] Atom types

**models**  
[list of *DeepPot* models] Models used to evaluate deviation

**fname**  
[`str` or `None`] File to dump results, default `None`

**frequency**  
[`int`] Steps between frames (if the system is given by molecular dynamics engine), default 1

**mixed\_type**  
[`bool`] Whether the input atype is in mixed\_type format or not

**fparam**  
[`numpy.ndarray`] frame specific parameters

**aparam**  
[`numpy.ndarray`] atomic specific parameters

**real\_data**  
[`dict`, optional] real data to calculate RMS real error

**atomic**  
[`bool`, default: `False`] If `True`, calculate the force model deviation of each atom.

**relative**  
[`float`, default: `None`] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

**relative\_v**  
[`float`, default: `None`] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

Returns

**model\_devi**  
[`numpy.ndarray`, n\_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max\_devi\_v, min\_devi\_v, avg\_devi\_v, max\_devi\_f, min\_devi\_f, avg\_devi\_f, devi\_e.

## Examples

```

>>> from deepmd.infer import calc_model_devi
>>> from deepmd.infer import DeepPot as DP
>>> import numpy as np
>>> coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1,0,1]
>>> graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
>>> model_devi = calc_model_devi(coord, cell, atype, graphs)

```

## Submodules

## deepmd.infer.data\_modifier module

```

class deepmd.infer.data_modifier.DipoleChargeModifier(model_name: str, model_charge_map:
List[float], sys_charge_map: List[float],
ewald_h: float = 1, ewald_beta: float = 1)

```

Bases: *DeepDipole*

## Parameters

**model\_name**  
The model file for the DeepDipole model

**model\_charge\_map**  
Gives the amount of charge for the wfcc

**sys\_charge\_map**  
Gives the amount of charge for the real atoms

**ewald\_h**  
Grid spacing of the reciprocal part of Ewald sum. Unit: Å

**ewald\_beta**  
Splitting parameter of the Ewald sum. Unit: Å<sup>-1</sup>

## Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.

## Methods

<code>build_fv_graph()</code>	Build the computational graph for the force and virial inference.
<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coord, box, atype[, eval_fv])</code>	Evaluate the modification.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>modify_data(data, data_sys)</code>	Modify data.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`build_fv_graph()` → Tensor

Build the computational graph for the force and virial inference.

`eval(coord: ndarray, box: ndarray, atype: ndarray, eval_fv: bool = True)` → Tuple[ndarray, ndarray, ndarray]

Evaluate the modification.

Parameters

`coord`  
The coordinates of atoms

`box`  
The simulation region. PBC is assumed

`atype`  
The atom types

`eval_fv`  
Evaluate force and virial

Returns

`tot_e`  
The energy modification

`tot_f`  
The force modification

`tot_v`  
The virial modification

**modify\_data**(data: dict, data\_sys: DeepmdData) → None

Modify data.

Parameters

data

Internal data of DeepmdData. Be a dict, has the following keys - coord coordinates - box simulation box - type atom types - find\_energy tells if data has energy - find\_force tells if data has force - find\_virial tells if data has virial - energy energy - force force - virial virial

data\_sys

[DeepmdData] The data system.

### deepmd.infer.deep\_dipole module

```
class deepmd.infer.deep_dipole.DeepDipole(model_file: Path, load_prefix: str = 'load',
                                           default_tf_graph: bool = False, input_map:
                                           Optional[dict] = None, neighbor_list=None)
```

Bases: *DeepTensor*

Constructor.

Parameters

model\_file

[Path] The name of the frozen model file.

load\_prefix: str

The prefix in the load computational graph

default\_tf\_graph

[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

input\_map

[dict, optional] The input map for tf.import\_graph\_def. Only work with default tf graph

neighbor\_list

[ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

Attributes

**model\_type**

Get type of model.

**model\_version**

Get version of model.

**sess**

Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`get_dim_aparam()` → `int`

Unsupported in this model.

`get_dim_fparam()` → `int`

Unsupported in this model.

## deepmd.infer.deep\_dos module

```
class deepmd.infer.deep_dos.DeepDOS(model_file: Path, load_prefix: str = 'load', default_tf_graph:
    bool = False, auto_batch_size: Union[bool, int, AutoBatchSize]
    = True, input_map: Optional[dict] = None)
```

Bases: *DeepEval*

Constructor.

Parameters

`model_file`

[Path] The name of the frozen model file.

`load_prefix`: str

The prefix in the load computational graph

`default_tf_graph`

[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

`auto_batch_size`

[bool or int or AutomaticBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

`input_map`

[dict, optional] The input map for tf.import\_graph\_def. Only work with default tf graph

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

#### Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.

#### Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the dos, atom_dos by using this model.
<code>eval_descriptor(coords, cells, atom_types[, ...])</code>	Evaluate descriptors by using this DP.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Get the number (dimension) of atomic parameters of this DP.
<code>get_dim_fparam()</code>	Get the number (dimension) of frame parameters of this DP.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_numb_dos()</code>	Get the length of DOS output of this DP model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Unsupported in this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

**eval** (coords: `ndarray`, cells: `ndarray`, atom\_types: `List[int]`, atomic: `bool` = False, fparam: `Optional[ndarray]` = None, aparam: `Optional[ndarray]` = None, mixed\_type: `bool` = False) → `Tuple[ndarray, ...]`

Evaluate the dos, atom\_dos by using this model.

#### Parameters

**coords**  
The coordinates of atoms. The array should be of size nframes x natoms x 3

**cells**  
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

**atom\_types**  
The atom types The list should contain natoms ints

atomic

Calculate the atomic energy and virial

fparam

The frame parameter. The array can be of size : - nframes x dim\_fparam. - dim\_fparam. Then all frames are assumed to be provided with the same fparam.

aparam

The atomic parameter The array can be of size : - nframes x natoms x dim\_aparam. - natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. - dim\_aparam. Then all frames and atoms are provided with the same aparam.

mixed\_type

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

**dos**

The electron density of state.

**atom\_dos**

The atom-sited density of state. Only returned when atomic == True

**eval\_descriptor**(coords: `ndarray`, cells: `ndarray`, atom\_types: `List[int]`, fparam: `Optional[ndarray]` = None, aparam: `Optional[ndarray]` = None, efield: `Optional[ndarray]` = None, mixed\_type: `bool` = False) → array

Evaluate descriptors by using this DP.

Parameters

coords

The coordinates of atoms. The array should be of size nframes x natoms x 3

cells

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom\_types

The atom types The list should contain natoms ints

fparam

The frame parameter. The array can be of size : - nframes x dim\_fparam. - dim\_fparam. Then all frames are assumed to be provided with the same fparam.

aparam

The atomic parameter The array can be of size : - nframes x natoms x dim\_aparam. - natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. - dim\_aparam. Then all frames and atoms are provided with the same aparam.

efield

The external field on atoms. The array should be of size nframes x natoms x 3

mixed\_type

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

**descriptor**  
Descriptors.

**get\_dim\_aparam()** → `int`  
Get the number (dimension) of atomic parameters of this DP.

**get\_dim\_fparam()** → `int`  
Get the number (dimension) of frame parameters of this DP.

**get\_ntypes()** → `int`  
Get the number of atom types of this model.

**get\_numb\_dos()** → `int`  
Get the length of DOS output of this DP model.

**get\_rcut()** → `float`  
Get the cut-off radius of this model.

**get\_sel\_type()** → `List[int]`  
Unsupported in this model.

**get\_type\_map()** → `List[str]`  
Get the type map (element name of the atom types) of this model.

**load\_prefix:** `str`

#### **deepmd.infer.deep\_eval module**

```
class deepmd.infer.deep_eval.DeepEval(model_file: Path, load_prefix: str = 'load', default_tf_graph:
    bool = False, auto_batch_size: Union[bool, int,
    AutoBatchSize] = False, input_map: Optional[dict] = None,
    neighbor_list=None)
```

Bases: `object`

Common methods for DeepPot, DeepWFC, DeepPolar, ...

Parameters

**model\_file**  
[`Path`] The name of the frozen model file.

**load\_prefix:** `str`  
The prefix in the load computational graph

**default\_tf\_graph**  
[`bool`] If uses the default tf graph, otherwise build a new tf graph for evaluation

**auto\_batch\_size**  
[`bool` or `int` or `AutomaticBatchSize`, default: `False`] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

**input\_map**  
[`dict`, `optional`] The input map for `tf.import_graph_def`. Only work with default tf graph

**neighbor\_list**  
[`ase.neighborlist.NewPrimitiveNeighborList`, `optional`] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.



## Attributes

*model\_type*  
Get type of model.

*model\_version*  
Get version of model.

*sess*  
Get TF session.

## Methods

<i>build_neighbor_list</i> (coords, cell, atype, ...)	Make the mesh with neighbor list for a single frame.
<i>eval_typeembed</i> ()	Evaluate output of type embedding network by using this model.
<i>make_natoms_vec</i> (atom_types[, mixed_type])	Make the natom vector used by deepmd-kit.
<i>reverse_map</i> (vec, imap)	Reverse mapping of a vector according to the index map.
<i>sort_input</i> (coord, atom_type[, sel_atoms, ...])	Sort atoms in the system according their types.

**build\_neighbor\_list**(coords: `ndarray`, cell: `Optional[ndarray]`, atype: `ndarray`, imap: `ndarray`, neighbor\_list)

Make the mesh with neighbor list for a single frame.

## Parameters

coords  
[`np.ndarray`] The coordinates of atoms. Should be of shape [natoms, 3]

cell  
[`Optional[np.ndarray]`] The cell of the system. Should be of shape [3, 3]

atype  
[`np.ndarray`] The type of atoms. Should be of shape [natoms]

imap  
[`np.ndarray`] The index map of atoms. Should be of shape [natoms]

neighbor\_list  
[`ase.neighborlist.NewPrimitiveNeighborList`] ASE neighbor list. The following method or attribute will be used/set: bothways, self\_interaction, update, build, first\_neigh, pair\_second, offset\_vec.

## Returns

natoms\_vec  
[`np.ndarray`] The number of atoms. This tensor has the length of Ntypes + 2  
natoms[0]: nloc natoms[1]: nall natoms[i]: 2 <= i < Ntypes+2, number of type i atoms for nloc

coords  
[`np.ndarray`] The coordinates of atoms, including ghost atoms. Should be of shape [nframes, nall, 3]

atype  
[`np.ndarray`] The type of atoms, including ghost atoms. Should be of shape [nall]

`mesh`  
[`np.ndarray`] The mesh in `nei_mode=4`.  
`imap`  
[`np.ndarray`] The index map of atoms. Should be of shape [nall]  
`ghost_map`  
[`np.ndarray`] The index map of ghost atoms. Should be of shape [nghost]

`eval_typeebd()` → `ndarray`

Evaluate output of type embedding network by using this model.

Returns

`np.ndarray`  
The output of type embedding network. The shape is [ntypes, o\_size], where ntypes is the number of types, and o\_size is the number of nodes in the output layer.

Raises

`KeyError`  
If the model does not enable type embedding.

See also:

`deepmd.utils.type_embed.TypeEmbedNet`  
The type embedding network.

## Examples

Get the output of type embedding network of graph.pb:

```
>>> from deepmd.infer import DeepPotential
>>> dp = DeepPotential('graph.pb')
>>> dp.eval_typeebd()
```

`load_prefix:` `str`

`make_natoms_vec(atom_types: ndarray, mixed_type: bool = False)` → `ndarray`

Make the natom vector used by deepmd-kit.

Parameters

`atom_types`  
The type of atoms  
`mixed_type`  
Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

`natoms`  
The number of atoms. This tensor has the length of Ntypes + 2  
natoms[0]: number of local atoms  
natoms[1]: total number of atoms held by this processor  
natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

```

property model_type: str
    Get type of model.

    :type:str

property model_version: str
    Get version of model.

    Returns

    str
        version of model

static reverse_map(vec: ndarray, imap: List[int]) → ndarray
    Reverse mapping of a vector according to the index map.

    Parameters

    vec
        Input vector. Be of shape [nframes, natoms, -1]

    imap
        Index map. Be of shape [natoms]

    Returns

    vec_out
        Reverse mapped vector.

property sess: Session
    Get TF session.

static sort_input(coord: ndarray, atom_type: ndarray, sel_atoms: Optional[List[int]] = None,
    mixed_type: bool = False)

    Sort atoms in the system according their types.

    Parameters

    coord
        The coordinates of atoms. Should be of shape [nframes, natoms, 3]

    atom_type
        The type of atoms Should be of shape [natoms]

    sel_atoms
        The selected atoms by type

    mixed_type
        Whether to perform the mixed_type mode. If True, the input data has the
        mixed_type format (see doc/model/train_se_atten.md), in which frames in a sys-
        tem may have different natoms_vec(s), with the same nloc.

    Returns

    coord_out
        The coordinates after sorting

    atom_type_out
        The atom types after sorting

    idx_map
        The index mapping from the input to the output. For example coord_out = co-
        ord[:,idx_map,:]

```

**sel\_atom\_type**

Only output if sel\_atoms is not None The sorted selected atom types

**sel\_idx\_map**

Only output if sel\_atoms is not None The index mapping from the selected atoms to sorted selected atoms.

### deepmd.infer.deep\_polar module

```
class deepmd.infer.deep_polar.DeepGlobalPolar(model_file: str, load_prefix: str = 'load',
                                              default_tf_graph: bool = False,
                                              neighbor_list=None)
```

Bases: *DeepTensor*

Constructor.

Parameters

model\_file

[str] The name of the frozen model file.

load\_prefix: str

The prefix in the load computational graph

default\_tf\_graph

[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

neighbor\_list

[ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

Attributes

**model\_type**

Get type of model.

**model\_version**

Get version of model.

**sess**

Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

**eval** (coords: `ndarray`, cells: `ndarray`, atom\_types: `List[int]`, atomic: `bool` = False, fparam: `Optional[ndarray]` = None, aparam: `Optional[ndarray]` = None, efield: `Optional[ndarray]` = None) → `ndarray`

Evaluate the model.

Parameters

coords

The coordinates of atoms. The array should be of size nframes x natoms x 3

cells

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom\_types

The atom types The list should contain natoms ints

atomic

Not used in this model

fparam

Not used in this model

aparam

Not used in this model

efield

Not used in this model

Returns

**tensor**

The returned tensor If atomic == False then of size nframes x variable\_dof else of size nframes x natoms x variable\_dof

`get_dim_aparam() → int`

Unsupported in this model.

`get_dim_fparam() → int`

Unsupported in this model.

```
class deepmd.infer.deep_polar.DeepPolar(model_file: Path, load_prefix: str = 'load',
                                         default_tf_graph: bool = False, input_map: Optional[dict]
                                         = None, neighbor_list=None)
```

Bases: *DeepTensor*

Constructor.

Parameters

`model_file`

[`Path`] The name of the frozen model file.

`load_prefix: str`

The prefix in the load computational graph

`default_tf_graph`

[`bool`] If uses the default tf graph, otherwise build a new tf graph for evaluation

`input_map`

[`dict`, `optional`] The input map for `tf.import_graph_def`. Only work with default tf graph

`neighbor_list`

[`ase.neighborlist.NeighborList`, `optional`] The neighbor list object. If None, then build the native neighbor list.

Warning: For developers: DeepTensor initializer must be called at the end after `self.tensors` are modified because it uses the data in `self.tensors` dict. Do not change the order!

Attributes

`model_type`

Get type of model.

`model_version`

Get version of model.

`sess`

Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`get_dim_aparam() → int`  
Unsupported in this model.

`get_dim_fparam() → int`  
Unsupported in this model.

## deepmd.infer.deep\_pot module

```
class deepmd.infer.deep_pot.DeepPot(model_file: Path, load_prefix: str = 'load', default_tf_graph:
    bool = False, auto_batch_size: Union[bool, int, AutoBatchSize]
    = True, input_map: Optional[dict] = None,
    neighbor_list=None)
```

Bases: *DeepEval*

Constructor.

Parameters

`model_file`  
[Path] The name of the frozen model file.

`load_prefix: str`  
The prefix in the load computational graph

`default_tf_graph`  
[bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

`auto_batch_size`  
[bool or int or AutomaticBatchSize, default: True] If True, automatic batch size will be used. If int, it will be used as the initial batch size.

`input_map`  
[dict, optional] The input map for tf.import\_graph\_def. Only work with default tf graph

`neighbor_list`  
[`ase.neighborlist.NewPrimitiveNeighborList`, `optional`] The ASE neighbor list class to produce the neighbor list. If None, the neighbor list will be built natively in the model.

Warning: For developers: DeepTensor initializer must be called at the end after `self.tensors` are modified because it uses the data in `self.tensors` dict. Do not change the order!

## Examples

```
>>> from deepmd.infer import DeepPot
>>> import numpy as np
>>> dp = DeepPot('graph.pb')
>>> coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1,0,1]
>>> e, f, v = dp.eval(coord, cell, atype)
```

where `e`, `f` and `v` are predicted energy, force and virial of the system, respectively.

### Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.



## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the energy, force and virial by using this DP.
<code>eval_descriptor(coords, cells, atom_types[, ...])</code>	Evaluate descriptors by using this DP.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_descriptor_type()</code>	Get the descriptor type of this model.
<code>get_dim_aparam()</code>	Get the number (dimension) of atomic parameters of this DP.
<code>get_dim_fparam()</code>	Get the number (dimension) of frame parameters of this DP.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_ntypes_spin()</code>	Get the number of spin atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Unsupported in this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`eval` (coords: `ndarray`, cells: `ndarray`, atom\_types: `List[int]`, atomic: `bool` = `False`, fparam: `Optional[ndarray]` = `None`, aparam: `Optional[ndarray]` = `None`, efield: `Optional[ndarray]` = `None`, mixed\_type: `bool` = `False`) → `Tuple[ndarray, ...]`

Evaluate the energy, force and virial by using this DP.

Parameters

coords

The coordinates of atoms. The array should be of size nframes x natoms x 3

cells

The cell of the region. If `None` then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom\_types

The atom types The list should contain natoms ints

atomic

Calculate the atomic energy and virial

fparam

The frame parameter. The array can be of size : - nframes x dim\_fparam. - dim\_fparam. Then all frames are assumed to be provided with the same fparam.

aparam

The atomic parameter The array can be of size : - nframes x natoms x dim\_aparam. - natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. - dim\_aparam. Then all frames and atoms are provided with the same aparam.

**efield**

The external field on atoms. The array should be of size nframes x natoms x 3

**mixed\_type**

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

**energy**

The system energy.

**force**

The force on each atom

**virial**

The virial

**atom\_energy**

The atomic energy. Only returned when atomic == True

**atom\_virial**

The atomic virial. Only returned when atomic == True

**eval\_descriptor**(coords: ndarray, cells: ndarray, atom\_types: List[int], fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed\_type: bool = False) → array

Evaluate descriptors by using this DP.

Parameters

**coords**

The coordinates of atoms. The array should be of size nframes x natoms x 3

**cells**

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

**atom\_types**

The atom types The list should contain natoms ints

**fparam**

The frame parameter. The array can be of size : - nframes x dim\_fparam. - dim\_fparam. Then all frames are assumed to be provided with the same fparam.

**aparam**

The atomic parameter The array can be of size : - nframes x natoms x dim\_aparam. - natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. - dim\_aparam. Then all frames and atoms are provided with the same aparam.

**efield**

The external field on atoms. The array should be of size nframes x natoms x 3

**mixed\_type**

Whether to perform the mixed\_type mode. If True, the input data has the mixed\_type format (see doc/model/train\_se\_atten.md), in which frames in a system may have different natoms\_vec(s), with the same nloc.

Returns

```

        descriptor
            Descriptors.

get_descriptor_type() → List[int]
    Get the descriptor type of this model.

get_dim_aparam() → int
    Get the number (dimension) of atomic parameters of this DP.

get_dim_fparam() → int
    Get the number (dimension) of frame parameters of this DP.

get_ntypes() → int
    Get the number of atom types of this model.

get_ntypes_spin()
    Get the number of spin atom types of this model.

get_rcut() → float
    Get the cut-off radius of this model.

get_sel_type() → List[int]
    Unsupported in this model.

get_type_map() → List[str]
    Get the type map (element name of the atom types) of this model.

load_prefix: str

```

### deepmd.infer.deep\_tensor module

```

class deepmd.infer.deep_tensor.DeepTensor(model_file: Path, load_prefix: str = 'load',
                                           default_tf_graph: bool = False, input_map:
                                           Optional[dict] = None, neighbor_list=None)

```

Bases: *DeepEval*

Evaluates a tensor model.

Parameters

```

    model_file: str
        The name of the frozen model file.

    load_prefix: str
        The prefix in the load computational graph

    default_tf_graph
        [bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

    input_map
        [dict, optional] The input map for tf.import_graph_def. Only work with default
        tf graph

    neighbor_list
        [ase.neighborlist.NeighborList, optional] The neighbor list object. If None,
        then build the native neighbor list.

```

Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Get the number (dimension) of atomic parameters of this DP.
<code>get_dim_fparam()</code>	Get the number (dimension) of frame parameters of this DP.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

**eval**(coords: ndarray, cells: ndarray, atom\_types: List[int], atomic: bool = True, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, efield: Optional[ndarray] = None, mixed\_type: bool = False) → ndarray

Evaluate the model.

Parameters

coords

The coordinates of atoms. The array should be of size nframes x natoms x 3

cells

The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size nframes x 9

atom\_types

The atom types The list should contain natoms ints

atomic

If True (default), return the atomic tensor Otherwise return the global tensor

fparam

Not used in this model

`aparam`  
Not used in this model

`efield`  
Not used in this model

`mixed_type`  
Whether to perform the `mixed_type` mode. If True, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

Returns

**tensor**  
The returned tensor If `atomic == False` then of size `nframes x output_dim` else of size `nframes x natoms x output_dim`

`eval_full`(coords: `ndarray`, cells: `ndarray`, atom\_types: `List[int]`, atomic: `bool = False`, fparam: `Optional[array] = None`, aparam: `Optional[array] = None`, efield: `Optional[array] = None`, mixed\_type: `bool = False`) → `Tuple[ndarray, ...]`

Evaluate the model with interface similar to the energy model. Will return global tensor, component-wise force and virial and optionally atomic tensor and atomic virial.

Parameters

`coords`  
The coordinates of atoms. The array should be of size `nframes x natoms x 3`

`cells`  
The cell of the region. If None then non-PBC is assumed, otherwise using PBC. The array should be of size `nframes x 9`

`atom_types`  
The atom types The list should contain `natoms` ints

`atomic`  
Whether to calculate atomic tensor and virial

`fparam`  
Not used in this model

`aparam`  
Not used in this model

`efield`  
Not used in this model

`mixed_type`  
Whether to perform the `mixed_type` mode. If True, the input data has the `mixed_type` format (see `doc/model/train_se_atten.md`), in which frames in a system may have different `natoms_vec(s)`, with the same `nloc`.

Returns

**tensor**  
The global tensor. shape: `[nframes x nout]`

**force**  
The component-wise force (negative derivative) on each atom. shape: `[nframes x nout x natoms x 3]`

**virial**  
The component-wise virial of the tensor. shape: `[nframes x nout x 9]`

```

    atom_tensor
        The atomic tensor. Only returned when atomic == True shape: [nframes x natoms
        x nout]

    atom_virial
        The atomic virial. Only returned when atomic == True shape: [nframes x nout x
        natoms x 9]

get_dim_aparam() → int
    Get the number (dimension) of atomic parameters of this DP.

get_dim_fparam() → int
    Get the number (dimension) of frame parameters of this DP.

get_ntypes() → int
    Get the number of atom types of this model.

get_rcut() → float
    Get the cut-off radius of this model.

get_sel_type() → List[int]
    Get the selected atom types of this model.

get_type_map() → List[str]
    Get the type map (element name of the atom types) of this model.

tensors: ClassVar[Dict[str, str]] = {'t_box': 't_box:0', 't_coord': 't_coord:0',
't_mesh': 't_mesh:0', 't_natoms': 't_natoms:0', 't_ntypes':
'descript_attr/ntypes:0', 't_output_dim': 'model_attr/output_dim:0', 't_rcut':
'descript_attr/rcut:0', 't_sel_type': 'model_attr/sel_type:0', 't_tmap':
'model_attr/tmap:0', 't_type': 't_type:0'}

```

### deepmd.infer.deep\_wfc module

```

class deepmd.infer.deep_wfc.DeepWFC(model_file: Path, load_prefix: str = 'load', default_tf_graph:
    bool = False, input_map: Optional[dict] = None)

```

Bases: *DeepTensor*

Constructor.

Parameters

```

model_file
    [Path] The name of the frozen model file.

load_prefix: str
    The prefix in the load computational graph

default_tf_graph
    [bool] If uses the default tf graph, otherwise build a new tf graph for evaluation

input_map
    [dict, optional] The input map for tf.import_graph_def. Only work with default
    tf graph

```

Warning: For developers: DeepTensor initializer must be called at the end after self.tensors are modified because it uses the data in self.tensors dict. Do not change the order!

## Attributes

**model\_type**  
Get type of model.

**model\_version**  
Get version of model.

**sess**  
Get TF session.

## Methods

<code>build_neighbor_list(coords, cell, atype, ...)</code>	Make the mesh with neighbor list for a single frame.
<code>eval(coords, cells, atom_types[, atomic, ...])</code>	Evaluate the model.
<code>eval_full(coords, cells, atom_types[, ...])</code>	Evaluate the model with interface similar to the energy model.
<code>eval_typeebd()</code>	Evaluate output of type embedding network by using this model.
<code>get_dim_aparam()</code>	Unsupported in this model.
<code>get_dim_fparam()</code>	Unsupported in this model.
<code>get_ntypes()</code>	Get the number of atom types of this model.
<code>get_rcut()</code>	Get the cut-off radius of this model.
<code>get_sel_type()</code>	Get the selected atom types of this model.
<code>get_type_map()</code>	Get the type map (element name of the atom types) of this model.
<code>make_natoms_vec(atom_types[, mixed_type])</code>	Make the natom vector used by deepmd-kit.
<code>reverse_map(vec, imap)</code>	Reverse mapping of a vector according to the index map.
<code>sort_input(coord, atom_type[, sel_atoms, ...])</code>	Sort atoms in the system according their types.

`get_dim_aparam() → int`  
Unsupported in this model.

`get_dim_fparam() → int`  
Unsupported in this model.

`load_prefix: str`

**deepmd.infer.ewald\_recip module**

`class deepmd.infer.ewald_recip.EwaldRecp(hh, beta)`

Bases: `object`

Evaluate the reciprocal part of the Ewald sum.

## Methods

<code>eval(coord, charge, box)</code>	Evaluate.
---------------------------------------	-----------

`eval(coord: ndarray, charge: ndarray, box: ndarray) → Tuple[ndarray, ndarray, ndarray]`

Evaluate.

Parameters

`coord`  
The coordinates of atoms

`charge`  
The atomic charge

`box`  
The simulation region. PBC is assumed

Returns

`e`  
The energy

`f`  
The force

`v`  
The virial

## deepmd.infer.model\_devi module

`deepmd.infer.model_devi.calc_model_devi(coord, box, atype, models, fname=None, frequency=1, mixed_type=False, fparam: Optional[ndarray] = None, aparam: Optional[ndarray] = None, real_data: Optional[dict] = None, atomic: bool = False, relative: Optional[float] = None, relative_v: Optional[float] = None)`

Python interface to calculate model deviation.

Parameters

`coord`  
[`numpy.ndarray`, `n_frames` x `n_atoms` x 3] Coordinates of system to calculate

`box`  
[`numpy.ndarray` or `None`, `n_frames` x 3 x 3] Box to specify periodic boundary condition. If `None`, no pbc will be used

`atype`  
[`numpy.ndarray`, `n_atoms` x 1] Atom types

`models`  
[list of DeepPot models] Models used to evaluate deviation

`fname`  
[`str` or `None`] File to dump results, default `None`

`frequency`  
[`int`] Steps between frames (if the system is given by molecular dynamics engine), default 1



**mixed\_type**  
 [bool] Whether the input atype is in mixed\_type format or not

**fparam**  
 [numpy.ndarray] frame specific parameters

**aparam**  
 [numpy.ndarray] atomic specific parameters

**real\_data**  
 [dict, optional] real data to calculate RMS real error

**atomic**  
 [bool, default: False] If True, calculate the force model deviation of each atom.

**relative**  
 [float, default: None] If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

**relative\_v**  
 [float, default: None] If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

**Returns**

**model\_devi**  
 [numpy.ndarray, n\_frames x 8] Model deviation results. The first column is index of steps, the other 7 columns are max\_devi\_v, min\_devi\_v, avg\_devi\_v, max\_devi\_f, min\_devi\_f, avg\_devi\_f, devi\_e.

## Examples

```

>>> from deepmd.infer import calc_model_devi
>>> from deepmd.infer import DeepPot as DP
>>> import numpy as np
>>> coord = np.array([[1,0,0], [0,0,1.5], [1,0,3]]).reshape([1, -1])
>>> cell = np.diag(10 * np.ones(3)).reshape([1, -1])
>>> atype = [1,0,1]
>>> graphs = [DP("graph.000.pb"), DP("graph.001.pb")]
>>> model_devi = calc_model_devi(coord, cell, atype, graphs)

```

deepmd.infer.model\_devi.calc\_model\_devi\_e(es: ndarray, real\_e: Optional[ndarray] = None) → ndarray

Calculate model deviation of total energy per atom.

Here we don't use the atomic energy, as the decomposition of energy is arbitrary and not unique. There is no fitting target for atomic energy.

Parameters

es

[numpy.ndarray] size of 'n\_models x n\_frames x 1

real\_e

[numpy.ndarray] real energy, size of n\_frames x 1. If given, the RMS real error is calculated instead.

Returns

`max_devi_e`  
`[numpy.ndarray]` maximum deviation of energy

`deepmd.infer.model_devi.calc_model_devi_f`(fs: `ndarray`, real\_f: `Optional[ndarray]` = None, relative: `Optional[float]` = None, atomic: `Literal[False]` = False) → `Tuple[ndarray, ndarray, ndarray]`

`deepmd.infer.model_devi.calc_model_devi_f`(fs: `ndarray`, real\_f: `Optional[ndarray]` = None, relative: `Optional[float]` = None, \*, atomic: `Literal[True]`) → `Tuple[ndarray, ndarray, ndarray, ndarray]`

Calculate model deviation of force.

#### Parameters

`fs`  
`[numpy.ndarray]` size of `n_models x n_frames x n_atoms x 3`

`real_f`  
`[numpy.ndarray or None]` real force, size of `n_frames x n_atoms x 3`. If given, the RMS real error is calculated instead.

`relative`  
`[float, default: None]` If given, calculate the relative model deviation of force. The value is the level parameter for computing the relative model deviation of the force.

`atomic`  
`[bool, default: False]` Whether return deviation of force in all atoms

#### Returns

`max_devi_f`  
`[numpy.ndarray]` maximum deviation of force in all atoms

`min_devi_f`  
`[numpy.ndarray]` minimum deviation of force in all atoms

`avg_devi_f`  
`[numpy.ndarray]` average deviation of force in all atoms

`fs_devi`  
`[numpy.ndarray]` deviation of force in all atoms, returned if `atomic=True`

`deepmd.infer.model_devi.calc_model_devi_v`(vs: `ndarray`, real\_v: `Optional[ndarray]` = None, relative: `Optional[float]` = None) → `Tuple[ndarray, ndarray, ndarray]`

Calculate model deviation of virial.

#### Parameters

`vs`  
`[numpy.ndarray]` size of `n_models x n_frames x 9`

`real_v`  
`[numpy.ndarray]` real virial, size of `n_frames x 9`. If given, the RMS real error is calculated instead.

`relative`  
`[float, default: None]` If given, calculate the relative model deviation of virial. The value is the level parameter for computing the relative model deviation of the virial.

#### Returns

```

max_devi_v
    [numpy.ndarray] maximum deviation of virial in 9 elements

min_devi_v
    [numpy.ndarray] minimum deviation of virial in 9 elements

avg_devi_v
    [numpy.ndarray] average deviation of virial in 9 elements

deepmd.infer.model_devi.make_model_devi(*, models: list, system: str, set_prefix: str, output: str,
                                         frequency: int, real_error: bool = False, atomic: bool =
                                         False, relative: Optional[float] = None, relative_v:
                                         Optional[float] = None, **kwargs)

```

Make model deviation calculation.

Parameters

```

models
    [list] A list of paths of models to use for making model deviation

system
    [str] The path of system to make model deviation calculation

set_prefix
    [str] The set prefix of the system

output
    [str] The output file for model deviation results

frequency
    [int] The number of steps that elapse between writing coordinates in a trajectory by
    a MD engine (such as Gromacs / Lammps). This paramter is used to determine the
    index in the output file.

real_error
    [bool, default: False] If True, calculate the RMS real error instead of model devia-
    tion.

atomic
    [bool, default: False] If True, calculate the force model deviation of each atom.

relative
    [float, default: None] If given, calculate the relative model deviation of force. The
    value is the level parameter for computing the relative model deviation of the force.

relative_v
    [float, default: None] If given, calculate the relative model deviation of virial. The
    value is the level parameter for computing the relative model deviation of the virial.

**kwargs
    Arbitrary keyword arguments.

```

```

deepmd.infer.model_devi.write_model_devi_out(devi: ndarray, fname: str, header: str = "", atomic:
                                             bool = False)

```

Write output of model deviation.

Parameters

```

devi
    [numpy.ndarray] the first column is the steps index

```

fname  
     [str] the file name to dump

header  
     [str, default=""] the header to dump

atomic  
     [bool, default: False] whether atomic model deviation is printed

### deepmd.loggers package

Alias of deepmd\_utils.loggers for backward compatibility.

`deepmd.loggers.set_log_handles`(level: int, log\_path: Optional[Path] = None, mpi\_log: Optional[str] = None)

Set desired level for package loggers and add file handlers.

Parameters

level  
     [int] logging level

log\_path  
     [Optional[str]] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi\_log  
     [Optional[str], optional] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

**RuntimeError**

If the argument mpi\_log is specified, package mpi4py is not installed.

### Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

## References

<https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U>      <https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error>  
<https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu>

## Submodules

### deepmd.loggers.loggers module

Alias of `deepmd_utils.loggers.loggers` for backward compatibility.

`deepmd.loggers.loggers.set_log_handles`(level: `int`, log\_path: `Optional[Path]` = None, mpi\_log: `Optional[str]` = None)

Set desired level for package loggers and add file handlers.

Parameters

level

[`int`] logging level

log\_path

[`Optional[str]`] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi\_log

[`Optional[str]`, `optional`] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

`RuntimeError`

If the argument `mpi_log` is specified, package `mpi4py` is not installed.

## Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

## References

<https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U>      <https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error>  
<https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu>

## deepmd.loss package

```
class deepmd.loss.DOSLoss(starter_learning_rate: float, numb_dos: int = 500, start_pref_dos: float = 1.0, limit_pref_dos: float = 1.0, start_pref_cdf: float = 1000, limit_pref_cdf: float = 1.0, start_pref_adofs: float = 0.0, limit_pref_adofs: float = 0.0, start_pref_acdf: float = 0.0, limit_pref_acdf: float = 0.0, protect_value: float = 1e-08, log_fit: bool = False, **kwargs)
```

Bases: *Loss*

Loss function for DeepDOS models.

## Methods

<i>build</i> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<i>display_if_exist</i> (loss, find_property)	Display NaN if labeled property is not found.
<i>eval</i> (sess, feed_dict, natoms)	Eval the loss function.

**build**(learning\_rate, natoms, model\_dict, label\_dict, suffix)

Build the loss function graph.

Parameters

**learning\_rate**  
 [*tf.Tensor*] learning rate

**natoms**  
 [*tf.Tensor*] number of atoms

**model\_dict**  
 [*dict*[*str*, *tf.Tensor*]] A dictionary that maps model keys to tensors

**label\_dict**  
 [*dict*[*str*, *tf.Tensor*]] A dictionary that maps label keys to tensors

**suffix**  
 [*str*] suffix

Returns

*tf.Tensor*  
 the total squared loss

*dict*[*str*, *tf.Tensor*]  
 A dictionary that maps loss keys to more loss tensors

**eval**(sess, feed\_dict, natoms)

Eval the loss function.

Parameters

`sess`  
`[tf.Session]` TensorFlow session  
`feed_dict`  
`[dict[tf.placeholder, tf.Tensor]]` A dictionary that maps graph elements to values  
`natoms`  
`[tf.Tensor]` number of atoms  
Returns  
`dict`  
A dictionary that maps keys to values. It should contain key `natoms`  
**class** `deepmd.loss.EnerDipoleLoss`(`starter_learning_rate`: `float`, `start_pref_e`: `float` = 0.1, `limit_pref_e`:  
`float` = 1.0, `start_pref_ed`: `float` = 1.0, `limit_pref_ed`: `float` = 1.0)  
Bases: `Loss`

## Methods

<code>build</code> ( <code>learning_rate</code> , <code>natoms</code> , <code>model_dict</code> , ...)	Build the loss function graph.
<code>display_if_exist</code> ( <code>loss</code> , <code>find_property</code> )	Display NaN if labeled property is not found.
<code>eval</code> ( <code>sess</code> , <code>feed_dict</code> , <code>natoms</code> )	Eval the loss function.

**build**(`learning_rate`, `natoms`, `model_dict`, `label_dict`, `suffix`)

Build the loss function graph.

Parameters

`learning_rate`  
`[tf.Tensor]` learning rate  
`natoms`  
`[tf.Tensor]` number of atoms  
`model_dict`  
`[dict[str, tf.Tensor]]` A dictionary that maps model keys to tensors  
`label_dict`  
`[dict[str, tf.Tensor]]` A dictionary that maps label keys to tensors  
`suffix`  
`[str]` suffix

Returns

`tf.Tensor`  
the total squared loss  
`dict[str, tf.Tensor]`  
A dictionary that maps loss keys to more loss tensors

**eval**(`sess`, `feed_dict`, `natoms`)

Eval the loss function.

Parameters

`sess`  
`[tf.Session]` TensorFlow session

`feed_dict`  
`[dict[tf.placeholder, tf.Tensor]]` A dictionary that maps graph elements to values

`natoms`  
`[tf.Tensor]` number of atoms

Returns

`dict`  
 A dictionary that maps keys to values. It should contain key `natoms`

```
class deepmd.loss.EnerSpinLoss(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e:
    float = 1.0, start_pref_fr: float = 1000, limit_pref_fr: float = 1.0,
    start_pref_fm: float = 10000, limit_pref_fm: float = 10.0,
    start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float
    = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0,
    limit_pref_pf: float = 0.0, relative_f: Optional[float] = None,
    enable_atom_ener_coeff: bool = False, use_spin: Optional[list] =
    None)
```

Bases: *Loss*

## Methods

<code>build(learning_rate, natoms, model_dict, ...)</code>	Build the loss function graph.
<code>display_if_exist(loss, find_property)</code>	Display NaN if labeled property is not found.
<code>eval(sess, feed_dict, natoms)</code>	Eval the loss function.

<code>print_header</code>	
<code>print_on_training</code>	

`build(learning_rate, natoms, model_dict, label_dict, suffix)`

Build the loss function graph.

Parameters

`learning_rate`  
`[tf.Tensor]` learning rate

`natoms`  
`[tf.Tensor]` number of atoms

`model_dict`  
`[dict[str, tf.Tensor]]` A dictionary that maps model keys to tensors

`label_dict`  
`[dict[str, tf.Tensor]]` A dictionary that maps label keys to tensors

`suffix`  
`[str]` suffix

Returns

`tf.Tensor`  
 the total squared loss



```

    dict[str, tf.Tensor]
        A dictionary that maps loss keys to more loss tensors
eval(sess, feed_dict, natoms)
    Eval the loss function.

    Parameters
        sess
            [tf.Session] TensorFlow session
        feed_dict
            [dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values
        natoms
            [tf.Tensor] number of atoms

    Returns
        dict
            A dictionary that maps keys to values. It should contain key natoms

print_header()

print_on_training(tb_writer, cur_batch, sess, natoms, feed_dict_test, feed_dict_batch)

class deepmd.loss.EnerStdLoss(starter_learning_rate: float, start_pref_e: float = 0.02, limit_pref_e:
    float = 1.0, start_pref_f: float = 1000, limit_pref_f: float = 1.0,
    start_pref_v: float = 0.0, limit_pref_v: float = 0.0, start_pref_ae: float
    = 0.0, limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0,
    limit_pref_pf: float = 0.0, relative_f: Optional[float] = None,
    enable_atom_ener_coeff: bool = False, start_pref_gf: float = 0.0,
    limit_pref_gf: float = 0.0, numb_generalized_coord: int = 0, **kwargs)

```

Bases: *Loss*

Standard loss function for DP models.

Parameters

```

    starter_learning_rate
        [float] The learning rate at the start of the training.
    start_pref_e
        [float] The prefactor of energy loss at the start of the training.
    limit_pref_e
        [float] The prefactor of energy loss at the end of the training.
    start_pref_f
        [float] The prefactor of force loss at the start of the training.
    limit_pref_f
        [float] The prefactor of force loss at the end of the training.
    start_pref_v
        [float] The prefactor of virial loss at the start of the training.
    limit_pref_v
        [float] The prefactor of virial loss at the end of the training.
    start_pref_ae
        [float] The prefactor of atomic energy loss at the start of the training.

```

`limit_pref_ae`  
`[float]` The prefactor of atomic energy loss at the end of the training.

`start_pref_pf`  
`[float]` The prefactor of atomic prefactor force loss at the start of the training.

`limit_pref_pf`  
`[float]` The prefactor of atomic prefactor force loss at the end of the training.

`relative_f`  
`[float]` If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by `relative_f`

`enable_atom_ener_coeff`  
`[bool]` if true, the energy will be computed as  $\sum_i c_i E_i$

`start_pref_gf`  
`[float]` The prefactor of generalized force loss at the start of the training.

`limit_pref_gf`  
`[float]` The prefactor of generalized force loss at the end of the training.

`numb_generalized_coord`  
`[int]` The dimension of generalized coordinates.

`**kwargs`  
 Other keyword arguments.

## Methods

<code>build</code> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<code>display_if_exist</code> (loss, find_property)	Display NaN if labeled property is not found.
<code>eval</code> (sess, feed_dict, natoms)	Eval the loss function.

**build**(learning\_rate, natoms, model\_dict, label\_dict, suffix)

Build the loss function graph.

Parameters

`learning_rate`

`[tf.Tensor]` learning rate

`natoms`

`[tf.Tensor]` number of atoms

`model_dict`

`[dict[str, tf.Tensor]]` A dictionary that maps model keys to tensors

`label_dict`

`[dict[str, tf.Tensor]]` A dictionary that maps label keys to tensors

`suffix`

`[str]` suffix

Returns

`tf.Tensor`

the total squared loss

`dict[str, tf.Tensor]`

A dictionary that maps loss keys to more loss tensors

`eval(sess, feed_dict, natoms)`

Eval the loss function.

Parameters

`sess`

`[tf.Session]` TensorFlow session

`feed_dict`

`[dict[tf.placeholder, tf.Tensor]]` A dictionary that maps graph elements to values

`natoms`

`[tf.Tensor]` number of atoms

Returns

`dict`

A dictionary that maps keys to values. It should contain key `natoms`

`class deepmd.loss.TensorLoss(jdata, **kwarg)`

Bases: `Loss`

Loss function for tensorial properties.

## Methods

<code>build(learning_rate, natoms, model_dict, ...)</code>	Build the loss function graph.
<code>display_if_exist(loss, find_property)</code>	Display NaN if labeled property is not found.
<code>eval(sess, feed_dict, natoms)</code>	Eval the loss function.

`build(learning_rate, natoms, model_dict, label_dict, suffix)`

Build the loss function graph.

Parameters

`learning_rate`

`[tf.Tensor]` learning rate

`natoms`

`[tf.Tensor]` number of atoms

`model_dict`

`[dict[str, tf.Tensor]]` A dictionary that maps model keys to tensors

`label_dict`

`[dict[str, tf.Tensor]]` A dictionary that maps label keys to tensors

`suffix`

`[str]` suffix

Returns

`tf.Tensor`

the total squared loss

`dict[str, tf.Tensor]`

A dictionary that maps loss keys to more loss tensors

```
eval(sess, feed_dict, natoms)
```

Eval the loss function.

Parameters

`sess`

[`tf.Session`] TensorFlow session

`feed_dict`

[`dict`[`tf.placeholder`, `tf.Tensor`]] A dictionary that maps graph elements to values

`natoms`

[`tf.Tensor`] number of atoms

Returns

`dict`

A dictionary that maps keys to values. It should contain key `natoms`

## Submodules

### deepmd.loss.dos module

```
class deepmd.loss.dos.DOSLoss(starter_learning_rate: float, numb_dos: int = 500, start_pref_dos: float = 1.0, limit_pref_dos: float = 1.0, start_pref_cdf: float = 1000, limit_pref_cdf: float = 1.0, start_pref_adof: float = 0.0, limit_pref_adof: float = 0.0, start_pref_acdf: float = 0.0, limit_pref_acdf: float = 0.0, protect_value: float = 1e-08, log_fit: bool = False, **kwargs)
```

Bases: `Loss`

Loss function for DeepDOS models.

## Methods

<code>build</code> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<code>display_if_exist</code> (loss, find_property)	Display NaN if labeled property is not found.
<code>eval</code> (sess, feed_dict, natoms)	Eval the loss function.

```
build(learning_rate, natoms, model_dict, label_dict, suffix)
```

Build the loss function graph.

Parameters

`learning_rate`

[`tf.Tensor`] learning rate

`natoms`

[`tf.Tensor`] number of atoms

`model_dict`

[`dict`[`str`, `tf.Tensor`]] A dictionary that maps model keys to tensors

`label_dict`

[`dict`[`str`, `tf.Tensor`]] A dictionary that maps label keys to tensors

suffix  
[`str`] suffix

Returns

`tf.Tensor`  
the total squared loss

`dict[str, tf.Tensor]`  
A dictionary that maps loss keys to more loss tensors

`eval(sess, feed_dict, natoms)`  
Eval the loss function.

Parameters

sess  
[`tf.Session`] TensorFlow session

feed\_dict  
[`dict[tf.placeholder, tf.Tensor]`] A dictionary that maps graph elements to values

natoms  
[`tf.Tensor`] number of atoms

Returns

`dict`  
A dictionary that maps keys to values. It should contain key natoms

### deepmd.loss.ener module

`class deepmd.loss.ener.EnerDipoleLoss(starter_learning_rate: float, start_pref_e: float = 0.1, limit_pref_e: float = 1.0, start_pref_ed: float = 1.0, limit_pref_ed: float = 1.0)`

Bases: `Loss`

### Methods

<code>build(learning_rate, natoms, model_dict, ...)</code>	Build the loss function graph.
<code>display_if_exist(loss, find_property)</code>	Display NaN if labeled property is not found.
<code>eval(sess, feed_dict, natoms)</code>	Eval the loss function.

`build(learning_rate, natoms, model_dict, label_dict, suffix)`

Build the loss function graph.

Parameters

learning\_rate  
[`tf.Tensor`] learning rate

natoms  
[`tf.Tensor`] number of atoms

model\_dict  
[`dict[str, tf.Tensor]`] A dictionary that maps model keys to tensors

```

    label_dict
        [dict[str, tf.Tensor]] A dictionary that maps label keys to tensors
    suffix
        [str] suffix
Returns
    tf.Tensor
        the total squared loss
    dict[str, tf.Tensor]
        A dictionary that maps loss keys to more loss tensors
eval(sess, feed_dict, natoms)
    Eval the loss function.
Parameters
    sess
        [tf.Session] TensorFlow session
    feed_dict
        [dict[tf.placeholder, tf.Tensor]] A dictionary that maps graph elements to values
    natoms
        [tf.Tensor] number of atoms
Returns
    dict
        A dictionary that maps keys to values. It should contain key natoms
class deepmd.loss.ener.EnerSpinLoss(starter_learning_rate: float, start_pref_e: float = 0.02,
                                     limit_pref_e: float = 1.0, start_pref_fr: float = 1000,
                                     limit_pref_fr: float = 1.0, start_pref_fm: float = 10000,
                                     limit_pref_fm: float = 10.0, start_pref_v: float = 0.0,
                                     limit_pref_v: float = 0.0, start_pref_ae: float = 0.0,
                                     limit_pref_ae: float = 0.0, start_pref_pf: float = 0.0,
                                     limit_pref_pf: float = 0.0, relative_f: Optional[float] = None,
                                     enable_atom_ener_coeff: bool = False, use_spin: Optional[list]
                                     = None)

```

Bases: *Loss*

## Methods

<i>build</i> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<i>display_if_exist</i> (loss, find_property)	Display NaN if labeled property is not found.
<i>eval</i> (sess, feed_dict, natoms)	Eval the loss function.

print_header	
print_on_training	

**build**(learning\_rate, natoms, model\_dict, label\_dict, suffix)

Build the loss function graph.

Parameters

learning\_rate

[`tf.Tensor`] learning rate

natoms

[`tf.Tensor`] number of atoms

model\_dict

[`dict[str, tf.Tensor]`] A dictionary that maps model keys to tensors

label\_dict

[`dict[str, tf.Tensor]`] A dictionary that maps label keys to tensors

suffix

[`str`] suffix

Returns

`tf.Tensor`

the total squared loss

`dict[str, tf.Tensor]`

A dictionary that maps loss keys to more loss tensors

**eval**(sess, feed\_dict, natoms)

Eval the loss function.

Parameters

sess

[`tf.Session`] TensorFlow session

feed\_dict

[`dict[tf.placeholder, tf.Tensor]`] A dictionary that maps graph elements to values

natoms

[`tf.Tensor`] number of atoms

Returns

`dict`

A dictionary that maps keys to values. It should contain key natoms

**print\_header**()

**print\_on\_training**(tb\_writer, cur\_batch, sess, natoms, feed\_dict\_test, feed\_dict\_batch)

```
class deepmd.loss.ener.EnerStdLoss(starter_learning_rate: float, start_pref_e: float = 0.02,
                                   limit_pref_e: float = 1.0, start_pref_f: float = 1000, limit_pref_f:
                                   float = 1.0, start_pref_v: float = 0.0, limit_pref_v: float = 0.0,
                                   start_pref_ae: float = 0.0, limit_pref_ae: float = 0.0,
                                   start_pref_pf: float = 0.0, limit_pref_pf: float = 0.0, relative_f:
                                   Optional[float] = None, enable_atom_ener_coeff: bool = False,
                                   start_pref_gf: float = 0.0, limit_pref_gf: float = 0.0,
                                   numb_generalized_coord: int = 0, **kwargs)
```

Bases: *Loss*

Standard loss function for DP models.

## Parameters

`starter_learning_rate`  
[float] The learning rate at the start of the training.

`start_pref_e`  
[float] The prefactor of energy loss at the start of the training.

`limit_pref_e`  
[float] The prefactor of energy loss at the end of the training.

`start_pref_f`  
[float] The prefactor of force loss at the start of the training.

`limit_pref_f`  
[float] The prefactor of force loss at the end of the training.

`start_pref_v`  
[float] The prefactor of virial loss at the start of the training.

`limit_pref_v`  
[float] The prefactor of virial loss at the end of the training.

`start_pref_ae`  
[float] The prefactor of atomic energy loss at the start of the training.

`limit_pref_ae`  
[float] The prefactor of atomic energy loss at the end of the training.

`start_pref_pf`  
[float] The prefactor of atomic prefactor force loss at the start of the training.

`limit_pref_pf`  
[float] The prefactor of atomic prefactor force loss at the end of the training.

`relative_f`  
[float] If provided, relative force error will be used in the loss. The difference of force will be normalized by the magnitude of the force in the label with a shift given by `relative_f`

`enable_atom_ener_coeff`  
[bool] if true, the energy will be computed as  $\sum_i c_i E_i$

`start_pref_gf`  
[float] The prefactor of generalized force loss at the start of the training.

`limit_pref_gf`  
[float] The prefactor of generalized force loss at the end of the training.

`numb_generalized_coord`  
[int] The dimension of generalized coordinates.

`**kwargs`  
Other keyword arguments.



## Methods

<code>build(learning_rate, natoms, model_dict, ...)</code>	Build the loss function graph.
<code>display_if_exist(loss, find_property)</code>	Display NaN if labeled property is not found.
<code>eval(sess, feed_dict, natoms)</code>	Eval the loss function.

**build**(learning\_rate, natoms, model\_dict, label\_dict, suffix)

Build the loss function graph.

Parameters

learning\_rate

[`tf.Tensor`] learning rate

natoms

[`tf.Tensor`] number of atoms

model\_dict

[`dict[str, tf.Tensor]`] A dictionary that maps model keys to tensors

label\_dict

[`dict[str, tf.Tensor]`] A dictionary that maps label keys to tensors

suffix

[`str`] suffix

Returns

`tf.Tensor`

the total squared loss

[`dict[str, tf.Tensor]`]

A dictionary that maps loss keys to more loss tensors

**eval**(sess, feed\_dict, natoms)

Eval the loss function.

Parameters

sess

[`tf.Session`] TensorFlow session

feed\_dict

[`dict[tf.placeholder, tf.Tensor]`] A dictionary that maps graph elements to values

natoms

[`tf.Tensor`] number of atoms

Returns

`dict`

A dictionary that maps keys to values. It should contain key natoms

**deepmd.loss.loss module**

```
class deepmd.loss.loss.Loss
```

Bases: `object`

The abstract class for the loss function.

**Methods**

<i>build</i> (learning_rate, natoms, model_dict, ...)	Build the loss function graph.
<i>display_if_exist</i> (loss, find_property)	Display NaN if labeled property is not found.
<i>eval</i> (sess, feed_dict, natoms)	Eval the loss function.

```
abstract build(learning_rate: Tensor, natoms: Tensor, model_dict: Dict[str, Tensor], label_dict: Dict[str, Tensor], suffix: str) → Tuple[Tensor, Dict[str, Tensor]]
```

Build the loss function graph.

Parameters

learning\_rate

[`tf.Tensor`] learning rate

natoms

[`tf.Tensor`] number of atoms

model\_dict

[`dict[str, tf.Tensor]`] A dictionary that maps model keys to tensors

label\_dict

[`dict[str, tf.Tensor]`] A dictionary that maps label keys to tensors

suffix

[`str`] suffix

Returns

`tf.Tensor`

the total squared loss

[`dict[str, tf.Tensor]`]

A dictionary that maps loss keys to more loss tensors

```
static display_if_exist(loss: Tensor, find_property: float) → Tensor
```

Display NaN if labeled property is not found.

Parameters

loss

[`tf.Tensor`] the loss tensor

find\_property

[`float`] whether the property is found

```
abstract eval(sess: Session, feed_dict: Dict[placeholder, Tensor], natoms: Tensor) → dict
```

Eval the loss function.

Parameters

`sess`  
`[tf.Session]` TensorFlow session  
`feed_dict`  
`[dict[tf.placeholder, tf.Tensor]]` A dictionary that maps graph elements to values  
`natoms`  
`[tf.Tensor]` number of atoms  
Returns  
`dict`  
A dictionary that maps keys to values. It should contain key `natoms`

### deepmd.loss.tensor module

`class deepmd.loss.tensor.TensorLoss(jdata, **kwarg)`

Bases: *Loss*

Loss function for tensorial properties.

#### Methods

<code>build(learning_rate, natoms, model_dict, ...)</code>	Build the loss function graph.
<code>display_if_exist(loss, find_property)</code>	Display NaN if labeled property is not found.
<code>eval(sess, feed_dict, natoms)</code>	Eval the loss function.

`build(learning_rate, natoms, model_dict, label_dict, suffix)`

Build the loss function graph.

Parameters

`learning_rate`  
`[tf.Tensor]` learning rate  
`natoms`  
`[tf.Tensor]` number of atoms  
`model_dict`  
`[dict[str, tf.Tensor]]` A dictionary that maps model keys to tensors  
`label_dict`  
`[dict[str, tf.Tensor]]` A dictionary that maps label keys to tensors  
`suffix`  
`[str]` suffix

Returns

`tf.Tensor`  
the total squared loss  
`dict[str, tf.Tensor]`  
A dictionary that maps loss keys to more loss tensors

```
eval(sess, feed_dict, natoms)
```

Eval the loss function.

Parameters

`sess`

[`tf.Session`] TensorFlow session

`feed_dict`

[`dict`[`tf.placeholder`, `tf.Tensor`]] A dictionary that maps graph elements to values

`natoms`

[`tf.Tensor`] number of atoms

Returns

`dict`

A dictionary that maps keys to values. It should contain key `natoms`

### deepmd.model package

```
class deepmd.model.DOSModel(*args, **kwargs)
```

Bases: *StandardModel*

DOS model.

Parameters

`descriptor`

Descriptor

`fitting_net`

Fitting net

`type_embedding`

Type embedding net

`type_map`

Mapping atom type to the name (str) of the type. For example `type_map[1]` gives the name of the type 1.

`data_stat_nbatch`

Number of frames used for data statistic

`data_stat_protect`

Protect parameter for atomic energy regression

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_numb_aparam()</code>	Get the number of atomic parameters.
<code>get_numb_dos()</code>	Get the number of gridpoints in energy space.
<code>get_numb_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

**build**(coord\_, atype\_, natoms, box, mesh, input\_dict, frz\_model=None, ckpt\_meta: Optional[str] = None, suffix="", reuse=None)

Build the model.

Parameters

coord\_  
[[tf.Tensor](#)] The coordinates of atoms

atype\_  
[[tf.Tensor](#)] The atom types of atoms

natoms  
[[tf.Tensor](#)] The number of atoms

box  
[[tf.Tensor](#)] The box vectors

mesh  
[[tf.Tensor](#)] The mesh vectors

input\_dict  
[[dict](#)] The input dict

frz\_model  
[[str](#), optional] The path to the frozen model

ckpt\_meta  
[[str](#), optional] The path prefix of the checkpoint and meta files

suffix  
[[str](#), optional] The suffix of the scope

```

reuse
    [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns
    dict
        The output dict

data_stat(data)
    Data statistics.

get_ntypes()
    Get the number of types.

get_numb_aparam() → int
    Get the number of atomic parameters.

get_numb_dos()
    Get the number of gridpoints in energy space.

get_numb_fparam() → int
    Get the number of frame parameters.

get_rcut()
    Get cutoff radius of the model.

get_type_map()
    Get the type map.

init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str
               = '') → None
    Init the embedding net variables with the given frozen model.

Parameters
    graph
        [tf.Graph] The input frozen model graph
    graph_def
        [tf.GraphDef] The input frozen model graph_def
    model_type
        [str] the type of the model
    suffix
        [str] suffix to name scope

model_type = 'dos'

class deepmd.model.DipoleModel(*args, **kwargs)
    Bases: TensorModel

```

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

```
class deepmd.model.EnerModel(*args, **kwargs)
```

Bases: *StandardModel*

Energy model.

Parameters

descriptor

Descriptor

fitting\_net

Fitting net

type\_embedding

Type embedding net

type\_map

Mapping atom type to the name (str) of the type. For example `type_map[1]` gives the name of the type 1.

data\_stat\_nbatch

Number of frames used for data statistic

data\_stat\_protect

Protect parameter for atomic energy regression

use\_srtab

The table for the short-range pairwise interaction added on top of DP. The table is

a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

`smin_alpha`

The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by `softmin`. This parameter is the decaying parameter in the `softmin`. It is only required when `use_srtab` is provided.

`sw_rmin`

The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

`sw_rmin`

The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

`srtab_add_bias`

[`bool`] Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when `use_srtab` is provided.

`spin`

`spin`

`data_stat_nsampl`

The number of training samples in a system to compute and change the energy bias.

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the <code>feed_dict</code> for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_numb_aparam()</code>	Get the number of atomic parameters.
<code>get_numb_dos()</code>	Get the number of gridpoints in energy space.
<code>get_numb_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.



natoms_match	
natoms_not_match	

**build**(coord\_, atype\_, natoms, box, mesh, input\_dict, frz\_model=None, ckpt\_meta: Optional[str] = None, suffix='', reuse=None)

Build the model.

Parameters

coord\_  
[[tf.Tensor](#)] The coordinates of atoms

atype\_  
[[tf.Tensor](#)] The atom types of atoms

natoms  
[[tf.Tensor](#)] The number of atoms

box  
[[tf.Tensor](#)] The box vectors

mesh  
[[tf.Tensor](#)] The mesh vectors

input\_dict  
[[dict](#)] The input dict

frz\_model  
[[str](#), optional] The path to the frozen model

ckpt\_meta  
[[str](#), optional] The path prefix of the checkpoint and meta files

suffix  
[[str](#), optional] The suffix of the scope

reuse  
[[bool](#) or [tf.AUTO\\_REUSE](#), optional] Whether to reuse the variables

Returns

[dict](#)  
The output dict

**change\_energy\_bias**(data: [DeepmdDataSystem](#), frozen\_model: [str](#), origin\_type\_map: [list](#), full\_type\_map: [str](#), bias\_shift: [str](#) = 'delta') → [None](#)

Change the energy bias according to the input data and the pretrained model.

Parameters

data  
[[DeepmdDataSystem](#)] The training data.

frozen\_model  
[[str](#)] The path file of frozen model.

origin\_type\_map  
[[list](#)] The original type\_map in dataset, they are targets to change the energy bias.

full\_type\_map  
[[str](#)] The full type\_map in pretrained model

```
    bias_shift
        [str] The mode for changing energy bias : ['delta', 'statistic'] 'delta' : perform
        predictions on energies of target dataset,
        and do least square on the errors to obtain the target shift as bias.
        'statistic' : directly use the statistic energy bias in the target dataset.

data_stat(data)
    Data statistics.

get_ntypes()
    Get the number of types.

get_numb_aparam() → int
    Get the number of atomic parameters.

get_numb_fparam() → int
    Get the number of frame parameters.

get_rcut()
    Get cutoff radius of the model.

get_type_map()
    Get the type map.

init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str
               = '') → None
    Init the embedding net variables with the given frozen model.
    Parameters
        graph
            [tf.Graph] The input frozen model graph
        graph_def
            [tf.GraphDef] The input frozen model graph_def
        model_type
            [str] the type of the model
        suffix
            [str] suffix to name scope

model_type = 'ener'

natoms_match(force, natoms)

natoms_not_match(force, natoms, atype)

class deepmd.model.GlobalPolarModel(*args, **kwargs)
    Bases: TensorModel
```

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

```
class deepmd.model.MultiModel(*args, **kwargs)
```

Bases: *Model*

Multi-task model.

Parameters

`descriptor`  
Descriptor

`fitting_net_dict`  
Dictionary of fitting nets

`fitting_type_dict`  
deprecated argument

`type_embedding`  
Type embedding net

`type_map`  
Mapping atom type to the name (str) of the type. For example `type_map[1]` gives the name of the type 1.

`data_stat_nbatch`  
Number of frames used for data statistic

`data_stat_protect`  
Protect parameter for atomic energy regression

**use\_srtab**

The table for the short-range pairwise interaction added on top of DP. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

**smin\_alpha**

The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when use\_srtab is provided.

**sw\_rmin**

The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use\_srtab is provided.

**sw\_rmin**

The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when use\_srtab is provided.

**Methods**

<i>build</i> (coord_, atype_, natoms, box, mesh, ...)	Build the model.
<i>build_descrpt</i> (coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
<i>build_type_embedding</i> (ntypes[, frz_model, ...])	Build the type embedding part of the model.
<i>change_energy_bias</i> (data, frozen_model, ...)	Change the energy bias according to the input data and the pretrained model.
<i>data_stat</i> (data)	Data statistics.
<i>enable_compression</i> ([suffix])	Enable compression.
<i>enable_mixed_precision</i> (mixed_prec)	Enable mixed precision for the model.
<i>get_class_by_input</i> (input)	Get the class by input data.
<i>get_feed_dict</i> (coord_, atype_, natoms, box, ...)	Generate the feed_dict for current descriptor.
<i>get_fitting</i> ()	Get the fitting(s).
<i>get_loss</i> (loss, lr)	Get the loss function(s).
<i>get_ntypes</i> ()	Get the number of types.
<i>get_numb_aparam</i> ()	Get the number of atomic parameters.
<i>get_numb_dos</i> ()	Get the number of gridpoints in energy space.
<i>get_numb_fparam</i> ()	Get the number of frame parameters.
<i>get_rcut</i> ()	Get cutoff radius of the model.
<i>get_type_map</i> ()	Get the type map.
<i>init_variables</i> (graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

**build**(coord\_, atype\_, natoms, box, mesh, input\_dict, frz\_model=None, ckpt\_meta: [Optional\[str\]](#) = None, suffix="", reuse=None)

Build the model.

Parameters

coord\_

[\[tf.Tensor\]](#) The coordinates of atoms

```

    atype_
        [tf.Tensor] The atom types of atoms
    natoms
        [tf.Tensor] The number of atoms
    box
        [tf.Tensor] The box vectors
    mesh
        [tf.Tensor] The mesh vectors
    input_dict
        [dict] The input dict
    frz_model
        [str, optional] The path to the frozen model
    ckpt_meta
        [str, optional] The path prefix of the checkpoint and meta files
    suffix
        [str, optional] The suffix of the scope
    reuse
        [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables
Returns
    dict
        The output dict
data_stat(data)
    Data statistics.
enable_mixed_precision(mixed_prec: dict)
    Enable mixed precision for the model.
Parameters
    mixed_prec
        [dict] The mixed precision config
get_fitting() → dict
    Get the fitting(s).
get_loss(loss: dict, lr: dict) → Dict[str, Loss]
    Get the loss function(s).
get_ntypes()
    Get the number of types.
get_numb_aparam() → dict
    Get the number of atomic parameters.
get_numb_dos() → dict
    Get the number of gridpoints in energy space.
get_numb_fparam() → dict
    Get the number of frame parameters.

```

`get_rcut()`

Get cutoff radius of the model.

`get_type_map()`

Get the type map.

`init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '') → None`

Init the embedding net variables with the given frozen model.

Parameters

`graph`

[`tf.Graph`] The input frozen model graph

`graph_def`

[`tf.GraphDef`] The input frozen model graph\_def

`model_type`

[`str`] the type of the model

`suffix`

[`str`] suffix to name scope

`model_type = 'multi_task'`

`classmethod update_sel(global_jdata: dict, local_jdata: dict)`

Update the selection and perform neighbor statistics.

Parameters

`global_jdata`

[`dict`] The global data, containing the training section

`local_jdata`

[`dict`] The local data refer to the current class

`class deepmd.model.PolarModel(*args, **kwargs)`

Bases: *TensorModel*

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

`class deepmd.model.WFCModel(*args, **kwargs)`

Bases: *TensorModel*

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

## Submodules

### deepmd.model.dos module

`class deepmd.model.dos.DOSModel(*args, **kwargs)`

Bases: *StandardModel*

DOS model.

Parameters

descriptor  
Descriptor

fitting\_net  
Fitting net

type\_embedding  
Type embedding net

type\_map  
Mapping atom type to the name (str) of the type. For example `type_map[1]` gives the name of the type 1.



`data_stat_nbatch`  
Number of frames used for data statistic

`data_stat_protect`  
Protect parameter for atomic energy regression

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_numb_aparam()</code>	Get the number of atomic parameters.
<code>get_numb_dos()</code>	Get the number of gridpoints in energy space.
<code>get_numb_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

**build**(coord\_, atype\_, natoms, box, mesh, input\_dict, frz\_model=None, ckpt\_meta: [Optional\[str\]](#) = None, suffix="", reuse=None)

Build the model.

Parameters

`coord_`  
[\[tf.Tensor\]](#) The coordinates of atoms

`atype_`  
[\[tf.Tensor\]](#) The atom types of atoms

`natoms`  
[\[tf.Tensor\]](#) The number of atoms

`box`  
[\[tf.Tensor\]](#) The box vectors

`mesh`  
[\[tf.Tensor\]](#) The mesh vectors

`input_dict`  
[\[dict\]](#) The input dict

```
    frz_model
        [str, optional] The path to the frozen model
    ckpt_meta
        [str, optional] The path prefix of the checkpoint and meta files
    suffix
        [str, optional] The suffix of the scope
    reuse
        [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables
Returns
    dict
        The output dict
data_stat(data)
    Data statistics.
get_ntypes()
    Get the number of types.
get_numb_aparam() → int
    Get the number of atomic parameters.
get_numb_dos()
    Get the number of gridpoints in energy space.
get_numb_fparam() → int
    Get the number of frame parameters.
get_rcut()
    Get cutoff radius of the model.
get_type_map()
    Get the type map.
init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str
              = '') → None
    Init the embedding net variables with the given frozen model.
Parameters
    graph
        [tf.Graph] The input frozen model graph
    graph_def
        [tf.GraphDef] The input frozen model graph_def
    model_type
        [str] the type of the model
    suffix
        [str] suffix to name scope
model_type = 'dos'
```

**deepmd.model.ener module**

```
class deepmd.model.ener.EnerModel(*args, **kwargs)
```

Bases: *StandardModel*

Energy model.

Parameters

descriptor

Descriptor

fitting\_net

Fitting net

type\_embedding

Type embedding net

type\_map

Mapping atom type to the name (str) of the type. For example `type_map[1]` gives the name of the type 1.

data\_stat\_nbatch

Number of frames used for data statistic

data\_stat\_protect

Protect parameter for atomic energy regression

use\_srtab

The table for the short-range pairwise interaction added on top of DP. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin\_alpha

The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when `use_srtab` is provided.

sw\_rmin

The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

sw\_rmin

The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

srtab\_add\_bias

[bool] Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when `use_srtab` is provided.

spin

spin

data\_stat\_nsample

The number of training samples in a system to compute and change the energy bias.

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_numb_aparam()</code>	Get the number of atomic parameters.
<code>get_numb_dos()</code>	Get the number of gridpoints in energy space.
<code>get_numb_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>natoms_match</code>	
<code>natoms_not_match</code>	

`build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: Optional[str] = None, suffix='', reuse=None)`

Build the model.

Parameters

`coord_`  
[`tf.Tensor`] The coordinates of atoms

`atype_`  
[`tf.Tensor`] The atom types of atoms

`natoms`  
[`tf.Tensor`] The number of atoms

`box`  
[`tf.Tensor`] The box vectors

`mesh`  
[`tf.Tensor`] The mesh vectors

`input_dict`  
[`dict`] The input dict

`frz_model`  
[`str`, `optional`] The path to the frozen model

`ckpt_meta`  
 [`str`, optional] The path prefix of the checkpoint and meta files  
`suffix`  
 [`str`, optional] The suffix of the scope  
`reuse`  
 [`bool` or `tf.AUTO_REUSE`, optional] Whether to reuse the variables  
 Returns  
`dict`  
 The output dict  
**change\_energy\_bias**(data: `DeepmdDataSystem`, frozen\_model: `str`, origin\_type\_map: `list`,  
 full\_type\_map: `str`, bias\_shift: `str` = 'delta') → `None`  
 Change the energy bias according to the input data and the pretrained model.  
 Parameters  
`data`  
 [`DeepmdDataSystem`] The training data.  
`frozen_model`  
 [`str`] The path file of frozen model.  
`origin_type_map`  
 [`list`] The original type\_map in dataset, they are targets to change the energy bias.  
`full_type_map`  
 [`str`] The full type\_map in pretrained model  
`bias_shift`  
 [`str`] The mode for changing energy bias : ['delta', 'statistic'] 'delta' : perform  
 predictions on energies of target dataset,  
 and do least square on the errors to obtain the target shift as bias.  
 'statistic' : directly use the statistic energy bias in the target dataset.  
**data\_stat**(data)  
 Data statistics.  
**get\_ntypes**()  
 Get the number of types.  
**get\_numb\_aparam**() → `int`  
 Get the number of atomic parameters.  
**get\_numb\_fparam**() → `int`  
 Get the number of frame parameters.  
**get\_rcut**()  
 Get cutoff radius of the model.  
**get\_type\_map**()  
 Get the type map.  
**init\_variables**(graph: `Graph`, graph\_def: `GraphDef`, model\_type: `str` = 'original\_model', suffix: `str`  
 = '') → `None`  
 Init the embedding net variables with the given frozen model.

## Parameters

`graph`  
[`tf.Graph`] The input frozen model graph

`graph_def`  
[`tf.GraphDef`] The input frozen model graph\_def

`model_type`  
[`str`] the type of the model

`suffix`  
[`str`] suffix to name scope

`model_type = 'ener'`

`natoms_match(force, natoms)`

`natoms_not_match(force, natoms, atype)`

**deepmd.model.frozen module**

```
class deepmd.model.frozen.FrozenModel(*args, **kwargs)
```

Bases: *Model*

Load model from a frozen model, which cannot be trained.

## Parameters

`model_file`  
[`str`] The path to the frozen model

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor, input\_dict: dict, frz\_model: Optional[str] = None, ckpt\_meta: Optional[str] = None, suffix: str = "", reuse: Optional[Union[bool, Enum]] = None) → dict

Build the model.

Parameters

`coord_`  
[[tf.Tensor](#)] The coordinates of atoms

`atype_`  
[[tf.Tensor](#)] The atom types of atoms

`natoms`  
[[tf.Tensor](#)] The number of atoms

`box`  
[[tf.Tensor](#)] The box vectors

`mesh`  
[[tf.Tensor](#)] The mesh vectors

`input_dict`  
[dict] The input dict

`frz_model`  
[str, optional] The path to the frozen model

`ckpt_meta`  
[str, optional] The path prefix of the checkpoint and meta files

suffix  
[**str**, optional] The suffix of the scope

reuse  
[**bool** or `tf.AUTO_REUSE`, optional] Whether to reuse the variables

Returns

**dict**  
The output dict

**data\_stat**(data)  
Data statistics.

**enable\_compression**(suffix: **str** = "") → **None**  
Enable compression.

Parameters

suffix  
[**str**] suffix to name scope

**get\_fitting**() → **Union**[**Fitting**, **dict**]  
Get the fitting(s).

**get\_loss**(loss: **dict**, lr) → **Optional**[**Union**[**Loss**, **dict**]]  
Get the loss function(s).

**get\_ntypes**() → **int**  
Get the number of types.

**get\_rcut**()  
Get cutoff radius of the model.

**get\_type\_map**() → **list**  
Get the type map.

**init\_variables**(graph: **Graph**, graph\_def: **GraphDef**, model\_type: **str** = 'original\_model', suffix: **str** = "") → **None**  
Init the embedding net variables with the given frozen model.

Parameters

graph  
[**tf.Graph**] The input frozen model graph

graph\_def  
[**tf.GraphDef**] The input frozen model graph\_def

model\_type  
[**str**] the type of the model

suffix  
[**str**] suffix to name scope

**classmethod update\_sel**(global\_jdata: **dict**, local\_jdata: **dict**)  
Update the selection and perform neighbor statistics.

Parameters

global\_jdata  
[**dict**] The global data, containing the training section



local\_jdata  
 [dict] The local data refer to the current class

## deepmd.model.linear module

**class** deepmd.model.linear.LinearEnergyModel(\*args, \*\*kwargs)

Bases: *LinearModel*

Linear energy model make linear combinations of several existing energy models.

## Methods

<i>build</i> (coord_, atype_, natoms, box, mesh, ...)	Build the model.
<i>build_descrpt</i> (coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
<i>build_type_embedding</i> (ntypes[, frz_model, ...])	Build the type embedding part of the model.
<i>change_energy_bias</i> (data, frozen_model, ...)	Change the energy bias according to the input data and the pretrained model.
<i>data_stat</i> (data)	Data statistics.
<i>enable_compression</i> ([suffix])	Enable compression.
<i>enable_mixed_precision</i> (mixed_prec)	Enable mixed precision for the model.
<i>get_class_by_input</i> (input)	Get the class by input data.
<i>get_feed_dict</i> (coord_, atype_, natoms, box, ...)	Generate the feed_dict for current descriptor.
<i>get_fitting</i> ()	Get the fitting(s).
<i>get_loss</i> (loss, lr)	Get the loss function(s).
<i>get_num_aparam</i> ()	Get the number of atomic parameters.
<i>get_num_dos</i> ()	Get the number of gridpoints in energy space.
<i>get_num_fparam</i> ()	Get the number of frame parameters.
<i>get_rcut</i> ()	Get cutoff radius of the model.
<i>get_type_map</i> ()	Get the type map.
<i>init_variables</i> (graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

get_ntypes	
------------	--

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor, input\_dict: dict, frz\_model: Optional[str] = None, ckpt\_meta: Optional[str] = None, suffix: str = "", reuse: Optional[Union[bool, Enum]] = None) → dict

Build the model.

Parameters

coord\_  
 [tf.Tensor] The coordinates of atoms

atype\_  
 [tf.Tensor] The atom types of atoms

natoms  
 [tf.Tensor] The number of atoms

`box`  
    [`tf.Tensor`] The box vectors

`mesh`  
    [`tf.Tensor`] The mesh vectors

`input_dict`  
    [`dict`] The input dict

`frz_model`  
    [`str`, optional] The path to the frozen model

`ckpt_meta`  
    [`str`, optional] The path prefix of the checkpoint and meta files

`suffix`  
    [`str`, optional] The suffix of the scope

`reuse`  
    [`bool` or `tf.AUTO_REUSE`, optional] Whether to reuse the variables

Returns

`dict`  
    The output dict

`model_type = 'ener'`

`class deepmd.model.linear.LinearModel(*args, **kwargs)`

Bases: *Model*

Linear model make linear combinations of several existing models.

Parameters

`models`  
    [`list[dict]`] A list of models to be combined.

`weights`  
    [`list[float]` or `str`] If the type is `list[float]`, a list of weights for each model. If “mean”, the weights are set to be `1 / len(models)`. If “sum”, the weights are set to be 1.

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

get_ntypes	
------------	--

**data\_stat**(data)

Data statics.

**enable\_compression**(suffix: str = '') → None

Enable compression.

Parameters

suffix

[str] suffix to name scope

**get\_fitting**() → Union[Fitting, dict]

Get the fitting(s).

**get\_loss**(loss: dict, lr) → Optional[Union[Loss, dict]]

Get the loss function(s).

**get\_ntypes**() → int

Get the number of types.

**get\_rcut**()

Get cutoff radius of the model.

**get\_type\_map**() → list

Get the type map.

```
init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '') → None
```

Init the embedding net variables with the given frozen model.

Parameters

graph  
[[tf.Graph](#)] The input frozen model graph

graph\_def  
[[tf.GraphDef](#)] The input frozen model graph\_def

model\_type  
[[str](#)] the type of the model

suffix  
[[str](#)] suffix to name scope

```
classmethod update_sel(global_jdata: dict, local_jdata: dict)
```

Update the selection and perform neighbor statistics.

Parameters

global\_jdata  
[[dict](#)] The global data, containing the training section

local\_jdata  
[[dict](#)] The local data refer to the current class

## deepmd.model.model module

```
class deepmd.model.model.Model(*args, **kwargs)
```

Bases: [ABC](#)

Abstract base model.

Parameters

type\_embedding  
Type embedding net

type\_map  
Mapping atom type to the name (str) of the type. For example type\_map[1] gives the name of the type 1.

data\_stat\_nbatch  
Number of frames used for data statistic

data\_bias\_nsample  
The number of training samples in a system to compute and change the energy bias.

data\_stat\_protect  
Protect parameter for atomic energy regression

use\_srtab  
The table for the short-range pairwise interaction added on top of DP. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

`smin_alpha`  
 The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when `use_srtab` is provided.

`sw_rmin`  
 The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

`sw_rmin`  
 The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

`srtab_add_bias`  
 [bool] Whether add energy bias from the statistics of the data to short-range tabulated atomic energy. It only takes effect when `use_srtab` is provided.

`spin`  
 spin

`compress`  
 Compression information for internal use

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data staticis.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_numb_aparam()</code>	Get the number of atomic parameters.
<code>get_numb_dos()</code>	Get the number of gridpoints in energy space.
<code>get_numb_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

**abstract** `build(coord_: Tensor, atype_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor, input_dict: dict, frz_model: Optional[str] = None, ckpt_meta: Optional[str] = None, suffix: str = "", reuse: Optional[Union[bool, Enum]] = None)`

Build the model.

Parameters

`coord_`  
[`tf.Tensor`] The coordinates of atoms

`atype_`  
[`tf.Tensor`] The atom types of atoms

`natoms`  
[`tf.Tensor`] The number of atoms

`box`  
[`tf.Tensor`] The box vectors

`mesh`  
[`tf.Tensor`] The mesh vectors

`input_dict`  
[`dict`] The input dict

`frz_model`  
[`str`, `optional`] The path to the frozen model

`ckpt_meta`  
[`str`, `optional`] The path prefix of the checkpoint and meta files

`suffix`  
[`str`, `optional`] The suffix of the scope

`reuse`  
[`bool` or `tf.AUTO_REUSE`, `optional`] Whether to reuse the variables

Returns

`dict`  
The output dict

**build\_descript**(`coord_`: `Tensor`, `atype_`: `Tensor`, `natoms`: `Tensor`, `box`: `Tensor`, `mesh`: `Tensor`,  
`input_dict`: `dict`, `frz_model`: `Optional[str]` = `None`, `ckpt_meta`: `Optional[str]` = `None`,  
`suffix`: `str` = "", `reuse`: `Optional[Union[bool, Enum]]` = `None`)

Build the descriptor part of the model.

Parameters

`coord_`  
[`tf.Tensor`] The coordinates of atoms

`atype_`  
[`tf.Tensor`] The atom types of atoms

`natoms`  
[`tf.Tensor`] The number of atoms

`box`  
[`tf.Tensor`] The box vectors

`mesh`  
[`tf.Tensor`] The mesh vectors

`input_dict`  
[`dict`] The input dict

`frz_model`  
[`str`, `optional`] The path to the frozen model

ckpt\_meta  
 [str, optional] The path prefix of the checkpoint and meta files  
 suffix  
 [str, optional] The suffix of the scope  
 reuse  
 [bool or tf.AUTO\_REUSE, optional] Whether to reuse the variables

Returns

tf.Tensor  
 The descriptor tensor

**build\_type\_embedding**(ntypes: int, frz\_model: Optional[str] = None, ckpt\_meta: Optional[str] = None, suffix: str = "", reuse: Optional[Union[bool, Enum]] = None) → Tensor

Build the type embedding part of the model.

Parameters

ntypes  
 [int] The number of types  
 frz\_model  
 [str, optional] The path to the frozen model  
 ckpt\_meta  
 [str, optional] The path prefix of the checkpoint and meta files  
 suffix  
 [str, optional] The suffix of the scope  
 reuse  
 [bool or tf.AUTO\_REUSE, optional] Whether to reuse the variables

Returns

tf.Tensor  
 The type embedding tensor

**change\_energy\_bias**(data: DeepmdDataSystem, frozen\_model: str, origin\_type\_map: list, full\_type\_map: str, bias\_shift: str = 'delta') → None

Change the energy bias according to the input data and the pretrained model.

Parameters

data  
 [DeepmdDataSystem] The training data.  
 frozen\_model  
 [str] The path file of frozen model.  
 origin\_type\_map  
 [list] The original type\_map in dataset, they are targets to change the energy bias.  
 full\_type\_map  
 [str] The full type\_map in pretrained model  
 bias\_shift  
 [str] The mode for changing energy bias : ['delta', 'statistic'] 'delta' : perform predictions on energies of target dataset, and do least square on the errors to obtain the target shift as bias.

‘statistic’ : directly use the statistic energy bias in the target dataset.

**abstract data\_stat**(data: dict)

Data statistics.

**enable\_compression**(suffix: str = "")

Enable compression.

Parameters

suffix

[str] suffix to name scope

**enable\_mixed\_precision**(mixed\_prec: dict)

Enable mixed precision for the model.

Parameters

mixed\_prec

[dict] The mixed precision config

**classmethod get\_class\_by\_input**(input: dict)

Get the class by input data.

Parameters

input

[dict] The input data

**get\_feed\_dict**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor,  
\*\*kwargs) → Dict[str, Tensor]

Generate the feed\_dict for current descriptor.

Parameters

coord\_

[tf.Tensor] The coordinate of atoms

atype\_

[tf.Tensor] The type of atoms

natoms

[tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2  
natoms[0]: number of local atoms natoms[1]: total number of atoms held by this  
processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

box

[tf.Tensor] The box. Can be generated by deepmd.model.make\_stat\_input

mesh

[tf.Tensor] For historical reasons, only the length of the Tensor matters. if size of  
mesh == 6, pbc is assumed. if size of mesh == 0, no-pbc is assumed.

\*\*kwargs

[dict] The additional arguments

Returns

feed\_dict

[Dict[str, tf.Tensor]] The output feed\_dict of current descriptor



```

abstract get_fitting() → Union[Fitting, dict]
    Get the fitting(s).

abstract get_loss(loss: dict, lr) → Optional[Union[Loss, dict]]
    Get the loss function(s).

abstract get_ntypes() → int
    Get the number of types.

get_numb_aparam() → Union[int, dict]
    Get the number of atomic parameters.

get_numb_dos() → Union[int, dict]
    Get the number of gridpoints in energy space.

get_numb_fparam() → Union[int, dict]
    Get the number of frame parameters.

abstract get_rcut() → float
    Get cutoff radius of the model.

get_type_map() → list
    Get the type map.

init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str
                = '') → None
    Init the embedding net variables with the given frozen model.

    Parameters
        graph
            [tf.Graph] The input frozen model graph
        graph_def
            [tf.GraphDef] The input frozen model graph_def
        model_type
            [str] the type of the model
        suffix
            [str] suffix to name scope

abstract classmethod update_sel(global_jdata: dict, local_jdata: dict) → dict
    Update the selection and perform neighbor statistics.

    Parameters
        global_jdata
            [dict] The global data, containing the training section
        local_jdata
            [dict] The local data refer to the current class

    Returns
        dict
            The updated local data

```

## Notes

Do not modify the input data without copying it.

```
class deepmd.model.model.StandardModel(*args, **kwargs)
```

Bases: *Model*

Standard model, which must contain a descriptor and a fitting.

Parameters

descriptor  
[Union[dict, Descriptor]] The descriptor

fitting\_net  
[Union[dict, Fitting]] The fitting network

type\_embedding  
[dict, optional] The type embedding

type\_map  
[list of dict, optional] The type map

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

```
enable_compression(suffix: str = '')
```

Enable compression.

Parameters

suffix  
[str] suffix to name scope

```

enable_mixed_precision(mixed_prec: dict)
    Enable mixed precision for the model.

    Parameters
        mixed_prec
            [dict] The mixed precision config

get_fitting() → Union[Fitting, dict]
    Get the fitting(s).

get_loss(loss: dict, lr) → Union[Loss, dict]
    Get the loss function(s).

get_ntypes() → int
    Get the number of types.

get_rcut() → float
    Get cutoff radius of the model.

classmethod update_sel(global_jdata: dict, local_jdata: dict)
    Update the selection and perform neighbor statistics.

    Parameters
        global_jdata
            [dict] The global data, containing the training section
        local_jdata
            [dict] The local data refer to the current class

```

### deepmd.model.model\_stat module

Alias for backward compatibility.

```
deepmd.model.model_stat.make_stat_input(data, nbatches, merge_sys=True)
```

Pack data for statistics.

Parameters

```

data
    The data

nbatches
    [int] The number of batches

merge_sys
    [bool (True)] Merge system data

```

Returns

```

all_stat:
    A dictionary of list of list storing data for stat. if merge_sys == False data can be
    accessed by
        all_stat[key][sys_idx][batch_idx][frame_idx]

    else merge_sys == True can be accessed by
        all_stat[key][batch_idx][frame_idx]

```

```
deepmd.model.model_stat.merge_sys_stat(all_stat)
```

### deepmd.model.multi module

```
class deepmd.model.multi.MultiModel(*args, **kwargs)
```

Bases: *Model*

Multi-task model.

Parameters

descriptor

Descriptor

fitting\_net\_dict

Dictionary of fitting nets

fitting\_type\_dict

deprecated argument

type\_embedding

Type embedding net

type\_map

Mapping atom type to the name (str) of the type. For example `type_map[1]` gives the name of the type 1.

data\_stat\_nbatch

Number of frames used for data statistic

data\_stat\_protect

Protect parameter for atomic energy regression

use\_srtab

The table for the short-range pairwise interaction added on top of DP. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

smin\_alpha

The short-range tabulated interaction will be swithed according to the distance of the nearest neighbor. This distance is calculated by softmin. This parameter is the decaying parameter in the softmin. It is only required when `use_srtab` is provided.

sw\_rmin

The lower boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

sw\_rmin

The upper boundary of the interpolation between short-range tabulated interaction and DP. It is only required when `use_srtab` is provided.

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_numb_aparam()</code>	Get the number of atomic parameters.
<code>get_numb_dos()</code>	Get the number of gridpoints in energy space.
<code>get_numb_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

**build**(coord\_, atype\_, natoms, box, mesh, input\_dict, frz\_model=None, ckpt\_meta: Optional[str] = None, suffix="", reuse=None)

Build the model.

Parameters

coord\_  
[[tf.Tensor](#)] The coordinates of atoms

atype\_  
[[tf.Tensor](#)] The atom types of atoms

natoms  
[[tf.Tensor](#)] The number of atoms

box  
[[tf.Tensor](#)] The box vectors

mesh  
[[tf.Tensor](#)] The mesh vectors

input\_dict  
[[dict](#)] The input dict

frz\_model  
[[str](#), optional] The path to the frozen model

ckpt\_meta  
[[str](#), optional] The path prefix of the checkpoint and meta files

suffix  
[[str](#), optional] The suffix of the scope

reuse  
[`bool` or `tf.AUTO_REUSE`, optional] Whether to reuse the variables

Returns

`dict`  
The output dict

`data_stat(data)`  
Data staticis.

`enable_mixed_precision(mixed_prec: dict)`  
Enable mixed precision for the model.

Parameters

mixed\_prec  
[`dict`] The mixed precision config

`get_fitting()` → `dict`  
Get the fitting(s).

`get_loss(loss: dict, lr: dict)` → `Dict[str, Loss]`  
Get the loss function(s).

`get_ntypes()`  
Get the number of types.

`get_numb_aparam()` → `dict`  
Get the number of atomic parameters.

`get_numb_dos()` → `dict`  
Get the number of gridpoints in energy space.

`get_numb_fparam()` → `dict`  
Get the number of frame parameters.

`get_rcut()`  
Get cutoff radius of the model.

`get_type_map()`  
Get the type map.

`init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '')` → `None`  
Init the embedding net variables with the given frozen model.

Parameters

graph  
[`tf.Graph`] The input frozen model graph

graph\_def  
[`tf.GraphDef`] The input frozen model graph\_def

model\_type  
[`str`] the type of the model

suffix  
[`str`] suffix to name scope

```
model_type = 'multi_task'
```

```
classmethod update_sel(global_jdata: dict, local_jdata: dict)
```

Update the selection and perform neighbor statistics.

Parameters

global\_jdata

[dict] The global data, containing the training section

local\_jdata

[dict] The local data refer to the current class

## deepmd.model.pairtab module

```
class deepmd.model.pairtab.PairTabModel(*args, **kwargs)
```

Bases: *Model*

Pairwise tabulation energy model.

This model can be used to tabulate the pairwise energy between atoms for either short-range or long-range interactions, such as D3, LJ, ZBL, etc. It should not be used alone, but rather as one submodel of a linear (sum) model, such as DP+D3.

Do not put the model on the first model of a linear model, since the linear model fetches the type map from the first model.

At this moment, the model does not smooth the energy at the cutoff radius, so one needs to make sure the energy has been smoothed to zero.

Parameters

tab\_file

[str] The path to the tabulation file.

rcut

[float] The cutoff radius

sel

[int or list[int]] The maximum number of atoms in the cut-off radius

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor, input\_dict: dict, frz\_model: Optional[str] = None, ckpt\_meta: Optional[str] = None, suffix: str = "", reuse: Optional[Union[bool, Enum]] = None)

Build the model.

Parameters

`coord_`  
[`tf.Tensor`] The coordinates of atoms

`atype_`  
[`tf.Tensor`] The atom types of atoms

`natoms`  
[`tf.Tensor`] The number of atoms

`box`  
[`tf.Tensor`] The box vectors

`mesh`  
[`tf.Tensor`] The mesh vectors

`input_dict`  
[dict] The input dict

`frz_model`  
[str, optional] The path to the frozen model

`ckpt_meta`  
[str, optional] The path prefix of the checkpoint and meta files



suffix  
 [str, optional] The suffix of the scope

reuse  
 [bool or tf.AUTO\_REUSE, optional] Whether to reuse the variables

Returns  
 dict  
 The output dict

**data\_stat**(data: dict)  
 Data statistics.

**enable\_compression**(suffix: str = '') → None  
 Enable compression.

Parameters  
 suffix  
 [str] suffix to name scope

**get\_fitting**() → Union[Fitting, dict]  
 Get the fitting(s).

**get\_loss**(loss: dict, lr) → Optional[Union[Loss, dict]]  
 Get the loss function(s).

**get\_ntypes**() → int  
 Get the number of types.

**get\_rcut**() → float  
 Get cutoff radius of the model.

**init\_variables**(graph: Graph, graph\_def: GraphDef, model\_type: str = 'original\_model', suffix: str = '') → None  
 Init the embedding net variables with the given frozen model.

Parameters  
 graph  
 [tf.Graph] The input frozen model graph  
 graph\_def  
 [tf.GraphDef] The input frozen model graph\_def  
 model\_type  
 [str] the type of the model  
 suffix  
 [str] suffix to name scope

**model\_type = 'ener'**

**classmethod update\_sel**(global\_jdata: dict, local\_jdata: dict) → dict  
 Update the selection and perform neighbor statistics.

Parameters  
 global\_jdata  
 [dict] The global data, containing the training section

local\_jdata  
[dict] The local data refer to the current class

Returns

dict  
The updated local data

### Notes

Do not modify the input data without copying it.

## deepmd.model.pairwise\_dprc module

`class deepmd.model.pairwise_dprc.PairwiseDPRC(*args, **kwargs)`

Bases: *Model*

Pairwise Deep Potential - Range Correction.

### Methods

<i>build</i> (coord_, atype_, natoms, box_, mesh, ...)	Build the model.
<i>build_descrpt</i> (coord_, atype_, natoms, box, ...)	Build the descriptor part of the model.
<i>build_type_embedding</i> (ntypes[, frz_model, ...])	Build the type embedding part of the model.
<i>change_energy_bias</i> (data, frozen_model, ...)	Change the energy bias according to the input data and the pretrained model.
<i>data_stat</i> (data)	Data statistics.
<i>enable_compression</i> ([suffix])	Enable compression.
<i>enable_mixed_precision</i> (mixed_prec)	Enable mixed precision for the model.
<i>get_class_by_input</i> (input)	Get the class by input data.
<i>get_feed_dict</i> (coord_, atype_, natoms, box, ...)	Generate the feed_dict for current descriptor.
<i>get_fitting</i> ()	Get the fitting(s).
<i>get_loss</i> (loss, lr)	Get the loss function(s).
<i>get_ntypes</i> ()	Get the number of types.
<i>get_numb_aparam</i> ()	Get the number of atomic parameters.
<i>get_numb_dos</i> ()	Get the number of gridpoints in energy space.
<i>get_numb_fparam</i> ()	Get the number of frame parameters.
<i>get_rcut</i> ()	Get cutoff radius of the model.
<i>get_type_map</i> ()	Get the type map.
<i>init_variables</i> (graph, graph_def[, ...])	Init the embedding net variables with the given frozen model.
<i>update_sel</i> (global_jdata, local_jdata)	Update the selection and perform neighbor statistics.

**build**(coord\_: Tensor, atype\_: Tensor, natoms: Tensor, box\_: Tensor, mesh: Tensor, input\_dict: dict, frz\_model=None, ckpt\_meta: Optional[str] = None, suffix: str = "", reuse: Optional[bool] = None)

Build the model.

Parameters

```

coord_
    [tf.Tensor] The coordinates of atoms

atype_
    [tf.Tensor] The atom types of atoms

natoms
    [tf.Tensor] The number of atoms

box
    [tf.Tensor] The box vectors

mesh
    [tf.Tensor] The mesh vectors

input_dict
    [dict] The input dict

frz_model
    [str, optional] The path to the frozen model

ckpt_meta
    [str, optional] The path prefix of the checkpoint and meta files

suffix
    [str, optional] The suffix of the scope

reuse
    [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables

Returns
    dict
        The output dict

data_stat(data)
    Data statistics.

enable_compression(suffix: str = "") → None
    Enable compression.

Parameters
    suffix
        [str] suffix to name scope

get_feed_dict(coord_: Tensor, atype_: Tensor, natoms: Tensor, box: Tensor, mesh: Tensor,
               **kwargs) → Dict[str, Tensor]
    Generate the feed_dict for current descriptor.

Parameters
    coord_
        [tf.Tensor] The coordinate of atoms

    atype_
        [tf.Tensor] The type of atoms

    natoms
        [tf.Tensor] The number of atoms. This tensor has the length of Ntypes + 2
        natoms[0]: number of local atoms natoms[1]: total number of atoms held by this
        processor natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

```

`box`  
[`tf.Tensor`] The box. Can be generated by `deepmd.model.make_stat_input`

`mesh`  
[`tf.Tensor`] For historical reasons, only the length of the Tensor matters. if size of `mesh == 6`, pbc is assumed. if size of `mesh == 0`, no-pbc is assumed.

`aparam`  
[`tf.Tensor`] The parameters of the descriptor

`**kwargs`  
[`dict`] The keyword arguments

Returns

`feed_dict`  
[`dict[str, tf.Tensor]`] The output `feed_dict` of current descriptor

`get_fitting()` → `Union[str, dict]`  
Get the fitting(s).

`get_loss(loss: dict, lr) → Union[Loss, dict]`  
Get the loss function(s).

`get_ntypes()` → `int`  
Get the number of types.

`get_rcut()`  
Get cutoff radius of the model.

`init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str = '')` → `None`  
Init the embedding net variables with the given frozen model.

Parameters

`graph`  
[`tf.Graph`] The input frozen model graph

`graph_def`  
[`tf.GraphDef`] The input frozen model graph\_def

`model_type`  
[`str`] the type of the model

`suffix`  
[`str`] suffix to name scope

`model_type = 'ener'`

`classmethod update_sel(global_jdata: dict, local_jdata: dict)`  
Update the selection and perform neighbor statistics.

Parameters

`global_jdata`  
[`dict`] The global data, containing the training section

`local_jdata`  
[`dict`] The local data refer to the current class

`deepmd.model.pairwise_dprc.gather_placeholder`(params: Tensor, indices: Tensor, placeholder: float = 0.0, \*\*kwargs) → Tensor

Call `tf.gather` but allow indices to contain placeholders (-1).

## deepmd.model.tensor module

`class deepmd.model.tensor.DipoleModel(*args, **kwargs)`

Bases: *TensorModel*

### Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

`class deepmd.model.tensor.GlobalPolarModel(*args, **kwargs)`

Bases: *TensorModel*

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

```
class deepmd.model.tensor.PolarModel(*args, **kwargs)
```

Bases: *TensorModel*

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

```
class deepmd.model.tensor.TensorModel(*args, **kwargs)
```

Bases: *StandardModel*

Tensor model.

Parameters

`tensor_name`  
Name of the tensor.

`descriptor`  
Descriptor

`fitting_net`  
Fitting net

`type_embedding`  
Type embedding net

`type_map`  
Mapping atom type to the name (str) of the type. For example `type_map[1]` gives the name of the type 1.

`data_stat_nbatch`  
Number of frames used for data statistic

`data_stat_protect`  
Protect parameter for atomic energy regression

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

`build(coord_, atype_, natoms, box, mesh, input_dict, frz_model=None, ckpt_meta: Optional[str] = None, suffix='', reuse=None)`

Build the model.

Parameters

`coord_`  
[`tf.Tensor`] The coordinates of atoms

`atype_`  
[`tf.Tensor`] The atom types of atoms

`natoms`  
[`tf.Tensor`] The number of atoms

`box`  
[`tf.Tensor`] The box vectors

`mesh`  
[`tf.Tensor`] The mesh vectors

`input_dict`  
[`dict`] The input dict

`frz_model`  
[`str`, `optional`] The path to the frozen model



---

```

    ckpt_meta
        [str, optional] The path prefix of the checkpoint and meta files
    suffix
        [str, optional] The suffix of the scope
    reuse
        [bool or tf.AUTO_REUSE, optional] Whether to reuse the variables
Returns
    dict
        The output dict
data_stat(data)
    Data statistics.
get_ntypes()
    Get the number of types.
get_out_size()
get_rcut()
    Get cutoff radius of the model.
get_sel_type()
get_type_map()
    Get the type map.
init_variables(graph: Graph, graph_def: GraphDef, model_type: str = 'original_model', suffix: str
               = '') → None
    Init the embedding net variables with the given frozen model.
Parameters
    graph
        [tf.Graph] The input frozen model graph
    graph_def
        [tf.GraphDef] The input frozen model graph_def
    model_type
        [str] the type of the model
    suffix
        [str] suffix to name scope
class deepmd.model.tensor.WFCModel(*args, **kwargs)
    Bases: TensorModel

```

## Methods

<code>build(coord_, atype_, natoms, box, mesh, ...)</code>	Build the model.
<code>build_descrpt(coord_, atype_, natoms, box, ...)</code>	Build the descriptor part of the model.
<code>build_type_embedding(ntypes[, frz_model, ...])</code>	Build the type embedding part of the model.
<code>change_energy_bias(data, frozen_model, ...)</code>	Change the energy bias according to the input data and the pretrained model.
<code>data_stat(data)</code>	Data statistics.
<code>enable_compression([suffix])</code>	Enable compression.
<code>enable_mixed_precision(mixed_prec)</code>	Enable mixed precision for the model.
<code>get_class_by_input(input)</code>	Get the class by input data.
<code>get_feed_dict(coord_, atype_, natoms, box, ...)</code>	Generate the feed_dict for current descriptor.
<code>get_fitting()</code>	Get the fitting(s).
<code>get_loss(loss, lr)</code>	Get the loss function(s).
<code>get_ntypes()</code>	Get the number of types.
<code>get_num_aparam()</code>	Get the number of atomic parameters.
<code>get_num_dos()</code>	Get the number of gridpoints in energy space.
<code>get_num_fparam()</code>	Get the number of frame parameters.
<code>get_rcut()</code>	Get cutoff radius of the model.
<code>get_type_map()</code>	Get the type map.
<code>init_variables(graph, graph_def[, ...])</code>	Init the embedding net variables with the given frozen model.
<code>update_sel(global_jdata, local_jdata)</code>	Update the selection and perform neighbor statistics.

<code>get_out_size</code>	
<code>get_sel_type</code>	

## deepmd.nvnmd package

### Subpackages

#### deepmd.nvnmd.data package

`nvnmd.data` =====.

Provides

1. hardware configuration
2. default input script
3. title and citation

## Data

**jdata\_sys**  
 action configuration

**jdata\_config**  
 hardware configuration

**dscp**  
 descriptor configuration

**fitn**  
 fitting network configuration

**size**  
 ram capacity

**ctrl**  
 control flag, such as Time Division Multiplexing (TDM)

**nbit**  
 number of bits of fixed-point number

**jdata\_config\_16 (disable)**  
 difference with configure fitting size as 16

**jdata\_config\_32 (disable)**  
 difference with configure fitting size as 32

**jdata\_config\_64 (disable)**  
 difference with configure fitting size as 64

**jdata\_config\_128 (default)**  
 difference with configure fitting size as 128

**jdata\_configs**  
 all configure of jdata\_config{nfit\_node}

**jdata\_deepmd\_input**  
 default input script for nvnmd training

**NVNMD\_WELCOME**  
 nvnmd title when logging

**NVNMD\_CITATION**  
 citation of nvnmd

## Submodules

**deepmd.nvnmd.data.data module**

**deepmd.nvnmd.descriptor package**

nvnmd.se\_a =====.

Provides

1. building descriptor with continuous embedding network
2. building descriptor with quantized embedding network

## Submodules

### deepmd.nvnmd.descriptor.se\_a module

`deepmd.nvnmd.descriptor.se_a.build_davg_dstd()`

Get the davg and dstd from the dictionary `nvnmd_cfg`. The davg and dstd have been obtained by training CNN.

`deepmd.nvnmd.descriptor.se_a.build_op_descriptor()`

Replace `se_a.py/DescrptSeA/build`.

`deepmd.nvnmd.descriptor.se_a.check_switch_range(davg, dstd)`

Check the range of switch, let it in range  $[-2, 14]$ .

`deepmd.nvnmd.descriptor.se_a.descrpt2r4(inputs, natoms)`

Replace  $r_{ji} \rightarrow r'_{ji}$  where  $r_{ji} = (x_{ji}, y_{ji}, z_{ji})$  and  $r'_{ji} = (s_{ji}, \frac{s_{ji}x_{ji}}{r_{ji}}, \frac{s_{ji}y_{ji}}{r_{ji}}, \frac{s_{ji}z_{ji}}{r_{ji}})$ .

`deepmd.nvnmd.descriptor.se_a.filter_GR2D(xyz_scatter_1)`

Replace `se_a.py/_filter`.

`deepmd.nvnmd.descriptor.se_a.filter_lower_R42GR(type_i, type_input, inputs_i, is_exclude, activation_fn, bavg, stddev, trainable, suffix, seed, seed_shift, uniform_seed, filter_neuron, filter_precision, filter_resnet_dt, embedding_net_variables)`

Replace `se_a.py/DescrptSeA/_filter_lower`.

### deepmd.nvnmd.descriptor.se\_atten module

`deepmd.nvnmd.descriptor.se_atten.build_davg_dstd()`

Get the davg and dstd from the dictionary `nvnmd_cfg`. The davg and dstd have been obtained by training CNN.

`deepmd.nvnmd.descriptor.se_atten.build_op_descriptor()`

Replace `se_a.py/DescrptSeA/build`.

`deepmd.nvnmd.descriptor.se_atten.check_switch_range(davg, dstd)`

Check the range of switch, let it in range  $[-2, 14]$ .

`deepmd.nvnmd.descriptor.se_atten.descrpt2r4(inputs, atype)`

Replace  $r_{ji} \rightarrow r'_{ji}$  where  $r_{ji} = (x_{ji}, y_{ji}, z_{ji})$  and  $r'_{ji} = (s_{ji}, \frac{s_{ji}x_{ji}}{r_{ji}}, \frac{s_{ji}y_{ji}}{r_{ji}}, \frac{s_{ji}z_{ji}}{r_{ji}})$ .

`deepmd.nvnmd.descriptor.se_atten.filter_GR2D(xyz_scatter_1)`

Replace `se_a.py/_filter`.

`deepmd.nvnmd.descriptor.se_atten.filter_lower_R42GR(inputs_i, atype, nei_type_vec)`

Replace `se_a.py/DescrptSeA/_filter_lower`.

**deepmd.nvnmd.entrpoints package**

`class deepmd.nvnmd.entrpoints.MapTable(config_file: str, weight_file: str, map_file: str)`

Bases: `object`

Generate the mapping table describing the relationship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$$r_{ji}^2 \rightarrow s_{ji}$$

$$r_{ji}^2 \rightarrow h_{ji}$$

$$r_{ji}^2 \rightarrow \mathcal{G}_{ji}$$

where  $s_{ji}$  is cut-off function,  $h_{ji} = \frac{s(r_{ji})}{r_{ji}}$ , and  $\mathcal{G}_{ji}$  is embedding matrix.

The mapping function can be define as:

$$y = f(x) = y_k + (x - x_k) * dy_k$$

$$y_k = f(x_k)$$

$$dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}$$

$$x_k \leq x < x_{k+1}$$

$$x_k = k * dx$$

where  $dx$  is interpolation interval.

Parameters

`config_file`

input file name an .npz file containing the configuration information of NVNMD model

`weight_file`

input file name an .npz file containing the weights of NVNMD model

`map_file`

output file name an .npz file containing the mapping tables of NVNMD model

**References**

DOI: 10.1038/s41524-022-00773-z

## Methods

<i>build_grad</i> (x, y, Nr, Nc)	: Build gradient of tensor y of x.
<i>build_map_coef</i> (cfgs, x, ys, grads, ...)	Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.
<i>build_s2g</i> (s)	Build s->G s is switch function G is embedding net output.
<i>build_s2g_grad</i> ()	Build gradient of G with respect to s.
<i>build_t2g</i> ()	Build t->G t is chemical species of center atom and neighbor atom G is embedding net output of type.
<i>build_u2s</i> (r2)	Build tensor s, s=s(r2).
<i>build_u2s_grad</i> ()	Build gradient of s with respect to u (r^2).
<i>cal_coef4</i> (cfgs, x, y, dy)	Build mapping table coefficient for one line coef4: $a x^3 + b x^2 + c x + d = y$ : $d = y_0$   $c = y'_0$   $b = (3 y_1 - dx dy' - 2dx y'_0 - 3y_0) / dx^2$   $a = (dx y'_1 - 2 y_1 + dx y'_0 + 2 y_0) / dx^3$ .
<i>mapping</i> (x, dic_map, cfgs)	Evaluate value by mapping table operation of tensorflow.
<i>mapping2</i> (x, dic_map, cfgs)	Evaluate value by mapping table of numpy.
<i>plot_lines</i> (x, dic1[, dic2])	Plot lines to see accuracy.
<i>run_s2g</i> ()	Build s-> graph and run it to get value of mapping table.
<i>run_t2g</i> ()	Build t-> graph and run it to get value of mapping table.
<i>run_u2s</i> ()	Build u->s graph and run it to get value of mapping table.

build_davg_dstd	
build_embedding_net	
build_map	

**build\_davg\_dstd()**

**build\_embedding\_net**(xx, wbs, activation\_fn=<function tanh>)

**build\_grad**(x, y, Nr, Nc)

: Build gradient of tensor y of x.

**build\_map**()

**build\_map\_coef**(cfgs, x, ys, grads, grad\_grads, Nr, Nc)

Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.

coef4:  $a x^3 + b x^2 + c x + d = y$ :  $d = y_0$  |  $c = y'_0$  |  $b = (3 y_1 - dx dy' - 2dx y'_0 - 3y_0) / dx^2$  |  $a = (dx y'_1 - 2 y_1 + dx y'_0 + 2 y_0) / dx^3$

**build\_s2g**(s)

Build s->G s is switch function G is embedding net output.

**build\_s2g\_grad**()

Build gradient of G with respect to s.

**build\_t2g()**  
Build t->G t is chemical species of center atom and neighbor atom G is embedding net output of type.

**build\_u2s(r2)**  
Build tensor s,  $s=s(r2)$ .

**build\_u2s\_grad()**  
Build gradient of s with respect to u ( $r^2$ ).

**cal\_coef4(cfgs, x, y, dy)**  
Build mapping table coefficient for one line coef4:  $a x^3 + b x^2 + c x + d = y$ ;  $d = y0 \mid c = y0' \mid b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2 \mid a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3$ .

**mapping(x, dic\_map, cfgs)**  
Evaluate value by mapping table operation of tensorflow.

**mapping2(x, dic\_map, cfgs)**  
Evaluate value by mapping table of numpy.

**plot\_lines(x, dic1, dic2=None)**  
Plot lines to see accuracy.

**run\_s2g()**  
Build s-> graph and run it to get value of mapping table.

**run\_t2g()**  
Build t-> graph and run it to get value of mapping table.

**run\_u2s()**  
Build u->s graph and run it to get value of mapping table.

**class deepmd.nvnmd.entrypoints.Wrap**(config\_file: str, weight\_file: str, map\_file: str, model\_file: str)  
Bases: `object`  
Generate the binary model file (model.pb).  
the model file can be use to run the NVNMD with lammmps the pair style need set as:

```
pair_style nvnmd model.pb
pair_coeff * *
```

#### Parameters

**config\_file**  
input file name an .npy file containing the configuration information of NVNMD model

**weight\_file**  
input file name an .npy file containing the weights of NVNMD model

**map\_file**  
input file name an .npy file containing the mapping tables of NVNMD model

**model\_file**  
output file name an .pb file containing the model using in the NVNMD

## References

DOI: 10.1038/s41524-022-00773-z

## Methods

<code>wrap_dscp()</code>	Wrap the configuration of descriptor.
<code>wrap_fitn()</code>	Wrap the weights of fitting net.
<code>wrap_head(nhs, nws)</code>	Wrap the head information.
<code>wrap_lut()</code>	Wrap the LUT.
<code>wrap_map()</code>	Wrap the mapping table of embedding network.
<code>wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)</code>	weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

wrap	
wrap_bias	

**wrap()**

**wrap\_bias**(bias, NBIT\_DATA, NBIT\_DATA\_FL)

**wrap\_dscp()**

Wrap the configuration of descriptor.

version 0: [NBIT\_IDX\_S2G-1:0] SHIFT\_IDX\_S2G

[NBIT\_NEIB\*NTYPE-1:0] SELs [NBIT\_FIXD\*M1\*NTYPE\*NTYPE-1:0] GSs [NBIT\_FLTE-1:0] NEXPO\_DIV\_NI

version 1:

[NBIT\_FLTE-1:0] NEXPO\_DIV\_NI

**wrap\_fitn()**

Wrap the weights of fitting net.

w weight b bias

**wrap\_head**(nhs, nws)

Wrap the head information.

version nhead nheight nwidth rcut cut-off radius ntype number of atomic species nnei number of neighbors atom\_ener atom bias energy

**wrap\_lut()**

Wrap the LUT.

**wrap\_map()**

Wrap the mapping table of embedding network.

**wrap\_weight**(weight, NBIT\_DISP, NBIT\_WEIGHT)

weight: weights of fittingNet NBIT\_DISP: nbits of exponent of weight max value NBIT\_WEIGHT: nbits of mantissa of weights.



`deepmd.nvnmd.entrpoints.save_weight`(sess, file\_name: str = 'nvnmd/weight.npy')

Save the dictionary of weight to a npy file.

## Submodules

### deepmd.nvnmd.entrpoints.freeze module

`deepmd.nvnmd.entrpoints.freeze.filter_tensorVariableList`(tensorVariableList) → dict

Get the name of variable for NVNMD.

```

train_attr/min_nbor_dist
descript_attr/t_avg:0
descript_attr/t_std:0
type_embed_net/matrix_{layer 1}:0
type_embed_net/bias_{layer 1}:0

version 0: | filter_type_{atom i}/matrix_{layer 1}_{atomj}:0

filter_type_{atom i}/bias_{layer 1}_{atomj}:0
layer_{layer 1}_type_{atom i}/matrix:0
layer_{layer 1}_type_{atom i}/bias:0
final_layer_type_{atom i}/matrix:0
final_layer_type_{atom i}/bias:0

version 1: | filter_type_all/matrix_{layer 1}:0

filter_type_all/bias_{layer 1}:0
filter_type_all/matrix_{layer 1}_two_side_ebd:0
filter_type_all/bias_{layer 1}_two_side_ebd:0
layer_{layer 1}/matrix:0
layer_{layer 1}/bias:0
final_layer/matrix:0
final_layer/bias:0

```

`deepmd.nvnmd.entrpoints.freeze.save_weight`(sess, file\_name: str = 'nvnmd/weight.npy')

Save the dictionary of weight to a npy file.

**deepmd.nvnmd.entrypoints.mapt module**

```
class deepmd.nvnmd.entrypoints.mapt.MapTable(config_file: str, weight_file: str, map_file: str)
```

Bases: `object`

Generate the mapping table describing the relationship of atomic distance, cutoff function, and embedding matrix.

three mapping table will be built:

$$r_{ji}^2 \rightarrow s_{ji}$$

$$r_{ji}^2 \rightarrow h_{ji}$$

$$r_{ji}^2 \rightarrow \mathcal{G}_{ji}$$

where  $s_{ji}$  is cut-off function,  $h_{ji} = \frac{s(r_{ji})}{r_{ji}}$ , and  $\mathcal{G}_{ji}$  is embedding matrix.

The mapping function can be define as:

$$y = f(x) = y_k + (x - x_k) * dy_k$$

$$y_k = f(x_k)$$

$$dy_k = \frac{f(x_{k+1}) - f(x_k)}{dx}$$

$$x_k \leq x < x_{k+1}$$

$$x_k = k * dx$$

where  $dx$  is interpolation interval.

Parameters

`config_file`

input file name an .npz file containing the configuration information of NVNMD model

`weight_file`

input file name an .npz file containing the weights of NVNMD model

`map_file`

output file name an .npz file containing the mapping tables of NVNMD model

**References**

DOI: 10.1038/s41524-022-00773-z

## Methods

<i>build_grad</i> (x, y, Nr, Nc)	: Build gradient of tensor y of x.
<i>build_map_coef</i> (cfgs, x, ys, grads, ...)	Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.
<i>build_s2g</i> (s)	Build s->G s is switch function G is embedding net output.
<i>build_s2g_grad</i> ()	Build gradient of G with respect to s.
<i>build_t2g</i> ()	Build t->G t is chemical species of center atom and neighbor atom G is embedding net output of type.
<i>build_u2s</i> (r2)	Build tensor s, s=s(r2).
<i>build_u2s_grad</i> ()	Build gradient of s with respect to u (r^2).
<i>cal_coef4</i> (cfgs, x, y, dy)	Build mapping table coefficient for one line coef4: $a x^3 + b x^2 + c x + d = y$ : $d = y_0$   $c = y_0'$   $b = (3 y_1 - dx dy' - 2dx y_0' - 3y_0) / dx^2$   $a = (dx y_1' - 2 y_1 + dx y_0' + 2 y_0) / dx^3$ .
<i>mapping</i> (x, dic_map, cfgs)	Evaluate value by mapping table operation of tensorflow.
<i>mapping2</i> (x, dic_map, cfgs)	Evaluate value by mapping table of numpy.
<i>plot_lines</i> (x, dic1[, dic2])	Plot lines to see accuracy.
<i>run_s2g</i> ()	Build s-> graph and run it to get value of mapping table.
<i>run_t2g</i> ()	Build t-> graph and run it to get value of mapping table.
<i>run_u2s</i> ()	Build u->s graph and run it to get value of mapping table.

build_davg_dstd	
build_embedding_net	
build_map	

**build\_davg\_dstd()**

**build\_embedding\_net**(xx, wbs, activation\_fn=<function tanh>)

**build\_grad**(x, y, Nr, Nc)

: Build gradient of tensor y of x.

**build\_map**()

**build\_map\_coef**(cfgs, x, ys, grads, grad\_grads, Nr, Nc)

Build mapping table coefficient cfgs: cfg list cfg = x0, x1, dx.

coef4:  $a x^3 + b x^2 + c x + d = y$ :  $d = y_0$  |  $c = y_0'$  |  $b = (3 y_1 - dx dy' - 2dx y_0' - 3y_0) / dx^2$  |  $a = (dx y_1' - 2 y_1 + dx y_0' + 2 y_0) / dx^3$

**build\_s2g**(s)

Build s->G s is switch function G is embedding net output.

**build\_s2g\_grad**()

Build gradient of G with respect to s.

**build\_t2g()**

Build  $t \rightarrow G$   $t$  is chemical species of center atom and neighbor atom  $G$  is embedding net output of type.

**build\_u2s(r2)**

Build tensor  $s$ ,  $s=s(r2)$ .

**build\_u2s\_grad()**

Build gradient of  $s$  with respect to  $u$  ( $r^2$ ).

**cal\_coef4(cfgs, x, y, dy)**

Build mapping table coefficient for one line coef4:  $a x^3 + b x^2 + c x + d = y$ :  $d = y0 \mid c = y0' \mid b = (3 y1 - dx dy' - 2dx y0' - 3y0) / dx^2$   $a = (dx y1' - 2 y1 + dx y0' + 2 y0) / dx^3$ .

**mapping(x, dic\_map, cfgs)**

Evaluate value by mapping table operation of tensorflow.

**mapping2(x, dic\_map, cfgs)**

Evaluate value by mapping table of numpy.

**plot\_lines(x, dic1, dic2=None)**

Plot lines to see accuracy.

**run\_s2g()**

Build  $s \rightarrow$  graph and run it to get value of mapping table.

**run\_t2g()**

Build  $t \rightarrow$  graph and run it to get value of mapping table.

**run\_u2s()**

Build  $u \rightarrow s$  graph and run it to get value of mapping table.

```
deepmd.nvnmd.entrypoints.mapt.mapt(*, nvnmd_config: Optional[str] = 'nvnmd/config.npy',
                                     nvnmd_weight: Optional[str] = 'nvnmd/weight.npy',
                                     nvnmd_map: Optional[str] = 'nvnmd/map.npy', **kwargs)
```

## deepmd.nvnmd.entrypoints.train module

**deepmd.nvnmd.entrypoints.train.normalized\_input(fn, PATH\_CNN, CONFIG\_CNN)**

Normalize a input script file for continuous neural network.

**deepmd.nvnmd.entrypoints.train.normalized\_input\_qnn(jdata, PATH\_QNN, CONFIG\_CNN, WEIGHT\_CNN, MAP\_CNN)**

Normalize a input script file for quantize neural network.

```
deepmd.nvnmd.entrypoints.train.train_nvnmd(*, INPUT: str, init_model: Optional[str], restart:
                                           Optional[str], step: str, skip_neighbor_stat: bool =
                                           False, **kwargs)
```

**deepmd.nvnmd.entrypoints.wrap module**

```
class deepmd.nvnmd.entrypoints.wrap.Wrap(config_file: str, weight_file: str, map_file: str, model_file: str)
```

Bases: `object`

Generate the binary model file (model.pb).

the model file can be use to run the NVNMD with lammmps the pair style need set as:

```
pair_style nvnmd model.pb
pair_coeff * *
```

**Parameters**

`config_file`  
input file name an .npy file containing the configuration information of NVNMD model

`weight_file`  
input file name an .npy file containing the weights of NVNMD model

`map_file`  
input file name an .npy file containing the mapping tables of NVNMD model

`model_file`  
output file name an .pb file containing the model using in the NVNMD

**References**

DOI: 10.1038/s41524-022-00773-z

**Methods**

<code>wrap_dscp()</code>	Wrap the configuration of descriptor.
<code>wrap_fitn()</code>	Wrap the weights of fitting net.
<code>wrap_head(nhs, nws)</code>	Wrap the head information.
<code>wrap_lut()</code>	Wrap the LUT.
<code>wrap_map()</code>	Wrap the mapping table of embedding network.
<code>wrap_weight(weight, NBIT_DISP, NBIT_WEIGHT)</code>	weight: weights of fittingNet NBIT_DISP: nbits of exponent of weight max value NBIT_WEIGHT: nbits of mantissa of weights.

wrap	
wrap_bias	

`wrap()`

`wrap_bias(bias, NBIT_DATA, NBIT_DATA_FL)`

**wrap\_dscp()**

Wrap the configuration of descriptor.

version 0: [NBIT\_IDX\_S2G-1:0] SHIFT\_IDX\_S2G

[NBIT\_NEIB\*NTYPE-1:0] SELs [NBIT\_FIXD\*M1\*NTYPE\*NTYPE-1:0] GSs [NBIT\_FLTE-1:0]  
NEXPO\_DIV\_NI

version 1:

[NBIT\_FLTE-1:0] NEXPO\_DIV\_NI

**wrap\_fitn()**

Wrap the weights of fitting net.

w weight b bias

**wrap\_head(nhs, nws)**

Wrap the head information.

version nhead nheight nwidth rcut cut-off radius ntype number of atomic species nnei number of  
neighbors atom\_ener atom bias energy

**wrap\_lut()**

Wrap the LUT.

**wrap\_map()**

Wrap the mapping table of embedding network.

**wrap\_weight(weight, NBIT\_DISP, NBIT\_WEIGHT)**

weight: weights of fittingNet NBIT\_DISP: nbits of exponent of weight max value NBIT\_WEIGHT:  
nbits of mantissa of weights.

```
deepmd.nvnmd.entrypoints.wrap.wrap(*, nvnmd_config: Optional[str] = 'nvnmd/config.npy',
                                   nvnmd_weight: Optional[str] = 'nvnmd/weight.npy',
                                   nvnmd_map: Optional[str] = 'nvnmd/map.npy', nvnmd_model:
                                   Optional[str] = 'nvnmd/model.pb', **kwargs)
```

**deepmd.nvnmd.fit package**

nvnmd.fit =====.

Provides

1. continuous fitting network
2. quantized fitting network

**Submodules****deepmd.nvnmd.fit.ener module**

```
deepmd.nvnmd.fit.ener.one_layer_nvnmd(inputs, outputs_size, activation_fn=<function tanh>,
                                       precision=tf.float64, stddev=1.0, bavg=0.0, name='linear',
                                       reuse=None, seed=None, use_timestep=False,
                                       trainable=True, useBN=False, uniform_seed=False,
                                       initial_variables=None, mixed_prec=None,
                                       final_layer=False)
```

Build one layer with continuous or quantized value. Its weight and bias can be initialed with random or constant value.

## deepmd.nvnmd.utils package

**class** deepmd.nvnmd.utils.Encode

Bases: `object`

Encoding value as hex, bin, and dec format.

### Methods

<i>bin2hex</i> (data)	Convert binary string list to hex string list.
<i>bin2hex_str</i> (sbin)	Convert binary string to hex string.
<i>byte2hex</i> (bs, nbyte)	Convert byte into hex bs: low byte in the first hex: low byte in the right.
<i>check_dec</i> (idec, nbit[, signed, name])	Check whether the data (idec) is in the range range is $[0, 2^{nbit} - 1]$ for unsigned range is $[-2^{nbit-1}, 2^{nbit-1} - 1]$ for signed.
<i>dec2bin</i> (idec[, nbit, signed, name])	Convert dec array to binary string list.
<i>extend_bin</i> (slbin, nfull)	Extend the element of list (slbin) to the length (nfull).
<i>extend_hex</i> (slhex, nfull)	Extend the element of list (slhex) to the length (nfull).
<i>extend_list</i> (slbin, nfull)	Extend the list (slbin) to the length (nfull) the attached element of list is 0.
<i>flt2bin</i> (data, nbit_expo, nbit_frac)	Convert float into binary string list.
<i>hex2bin</i> (data)	Convert hex string list to binary string list.
<i>hex2bin_str</i> (shex)	Convert hex string to binary string.
<i>merge_bin</i> (slbin, nmerge)	Merge binary string list per nmerge value.
<i>qc</i> (v[, nbit])	Quantize value using ceil.
<i>qf</i> (v[, nbit])	Quantize value using floor.
<i>qr</i> (v[, nbit])	Quantize value using round.
<i>reverse_bin</i> (slbin, nreverse)	Reverse binary string list per nreverse value.
<i>split_bin</i> (sbin, nbit)	Split sbin into many segment with the length nbit.

find_max_expo	
flt2bin_one	
norm_expo	
split_expo_mant	

**bin2hex**(data)

Convert binary string list to hex string list.

**bin2hex\_str**(sbin)

Convert binary string to hex string.

**byte2hex**(bs, nbyte)

Convert byte into hex bs: low byte in the first hex: low byte in the right.

**check\_dec**(idec, nbit, signed=False, name='')

Check whether the data (idec) is in the range range is  $[0, 2^{nbit} - 1]$  for unsigned range is  $[-2^{nbit-1}, 2^{nbit-1} - 1]$  for signed.

**dec2bin**(idec, nbit=10, signed=False, name='')

Convert dec array to binary string list.

**extend\_bin**(slbin, nfull)

Extend the element of list (slbin) to the length (nfull).

such as, when

```
slbin = ['10010', '10100'],  
nfull = 6
```

extent to

```
['010010', '010100']
```

**extend\_hex**(slhex, nfull)

Extend the element of list (slhex) to the length (nfull).

**extend\_list**(slbin, nfull)

Extend the list (slbin) to the length (nfull) the attached element of list is 0.

such as, when

```
slbin = ['10010', '10100'],  
nfull = 4
```

extent it to

```
['10010', '10100', '00000', '00000']
```

**find\_max\_expo**(v, expo\_min=-1000)

**flt2bin**(data, nbit\_expo, nbit\_frac)

Convert float into binary string list.

**flt2bin\_one**(v, nbit\_expo, nbit\_frac)

**hex2bin**(data)

Convert hex string list to binary string list.

**hex2bin\_str**(shex)

Convert hex string to binary string.

**merge\_bin**(slbin, nmerge)

Merge binary string list per nmerge value.

**norm\_expo**(v, nbit\_frac=20, expo\_min=-1000)



```

qc(v, nbit: int = 14)
    Quantize value using ceil.
qf(v, nbit: int = 14)
    Quantize value using floor.
qr(v, nbit: int = 14)
    Quantize value using round.
reverse_bin(slbin, nreverse)
    Reverse binary string list per nreverse value.
split_bin(sbin, nbit: int)
    Split sbin into many segment with the length nbit.
split_expo_mant(v, min=-1000)

```

```

class deepmd.nvnmd.utils.FioBin
    Bases: object
    Input and output for binary file.

```

### Methods

<i><b>load</b></i> ([file_name, default_value])	Load binary file into bytes value.
<i><b>save</b></i> (file_name, data)	Save hex string into binary file.

```

load(file_name="", default_value="")
    Load binary file into bytes value.
save(file_name: str, data: List[str])
    Save hex string into binary file.

```

```

class deepmd.nvnmd.utils.FioDic
    Bases: object
    Input and output for dict class data the file can be .json or .npz file containing a dictionary.

```

### Methods

<i><b>update</b></i> (jdata, jdata_o)	Update key-value pair is key in jdata_o.keys().
---------------------------------------	---

<b>get</b>	
<b>load</b>	
<b>save</b>	

```

get(jdata, key, default_value)
load(file_name="", default_value={})
save(file_name="", dic={})

```

**update**(jdata, jdata\_o)

Update key-value pair is key in jdata\_o.keys().

Parameters

jdata  
new jdata  
jdata\_o  
origin jdata

**class** deepmd.nvnmd.utils.FioTxt

Bases: `object`

Input and output for .txt file with string.

## Methods

<code>load</code> (file_name, default_value)	Load .txt file into string list.
<code>save</code> (file_name, data)	Save string list into .txt file.

**load**(file\_name="", default\_value=[])

Load .txt file into string list.

**save**(file\_name: str = "", data: list = [])

Save string list into .txt file.

deepmd.nvnmd.utils.**get\_filter\_weight**(weights: int, spe\_j: int, layer\_l: int)

Get weight and bias of embedding network.

Parameters

weights  
[dict] weights  
spe\_j  
[int] special order of neighbor atom j 0~ntype-1  
layer\_l  
layer order in embedding network 1~nlayer

deepmd.nvnmd.utils.**get\_fitnet\_weight**(weights: dict, spe\_i: int, layer\_l: int, nlayer: int = 10)

Get weight and bias of fitting network.

Parameters

weights  
[dict] weights  
spe\_i  
[int] special order of central atom i 0~ntype-1  
layer\_l  
[int] layer order in embedding network 0~nlayer-1  
nlayer  
[int] number of layers

```
deepmd.nvnmd.utils.map_nvnmd(x, map_y, map_dy, prec, nbit=None)
```

Mapping function implemented by numpy.

```
deepmd.nvnmd.utils.nvnmd_args()
```

```
deepmd.nvnmd.utils.one_layer(inputs, outputs_size, activation_fn=<function tanh>,
                             precision=tf.float64, stddev=1.0, bavg=0.0, name='linear', reuse=None,
                             seed=None, use_timestep=False, trainable=True, useBN=False,
                             uniform_seed=False, initial_variables=None, mixed_prec=None,
                             final_layer=False)
```

Build one layer with continuous or quantized value. Its weight and bias can be initialed with random or constant value.

## Submodules

### deepmd.nvnmd.utils.argcheck module

Alias for backward compatibility.

```
deepmd.nvnmd.utils.argcheck.nvnmd_args()
```

### deepmd.nvnmd.utils.config module

```
class deepmd.nvnmd.utils.config.NvnmdConfig(jdata: dict)
```

Bases: `object`

Configuration for NVNMD record the message of model such as size, using nvnmd or not.

Parameters

jdata

a dictionary of input script

## References

DOI: 10.1038/s41524-022-00773-z

## Methods

<i>disp_message()</i>	Display the log of NVNMD.
<i>get_deepmd_jdata()</i>	Generate input script with member element one by one.
<i>get_dp_init_weights()</i>	Build the weight dict for initialization of net.
<i>get_dscp_jdata()</i>	Generate model/descriptor in input script.
<i>get_fitn_jdata()</i>	Generate model/fitting_net in input script.
<i>get_learning_rate_jdata()</i>	Generate learning_rate in input script.
<i>get_loss_jdata()</i>	Generate loss in input script.
<i>get_model_jdata()</i>	Generate model in input script.
<i>get_nvnmd_jdata()</i>	Generate nvnmd in input script.
<i>get_s_range(davg, dstd)</i>	Get the range of switch function.
<i>get_training_jdata()</i>	Generate training in input script.
<i>init_config_by_version(version, max_nnei)</i>	Initialize version-dependent parameters.
<i>init_ctrl(jdata[, jdata_parent])</i>	Initialize members about control signal.
<i>init_dpjn(jdata[, jdata_parent])</i>	Initialize members about other deepmd input.
<i>init_dscp(jdata[, jdata_parent])</i>	Initialize members about descriptor.
<i>init_fitn(jdata[, jdata_parent])</i>	Initialize members about fitting network.
<i>init_from_config(jdata)</i>	Initialize member element one by one.
<i>init_from_deepmd_input(jdata)</i>	Initialize members with input script of deepmd.
<i>init_from_jdata([jdata])</i>	Initialize this class with jdata loaded from input script.
<i>init_nbit(jdata[, jdata_parent])</i>	Initialize members about quantification precision.
<i>init_net_size()</i>	Initialize net_size.
<i>init_size(jdata[, jdata_parent])</i>	Initialize members about ram capacity.
<i>init_train_mode([mod])</i>	Configure for taining cnn or qnn.
<i>init_value()</i>	Initialize member with dict.
<i>save([file_name])</i>	Save all configuration to file.
<i>set_ntype(ntype)</i>	Set the number of type.
<i>update_config()</i>	Update config from dict.

**disp\_message()**

Display the log of NVNMD.

**get\_deepmd\_jdata()**

Generate input script with member element one by one.

**get\_dp\_init\_weights()**

Build the weight dict for initialization of net.

**get\_dscp\_jdata()**

Generate model/descriptor in input script.

**get\_fitn\_jdata()**

Generate model/fitting\_net in input script.

**get\_learning\_rate\_jdata()**

Generate learning\_rate in input script.

**get\_loss\_jdata()**

Generate loss in input script.

```

get_model_jdata()
    Generate model in input script.

get_nvnmmd_jdata()
    Generate nvnmmd in input script.

get_s_range(davg, dstd)
    Get the range of switch function.

get_training_jdata()
    Generate training in input script.

init_config_by_version(version, max_nnei)
    Initialize version-dependent parameters.

init_ctrl(jdata: dict, jdata_parent: dict = {}) → dict
    Initialize members about control signal.

init_dpin(jdata: dict, jdata_parent: dict = {}) → dict
    Initialize members about other deepmd input.

init_dscp(jdata: dict, jdata_parent: dict = {}) → dict
    Initialize members about descriptor.

init_fitn(jdata: dict, jdata_parent: dict = {}) → dict
    Initialize members about fitting network.

init_from_config(jdata)
    Initialize member element one by one.

init_from_deepmd_input(jdata)
    Initialize members with input script of deepmd.

init_from_jdata(jdata: dict = {})
    Initialize this class with jdata loaded from input script.

init_nbit(jdata: dict, jdata_parent: dict = {}) → dict
    Initialize members about quantification precision.

init_net_size()
    Initialize net_size.

init_size(jdata: dict, jdata_parent: dict = {}) → dict
    Initialize members about ram capacity.

init_train_mode(mod='cnn')
    Configure for taining cnn or qnn.

init_value()
    Initialize member with dict.

save(file_name=None)
    Save all configuration to file.

set_ntype(ntype)
    Set the number of type.

update_config()
    Update config from dict.

```

## deepmd.nvnmd.utils.encode module

```
class deepmd.nvnmd.utils.encode.Encode
```

Bases: `object`

Encoding value as hex, bin, and dec format.

## Methods

<i>bin2hex</i> (data)	Convert binary string list to hex string list.
<i>bin2hex_str</i> (sbin)	Convert binary string to hex string.
<i>byte2hex</i> (bs, nbyte)	Convert byte into hex bs: low byte in the first hex: low byte in the right.
<i>check_dec</i> (idec, nbit[, signed, name])	Check whether the data (idec) is in the range range is $[0, 2^{nbit} - 1]$ for unsigned range is $[-2^{nbit-1}, 2^{nbit-1} - 1]$ for signed.
<i>dec2bin</i> (idec[, nbit, signed, name])	Convert dec array to binary string list.
<i>extend_bin</i> (slbin, nfull)	Extend the element of list (slbin) to the length (nfull).
<i>extend_hex</i> (slhex, nfull)	Extend the element of list (slhex) to the length (nfull).
<i>extend_list</i> (slbin, nfull)	Extend the list (slbin) to the length (nfull) the attached element of list is 0.
<i>flt2bin</i> (data, nbit_expo, nbit_frac)	Convert float into binary string list.
<i>hex2bin</i> (data)	Convert hex string list to binary string list.
<i>hex2bin_str</i> (shex)	Convert hex string to binary string.
<i>merge_bin</i> (slbin, nmerge)	Merge binary string list per nmerge value.
<i>qc</i> (v[, nbit])	Quantize value using ceil.
<i>qf</i> (v[, nbit])	Quantize value using floor.
<i>qr</i> (v[, nbit])	Quantize value using round.
<i>reverse_bin</i> (slbin, nreverse)	Reverse binary string list per nreverse value.
<i>split_bin</i> (sbin, nbit)	Split sbin into many segment with the length nbit.

find_max_expo	
flt2bin_one	
norm_expo	
split_expo_mant	

```
bin2hex(data)
```

Convert binary string list to hex string list.

```
bin2hex_str(sbin)
```

Convert binary string to hex string.

```
byte2hex(bs, nbyte)
```

Convert byte into hex bs: low byte in the first hex: low byte in the right.

```
check_dec(idec, nbit, signed=False, name='')
```

Check whether the data (idec) is in the range range is  $[0, 2^{nbit} - 1]$  for unsigned range is  $[-2^{nbit-1}, 2^{nbit-1} - 1]$  for signed.

```

dec2bin(idec, nbit=10, signed=False, name='')
    Convert dec array to binary string list.

extend_bin(slbin, nfull)
    Extend the element of list (slbin) to the length (nfull).
    such as, when

    slbin = ['10010', '10100'],
    nfull = 6

    extent to
    ['010010', '010100']

extend_hex(slhex, nfull)
    Extend the element of list (slhex) to the length (nfull).

extend_list(slbin, nfull)
    Extend the list (slbin) to the length (nfull) the attached element of list is 0.
    such as, when

    slbin = ['10010', '10100'],
    nfull = 4

    extent it to
    ['10010', '10100', '00000', '00000']

find_max_expo(v, expo_min=-1000)

flt2bin(data, nbit_expo, nbit_frac)
    Convert float into binary string list.

flt2bin_one(v, nbit_expo, nbit_frac)

hex2bin(data)
    Convert hex string list to binary string list.

hex2bin_str(shex)
    Convert hex string to binary string.

merge_bin(slbin, nmerge)
    Merge binary string list per nmerge value.

norm_expo(v, nbit_frac=20, expo_min=-1000)

qc(v, nbit: int = 14)
    Quantize value using ceil.

qf(v, nbit: int = 14)
    Quantize value using floor.

```

```
qr(v, nbit: int = 14)
    Quantize value using round.

reverse_bin(sbin, nreverse)
    Reverse binary string list per nreverse value.

split_bin(sbin, nbit: int)
    Split sbins into many segments with the length nbit.

split_expo_mant(v, min=-1000)
```

### deepmd.nvnmd.utils.fio module

```
class deepmd.nvnmd.utils.fio.Fio
    Bases: object
    Basic class for FIO.
```

#### Methods

create_file_path	
exits	
get_file_list	
is_file	
is_path	
mkdir	

```
create_file_path(file_name='')

exits(file_name='')

get_file_list(path) → list

is_file(file_name)

is_path(path)

mkdir(path_name='')

class deepmd.nvnmd.utils.fio.FioBin
    Bases: object
    Input and output for binary file.
```



## Methods

<code>load([file_name, default_value])</code>	Load binary file into bytes value.
<code>save(file_name, data)</code>	Save hex string into binary file.

`load(file_name="", default_value="")`  
Load binary file into bytes value.

`save(file_name: str, data: List[str])`  
Save hex string into binary file.

`class deepmd.nvnmd.utils.fio.FioDic`

Bases: `object`

Input and output for dict class data the file can be .json or .npy file containing a dictionary.

## Methods

<code>update(jdata, jdata_o)</code>	Update key-value pair is key in jdata_o.keys().
-------------------------------------	---

get	
load	
save	

`get(jdata, key, default_value)`

`load(file_name="", default_value={})`

`save(file_name="", dic={})`

`update(jdata, jdata_o)`

Update key-value pair is key in jdata\_o.keys().

Parameters

jdata  
new jdata  
jdata\_o  
origin jdata

`class deepmd.nvnmd.utils.fio.FioJsonDic`

Bases: `object`

Input and output for .json file containing dictionary.

## Methods

<code>load([file_name, default_value])</code>	Load .json file into dict.
<code>save([file_name, dic])</code>	Save dict into .json file.

```
load(file_name="", default_value={})
```

Load .json file into dict.

```
save(file_name="", dic={})
```

Save dict into .json file.

```
class deepmd.nvnmd.utils.fio.FioNpyDic
```

Bases: `object`

Input and output for .npz file containing dictionary.

## Methods

load	
save	

```
load(file_name="", default_value={})
```

```
save(file_name="", dic={})
```

```
class deepmd.nvnmd.utils.fio.FioTxt
```

Bases: `object`

Input and output for .txt file with string.

## Methods

<code>load([file_name, default_value])</code>	Load .txt file into string list.
<code>save([file_name, data])</code>	Save string list into .txt file.

```
load(file_name="", default_value=[])
```

Load .txt file into string list.

```
save(file_name: str = "", data: list = [])
```

Save string list into .txt file.

## deepmd.nvnmd.utils.network module

```
deepmd.nvnmd.utils.network.get_sess()
```

```
deepmd.nvnmd.utils.network.matmul2_qq(a, b, nbit)
```

Quantized matmul operation for 2d tensor. a and b is input tensor, nbit represent quantification precision.

`deepmd.nvnmd.utils.network.matmul3_qq(a, b, nbit)`

Quantized matmul operation for 3d tensor. a and b is input tensor, nbit represent quantification precision.

`deepmd.nvnmd.utils.network.one_layer(inputs, outputs_size, activation_fn=<function tanh>, precision=tf.float64, stddev=1.0, bavg=0.0, name='linear', reuse=None, seed=None, use_timestep=False, trainable=True, useBN=False, uniform_seed=False, initial_variables=None, mixed_prec=None, final_layer=False)`

Build one layer with continuous or quantized value. Its weight and bias can be initialed with random or constant value.

`deepmd.nvnmd.utils.network.one_layer_t(shape, outputs_size, bavg, stddev, precision, trainable, initial_variables, seed, uniform_seed, name)`

`deepmd.nvnmd.utils.network.one_layer_wb(shape, outputs_size, bavg, stddev, precision, trainable, initial_variables, seed, uniform_seed, name)`

`deepmd.nvnmd.utils.network.qf(x, nbit)`

Quantize and floor tensor x with quantification precision nbit.

`deepmd.nvnmd.utils.network.qr(x, nbit)`

Quantize and round tensor x with quantification precision nbit.

`deepmd.nvnmd.utils.network.tanh4(x)`

### deepmd.nvnmd.utils.op module

`deepmd.nvnmd.utils.op.map_nvnmd(x, map_y, map_dy, prec, nbit=None)`

Mapping function implemented by numpy.

`deepmd.nvnmd.utils.op.r2s(r, rmin, rmax)`

### deepmd.nvnmd.utils.weight module

`deepmd.nvnmd.utils.weight.get_constant_initializer(weights, name)`

Get initial value by name and create a initializer.

`deepmd.nvnmd.utils.weight.get_filter_type_weight(weights: dict, layer_l: int)`

Get weight and bias of two\_side\_type\_embedding network.

Parameters

weights

[dict] weights

layer\_l

layer order in embedding network 1~nlayer

`deepmd.nvnmd.utils.weight.get_filter_weight(weights: int, spe_j: int, layer_l: int)`

Get weight and bias of embedding network.

Parameters

weights

[dict] weights

spe\_j  
[[int](#)] special order of neighbor atom j 0~n<sub>type</sub>-1

layer\_l  
layer order in embedding network 1~n<sub>layer</sub>

`deepmd.nvnmd.utils.weight.get_fitnet_weight(weights: dict, spe_i: int, layer_l: int, nlayer: int = 10)`

Get weight and bias of fitting network.

Parameters

weights  
[[dict](#)] weights

spe\_i  
[[int](#)] special order of central atom i 0~n<sub>type</sub>-1

layer\_l  
[[int](#)] layer order in embedding network 0~n<sub>layer</sub>-1

nlayer  
[[int](#)] number of layers

`deepmd.nvnmd.utils.weight.get_normalize(weights: dict)`

Get normalize parameter (avg and std) of  $s_{ji}$ .

`deepmd.nvnmd.utils.weight.get_type_embedding_weight(weights: dict, layer_l: int)`

Get weight and bias of type\_embedding network.

Parameters

weights  
[[dict](#)] weights

layer\_l  
layer order in embedding network 1~n<sub>layer</sub>

`deepmd.nvnmd.utils.weight.get_type_weight(weights: dict, layer_l: int)`

Get weight and bias of fitting network.

Parameters

weights  
[[dict](#)] weights

layer\_l  
[[int](#)] layer order in embedding network 0~n<sub>layer</sub>-1

`deepmd.nvnmd.utils.weight.get_weight(weights, key)`

Get weight value according to key.

## deepmd.op package

This module will house cust Tf OPs after CMake installation.

```
deepmd.op.import_ops()
```

Import all custom TF ops that are present in this submodule.

### Notes

Initially this subdir is unpopulated. CMake will install all the op module python files and shared libs.

## deepmd.train package

### Submodules

#### deepmd.train.run\_options module

Module taking care of important package constants.

```
class deepmd.train.run_options.RunOptions(init_model: Optional[str] = None, init_frz_model:
                                          Optional[str] = None, finetune: Optional[str] = None,
                                          restart: Optional[str] = None, log_path: Optional[str] =
                                          None, log_level: int = 0, mpi_log: str = 'master')
```

Bases: `object`

Class with info on how to run training (cluster, MPI and GPU config).

#### Attributes

`gpus`: `Optional[List[int]]`  
list of GPUs if any are present else None

`is_chief`: `bool`  
in distribured training it is true for tha main MPI process in serail it is always true

`world_size`: `int`  
total worker count

`my_rank`: `int`  
index of the MPI task

`nodename`: `str`  
name of the node

`node_list`:  
`[List[str]]` the list of nodes of the current mpirun

`my_device`: `str`  
deviice type - gpu or cpu

## Methods

---

<code>print_resource_summary()</code>	Print build and current running cluster configuration summary.
---------------------------------------	--

---

`gpus: Optional[List[int]]`  
`property is_chief`  
     Whether my rank is 0.  
`my_device: str`  
`my_rank: int`  
`odelist: List[int]`  
`nodename: str`  
`print_resource_summary()`  
     Print build and current running cluster configuration summary.  
`world_size: int`

## deepmd.train.trainer module

`class deepmd.train.trainer.DPTrainer(jdata, run_opt, is_compress=False)`  
 Bases: `object`

## Methods

---

<code>save_compressed()</code>	Save the compressed graph.
--------------------------------	----------------------------

---

build	
eval_single_list	
get_evaluation_results	
get_feed_dict	
get_global_step	
print_header	
print_on_training	
save_checkpoint	
train	
valid_on_the_fly	

`build(data=None, stop_batch=0, origin_type_map=None, suffix='')`  
`static eval_single_list(single_batch_list, loss, sess, get_feed_dict_func, prefix='')`  
`get_evaluation_results(batch_list)`  
`get_feed_dict(batch, is_training)`

```

get_global_step()

static print_header(fp, train_results, valid_results, multi_task_mode=False)

static print_on_training(fp, train_results, valid_results, cur_batch, cur_lr,
                        multi_task_mode=False, cur_lr_dict=None)

save_checkpoint(cur_batch: int)

save_compressed()
    Save the compressed graph.

train(train_data=None, valid_data=None)

valid_on_the_fly(fp, train_batches, valid_batches, print_header=False, fitting_key=None)

class deepmd.train.trainer.DatasetLoader(train_data: DeepmdDataSystem)
    Bases: object

    Generate an OP that loads the training data from the given DeepmdDataSystem.

    It can be used to load the training data in the training process, so there is no waiting time between
    training steps.

    Parameters
        train_data
            [DeepmdDataSystem] The training data.

```

### Examples

```

>>> loader = DatasetLoader(train_data)
>>> data_op = loader.build()
>>> with tf.Session() as sess:
>>>     data_list = sess.run(data_op)
>>> data_dict = loader.get_data_dict(data_list)

```

### Methods

<i>build()</i>	Build the OP that loads the training data.
<i>get_data_dict</i> (batch_list)	Generate a dict of the loaded data.

**build()** → List[[Tensor](#)]

Build the OP that loads the training data.

Returns

List[[tf.Tensor](#)]

Tensor of the loaded data.

**get\_data\_dict**(batch\_list: List[[ndarray](#)]) → Dict[str, [ndarray](#)]

Generate a dict of the loaded data.

Parameters

batch\_list

[List[[np.ndarray](#)]] The loaded data.

Returns

`Dict[str, np.ndarray]`  
The dict of the loaded data.

### deepmd.utils package

```
class deepmd.utils.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True,
                              type_map: Optional[List[str]] = None, optional_type_map: bool =
                              True, modifier=None, trn_all_set: bool = False, sort_atoms: bool =
                              True)
```

Bases: `object`

Class for a data system.

It loads data from hard disk, and maintains the data as a `data_dict`

Parameters

`sys_path`  
Path to the data system

`set_prefix`  
Prefix for the directories of different sets

`shuffle_test`  
If the test data are shuffled

`type_map`  
Gives the name of different atom types

`optional_type_map`  
If the `type_map.raw` in each system is optional

`modifier`  
Data modifier that has the method `modify_data`

`trn_all_set`  
Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

`sort_atoms`  
[bool] Sort atoms by atom types. Required to enable when the data is directly feeded to descriptors except mixed types.



## Methods

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded.
<code>avg(key)</code>	Return the average value of an item.
<code>check_batch_size(batch_size)</code>	Check if the system can get a batch of data with <code>batch_size</code> frames.
<code>check_test_size(test_size)</code>	Check if the system can get a test dataset with <code>test_size</code> frames.
<code>get_atom_type()</code>	Get atom types.
<code>get_batch(batch_size)</code>	Get a batch of data with <code>batch_size</code> frames.
<code>get_data_dict()</code>	Get the <code>data_dict</code> .
<code>get_natoms()</code>	Get number of atoms.
<code>get_natoms_vec(ntypes)</code>	Get number of atoms and number of atoms in different types.
<code>get_ntypes()</code>	Number of atom types in the system.
<code>get_num_batch(batch_size, set_idx)</code>	Get the number of batches in a set.
<code>get_num_set()</code>	Get number of training sets.
<code>get_sys_num_batch(batch_size)</code>	Get the number of batches in the data system.
<code>get_test([ntests])</code>	Get the test data with <code>ntests</code> frames.
<code>get_type_map()</code>	Get the type map.
<code>reduce(key_out, key_in)</code>	Generate a new item from the reduction of another atom.

reset_get_batch	
-----------------	--

**add**(key: `str`, ndof: `int`, atomic: `bool` = False, must: `bool` = False, high\_prec: `bool` = False, type\_sel: `Optional[List[int]]` = None, repeat: `int` = 1, default: `float` = 0.0, dtype: `Optional[dtype]` = None)  
Add a data item that to be loaded.

### Parameters

**key**  
The key of the item. The corresponding data is stored in `sys_path/set.*/key.npy`

**ndof**  
The number of dof

**atomic**  
The item is an atomic property. If False, the size of the data should be `nframes x ndof` If True, the size of data should be `nframes x natoms x ndof`

**must**  
The data file `sys_path/set.*/key.npy` must exist. If `must` is False and the data file does not exist, the `data_dict[find_key]` is set to 0.0

**high\_prec**  
Load the data and store in float64, otherwise in float32

**type\_sel**  
Select certain type of atoms

**repeat**  
The data will be repeated `repeat` times.

**default**  
[`float`, default=0.] default value of data

`dtype`  
[`np.dtype`, optional] the dtype of data, overwrites `high_prec` if provided

`avg(key)`  
Return the average value of an item.

`check_batch_size(batch_size)`  
Check if the system can get a batch of data with `batch_size` frames.

`check_test_size(test_size)`  
Check if the system can get a test dataset with `test_size` frames.

`get_atom_type()` → `List[int]`  
Get atom types.

`get_batch(batch_size: int) → dict`  
Get a batch of data with `batch_size` frames. The frames are randomly picked from the data system.

Parameters

`batch_size`  
size of the batch

`get_data_dict()` → `dict`  
Get the `data_dict`.

`get_natoms()`  
Get number of atoms.

`get_natoms_vec(ntypes: int)`  
Get number of atoms and number of atoms in different types.

Parameters

`ntypes`  
Number of types (may be larger than the actual number of types in the system).

Returns

`natoms`  
`natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms

`get_ntypes()` → `int`  
Number of atom types in the system.

`get_numb_batch(batch_size: int, set_idx: int) → int`  
Get the number of batches in a set.

`get_numb_set()` → `int`  
Get number of training sets.

`get_sys_numb_batch(batch_size: int) → int`  
Get the number of batches in the data system.

`get_test(ntests: int = -1) → dict`  
Get the test data with `ntests` frames.

Parameters

`ntests`  
Size of the test data set. If `ntests` is -1, all test data will be get.

`get_type_map()` → `List[str]`

Get the type map.

`reduce(key_out: str, key_in: str)`

Generate a new item from the reduction of another atom.

Parameters

`key_out`

The name of the reduced item

`key_in`

The name of the data item to be reduced

`reset_get_batch()`

```
class deepmd.utils.DeepmdDataSystem(systems: List[str], batch_size: int, test_size: int, rcut:
    Optional[float] = None, set_prefix: str = 'set', shuffle_test: bool
    = True, type_map: Optional[List[str]] = None,
    optional_type_map: bool = True, modifier=None,
    trn_all_set=False, sys_probs=None,
    auto_prob_style='prob_sys_size', sort_atoms: bool = True)
```

Bases: `object`

Class for manipulating many data systems.

It is implemented with the help of `DeepmdData`

Attributes

`default_mesh`

Mesh for each system.

## Methods

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded.
<code>add_dict(adict)</code>	Add items to the data system by a dict.
<code>get_batch([sys_idx])</code>	Get a batch of data from the data systems.
<code>get_batch_mixed()</code>	Get a batch of data from the data systems in the mixed way.
<code>get_batch_size()</code>	Get the batch size.
<code>get_batch_standard([sys_idx])</code>	Get a batch of data from the data systems in the standard way.
<code>get_nbatchs()</code>	Get the total number of batches.
<code>get_nsystems()</code>	Get the number of data systems.
<code>get_ntypes()</code>	Get the number of types.
<code>get_sys(idx)</code>	Get a certain data system.
<code>get_sys_ntest([sys_idx])</code>	Get number of tests for the currently selected system, or one defined by <code>sys_idx</code> .
<code>get_test([sys_idx, n_test])</code>	Get test data from the the data systems.
<code>get_type_map()</code>	Get the type map.
<code>reduce(key_out, key_in)</code>	Generate a new item from the reduction of another atom.

compute_energy_shift	
get_data_dict	
print_summary	
set_sys_probs	

**add**(key: str, ndof: int, atomic: bool = False, must: bool = False, high\_prec: bool = False, type\_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0)

Add a data item that to be loaded.

Parameters

key

The key of the item. The corresponding data is stored in sys\_path/set.\*/key.npy

ndof

The number of dof

atomic

The item is an atomic property. If False, the size of the data should be nframes x ndof If True, the size of data should be nframes x natoms x ndof

must

The data file sys\_path/set.\*/key.npy must exist. If must is False and the data file does not exist, the data\_dict[find\_key] is set to 0.0

high\_prec

Load the data and store in float64, otherwise in float32

type\_sel

Select certain type of atoms

repeat

The data will be repeated repeat times.

default, default=0.

Default value of data

**add\_dict**(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

```
adict[key] = {
    "ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec, "type_sel":
    type_sel, "repeat": repeat,
}
```

For the explanation of the keys see add

**compute\_energy\_shift**(rcond=None, key='energy')

**property default\_mesh:** List[ndarray]

Mesh for each system.

**get\_batch**(sys\_idx: Optional[int] = None) → dict

Get a batch of data from the data systems.

Parameters

sys\_idx

[int] The index of system from which the batch is get. If sys\_idx is not None,

`sys_probs` and `auto_prob_style` are ignored. If `sys_idx` is `None`, automatically determine the system according to `sys_probs` or `auto_prob_style`, see the following. This option does not work for mixed systems.

Returns

`dict`

The batch data

`get_batch_mixed()` → `dict`

Get a batch of data from the data systems in the mixed way.

Returns

`dict`

The batch data

`get_batch_size()` → `int`

Get the batch size.

`get_batch_standard(sys_idx: Optional[int] = None)` → `dict`

Get a batch of data from the data systems in the standard way.

Parameters

`sys_idx`

[`int`] The index of system from which the batch is get. If `sys_idx` is not `None`, `sys_probs` and `auto_prob_style` are ignored. If `sys_idx` is `None`, automatically determine the system according to `sys_probs` or `auto_prob_style`, see the following.

Returns

`dict`

The batch data

`get_data_dict(ii: int = 0)` → `dict`

`get_nbatches()` → `int`

Get the total number of batches.

`get_nsystems()` → `int`

Get the number of data systems.

`get_ntypes()` → `int`

Get the number of types.

`get_sys(idx: int)` → `DeepmdData`

Get a certain data system.

`get_sys_nctest(sys_idx=None)`

Get number of tests for the currently selected system, or one defined by `sys_idx`.

`get_test(sys_idx: Optional[int] = None, n_test: int = -1)`

Get test data from the the data systems.

Parameters

`sys_idx`

The test dat of system with index `sys_idx` will be returned. If is `None`, the currently selected system will be returned.

```

    n_test
        Number of test data. If set to -1 all test data will be get.

get_type_map() → List[str]
    Get the type map.

print_summary(name)

reduce(key_out, key_in)
    Generate a new item from the reduction of another atom.

    Parameters
        key_out
            The name of the reduced item
        key_in
            The name of the data item to be reduced

set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')

class deepmd.utils.LearningRateExp(start_lr: float, stop_lr: float = 5e-08, decay_steps: int = 5000,
                                   decay_rate: float = 0.95)

```

Bases: `object`

The exponentially decaying learning rate.

The learning rate at step  $t$  is given by

$$\alpha(t) = \alpha_0 \lambda^{t/\tau}$$

where  $\alpha$  is the learning rate,  $\alpha_0$  is the starting learning rate,  $\lambda$  is the decay rate, and  $\tau$  is the decay steps.

Parameters

start\_lr  
Starting learning rate  $\alpha_0$

stop\_lr  
Stop learning rate  $\alpha_1$

decay\_steps  
Learning rate decay every this number of steps  $\tau$

decay\_rate  
The decay rate  $\lambda$ . If stop\_step is provided in build, then it will be determined automatically and overwritten.

## Methods

<code>build(global_step[, stop_step])</code>	Build the learning rate.
<code>start_lr()</code>	Get the start lr.
<code>value(step)</code>	Get the lr at a certain step.

`build(global_step: Tensor, stop_step: Optional[int] = None) → Tensor`

Build the learning rate.

Parameters

`global_step`

The tf Tensor providing the global training step

`stop_step`

The stop step. If provided, the `decay_rate` will be determined automatically and overwritten.

Returns

`learning_rate`

The learning rate

`start_lr()` → float

Get the start lr.

`value(step: int)` → float

Get the lr at a certain step.

`class deepmd.utils.PairTab(filename: str)`

Bases: `object`

Pairwise tabulated potential.

Parameters

`filename`

File name for the short-range tabulated potential. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

## Methods

<code>get()</code>	Get the serialized table.
<code>reinit(filename)</code>	Initialize the tabulated interaction.

`get()` → `Tuple[array, array]`

Get the serialized table.

`reinit(filename: str)` → `None`

Initialize the tabulated interaction.

Parameters

`filename`

File name for the short-range tabulated potential. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

`class deepmd.utils.Plugin`

Bases: `object`

A class to register and restore plugins.

## Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins['xx'])
```

### Attributes

plugins  
[Dict[str, object]] plugins

## Methods

<code>get_plugin(key)</code>	Visit a plugin by key.
<code>register(key)</code>	Register a plugin.

`get_plugin(key) → object`

Visit a plugin by key.

### Parameters

key  
[str] key of the plugin

### Returns

object  
the plugin

`register(key: str) → Callable[[object], object]`

Register a plugin.

### Parameters

key  
[str] key of the plugin

### Returns

Callable[[object], object]  
decorator

`class deepmd.utils.PluginVariant(*args, **kwargs)`

Bases: object

A class to remove type from input arguments.



## Submodules

### deepmd.utils.argcheck module

Alias for backward compatibility.

`deepmd.utils.argcheck.gen_args(**kwargs) → List[Argument]`

`deepmd.utils.argcheck.gen_doc(*, make_anchor=True, make_link=True, **kwargs)`

`deepmd.utils.argcheck.gen_json(**kwargs)`

`deepmd.utils.argcheck.list_to_doc(xx)`

`deepmd.utils.argcheck.normalize(data)`

`deepmd.utils.argcheck.type_embedding_args()`

### deepmd.utils.batch\_size module

`class deepmd.utils.batch_size.AutoBatchSize(initial_batch_size: int = 1024, factor: float = 2.0)`

Bases: *AutoBatchSize*

#### Methods

<code>execute(callable, start_index, natoms)</code>	Excuate a method with given batch size.
<code>execute_all(callable, total_size, natoms, ...)</code>	Excuate a method with all given data.
<code>is_gpu_available()</code>	Check if GPU is available.
<code>is_oom_error(e)</code>	Check if the exception is an OOM error.

`is_gpu_available() → bool`

Check if GPU is available.

Returns

`bool`

True if GPU is available

`is_oom_error(e: Exception) → bool`

Check if the exception is an OOM error.

Parameters

`e`

[`Exception`] Exception

**deepmd.utils.compat module**

Alias for backward compatibility.

```
deepmd.utils.compat.convert_input_v0_v1(jdata: Dict[str, Any], warning: bool = True, dump:
                                         Optional[Union[str, Path]] = None) → Dict[str, Any]
```

Convert input from v0 format to v1.

Parameters

jdata

[Dict[str, Any]] loaded json/yaml file

warning

[bool, optional] whether to show deprecation warning, by default True

dump

[Optional[Union[str, Path]], optional] whether to dump converted file, by default None

Returns

Dict[str, Any]

converted output

```
deepmd.utils.compat.convert_input_v1_v2(jdata: Dict[str, Any], warning: bool = True, dump:
                                         Optional[Union[str, Path]] = None) → Dict[str, Any]
```

```
deepmd.utils.compat.deprecate_numb_test(jdata: Dict[str, Any], warning: bool = True, dump:
                                         Optional[Union[str, Path]] = None) → Dict[str, Any]
```

Deprecate numb\_test since v2.1. It has taken no effect since v2.0.

See #1243.

Parameters

jdata

[Dict[str, Any]] loaded json/yaml file

warning

[bool, optional] whether to show deprecation warning, by default True

dump

[Optional[Union[str, Path]], optional] whether to dump converted file, by default None

Returns

Dict[str, Any]

converted output

```
deepmd.utils.compat.update_deepmd_input(jdata: Dict[str, Any], warning: bool = True, dump:
                                         Optional[Union[str, Path]] = None) → Dict[str, Any]
```

**deepmd.utils.compress module**

```
deepmd.utils.compress.get_extra_side_embedding_net_variable(self, graph_def, type_side_suffix,
                                                            varialbe_name, suffix)
```

```
deepmd.utils.compress.get_two_side_type_embedding(self, graph)
```

```
deepmd.utils.compress.get_type_embedding(self, graph)
```

```
deepmd.utils.compress.make_data(self, xx)
```

**deepmd.utils.convert module**

```
deepmd.utils.convert.convert_012_to_21(input_model: str, output_model: str)
```

Convert DP 0.12 graph to 2.1 graph.

Parameters

input\_model  
[str] filename of the input graph

output\_model  
[str] filename of the output graph

```
deepmd.utils.convert.convert_10_to_21(input_model: str, output_model: str)
```

Convert DP 1.0 graph to 2.1 graph.

Parameters

input\_model  
[str] filename of the input graph

output\_model  
[str] filename of the output graph

```
deepmd.utils.convert.convert_12_to_21(input_model: str, output_model: str)
```

Convert DP 1.2 graph to 2.1 graph.

Parameters

input\_model  
[str] filename of the input graph

output\_model  
[str] filename of the output graph

```
deepmd.utils.convert.convert_13_to_21(input_model: str, output_model: str)
```

Convert DP 1.3 graph to 2.1 graph.

Parameters

input\_model  
[str] filename of the input graph

output\_model  
[str] filename of the output graph

`deepmd.utils.convert.convert_20_to_21(input_model: str, output_model: str)`

Convert DP 2.0 graph to 2.1 graph.

Parameters

`input_model`  
[str] filename of the input graph

`output_model`  
[str] filename of the output graph

`deepmd.utils.convert.convert_dp012_to_dp10(file: str)`

Convert DP 0.12 graph text to 1.0 graph text.

Parameters

`file`  
[str] filename of the graph text

`deepmd.utils.convert.convert_dp10_to_dp11(file: str)`

Convert DP 1.0 graph text to 1.1 graph text.

Parameters

`file`  
[str] filename of the graph text

`deepmd.utils.convert.convert_dp12_to_dp13(file: str)`

Convert DP 1.2 graph text to 1.3 graph text.

Parameters

`file`  
[str] filename of the graph text

`deepmd.utils.convert.convert_dp13_to_dp20(fname: str)`

Convert DP 1.3 graph text to 2.0 graph text.

Parameters

`fname`  
[str] filename of the graph text

`deepmd.utils.convert.convert_dp20_to_dp21(fname: str)`

`deepmd.utils.convert.convert_pb_to_pbtxt(pbfile: str, pbtxtfile: str)`

Convert DP graph to graph text.

Parameters

`pbfile`  
[str] filename of the input graph

`pbtxtfile`  
[str] filename of the output graph text

`deepmd.utils.convert.convert_pbtxt_to_pb(pbtxtfile: str, pbfile: str)`

Convert DP graph text to graph.

Parameters

`pbtxtfile`  
[str] filename of the input graph text

pbfile  
     [str] filename of the output graph

`deepmd.utils.convert.convert_to_21(input_model: str, output_model: str, version: Optional[str] = None)`

Convert DP graph to 2.1 graph.

Parameters

input\_model  
     [str] filename of the input graph

output\_model  
     [str] filename of the output graph

version  
     [str] version of the input graph, if not specified, it will be detected automatically

`deepmd.utils.convert.detect_model_version(input_model: str)`

Detect DP graph version.

Parameters

input\_model  
     [str] filename of the input graph

## deepmd.utils.data module

Alias for backward compatibility.

`class deepmd.utils.data.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool = True, type_map: Optional[List[str]] = None, optional_type_map: bool = True, modifier=None, trn_all_set: bool = False, sort_atoms: bool = True)`

Bases: `object`

Class for a data system.

It loads data from hard disk, and mantains the data as a `data_dict`

Parameters

sys\_path  
     Path to the data system

set\_prefix  
     Prefix for the directories of different sets

shuffle\_test  
     If the test data are shuffled

type\_map  
     Gives the name of different atom types

optional\_type\_map  
     If the `type_map.raw` in each system is optional

modifier  
     Data modifier that has the method `modify_data`

`trn_all_set`  
Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

`sort_atoms`  
[bool] Sort atoms by atom types. Required to enable when the data is directly feeded to descriptors except mixed types.

## Methods

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded.
<code>avg(key)</code>	Return the average value of an item.
<code>check_batch_size(batch_size)</code>	Check if the system can get a batch of data with <code>batch_size</code> frames.
<code>check_test_size(test_size)</code>	Check if the system can get a test dataset with <code>test_size</code> frames.
<code>get_atom_type()</code>	Get atom types.
<code>get_batch(batch_size)</code>	Get a batch of data with <code>batch_size</code> frames.
<code>get_data_dict()</code>	Get the <code>data_dict</code> .
<code>get_natoms()</code>	Get number of atoms.
<code>get_natoms_vec(ntypes)</code>	Get number of atoms and number of atoms in different types.
<code>get_ntypes()</code>	Number of atom types in the system.
<code>get_num_batch(batch_size, set_idx)</code>	Get the number of batches in a set.
<code>get_num_set()</code>	Get number of training sets.
<code>get_sys_num_batch(batch_size)</code>	Get the number of batches in the data system.
<code>get_test([ntests])</code>	Get the test data with <code>ntests</code> frames.
<code>get_type_map()</code>	Get the type map.
<code>reduce(key_out, key_in)</code>	Generate a new item from the reduction of another atom.

reset_get_batch	
-----------------	--

**add**(key: str, ndof: int, atomic: bool = False, must: bool = False, high\_prec: bool = False, type\_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0, dtype: Optional[dtype] = None)

Add a data item that to be loaded.

### Parameters

**key**  
The key of the item. The corresponding data is stored in `sys_path/set.*/key.npy`

**ndof**  
The number of dof

**atomic**  
The item is an atomic property. If False, the size of the data should be `nframes x ndof` If True, the size of data should be `nframes x natoms x ndof`

**must**  
The data file `sys_path/set.*/key.npy` must exist. If `must` is False and the data file does not exist, the `data_dict[find_key]` is set to 0.0

`high_prec`  
 Load the data and store in float64, otherwise in float32

`type_sel`  
 Select certain type of atoms

`repeat`  
 The data will be repeated repeat times.

`default`  
`[float, default=0.]` default value of data

`dtype`  
`[np.dtype, optional]` the dtype of data, overwrites `high_prec` if provided

**avg(key)**  
 Return the average value of an item.

**check\_batch\_size(batch\_size)**  
 Check if the system can get a batch of data with `batch_size` frames.

**check\_test\_size(test\_size)**  
 Check if the system can get a test dataset with `test_size` frames.

**get\_atom\_type()** → `List[int]`  
 Get atom types.

**get\_batch(batch\_size: int) → dict**  
 Get a batch of data with `batch_size` frames. The frames are randomly picked from the data system.

Parameters  
`batch_size`  
 size of the batch

**get\_data\_dict()** → `dict`  
 Get the `data_dict`.

**get\_natoms()**  
 Get number of atoms.

**get\_natoms\_vec(ntypes: int)**  
 Get number of atoms and number of atoms in different types.

Parameters  
`ntypes`  
 Number of types (may be larger than the actual number of types in the system).

Returns  
**natoms**  
`natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms

**get\_ntypes()** → `int`  
 Number of atom types in the system.

**get\_num\_batch(batch\_size: int, set\_idx: int) → int**  
 Get the number of batches in a set.

`get_numb_set()` → `int`

Get number of training sets.

`get_sys_numb_batch(batch_size: int)` → `int`

Get the number of batches in the data system.

`get_test(ntests: int = -1)` → `dict`

Get the test data with ntests frames.

Parameters

`ntests`

Size of the test data set. If ntests is -1, all test data will be get.

`get_type_map()` → `List[str]`

Get the type map.

`reduce(key_out: str, key_in: str)`

Generate a new item from the reduction of another atom.

Parameters

`key_out`

The name of the reduced item

`key_in`

The name of the data item to be reduced

`reset_get_batch()`

## deepmd.utils.data\_system module

Alias for backward compatibility.

```
class deepmd.utils.data_system.DeepmdDataSystem(systems: List[str], batch_size: int, test_size: int,
rcut: Optional[float] = None, set_prefix: str =
'set', shuffle_test: bool = True, type_map:
Optional[List[str]] = None, optional_type_map:
bool = True, modifier=None, trn_all_set=False,
sys_probs=None,
auto_prob_style='prob_sys_size', sort_atoms:
bool = True)
```

Bases: `object`

Class for manipulating many data systems.

It is implemented with the help of DeepmdData

Attributes

`default_mesh`

Mesh for each system.



## Methods

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded.
<code>add_dict(adict)</code>	Add items to the data system by a dict.
<code>get_batch([sys_idx])</code>	Get a batch of data from the data systems.
<code>get_batch_mixed()</code>	Get a batch of data from the data systems in the mixed way.
<code>get_batch_size()</code>	Get the batch size.
<code>get_batch_standard([sys_idx])</code>	Get a batch of data from the data systems in the standard way.
<code>get_nbatches()</code>	Get the total number of batches.
<code>get_nsystems()</code>	Get the number of data systems.
<code>get_ntypes()</code>	Get the number of types.
<code>get_sys(idx)</code>	Get a certain data system.
<code>get_sys_ntest([sys_idx])</code>	Get number of tests for the currently selected system, or one defined by <code>sys_idx</code> .
<code>get_test([sys_idx, n_test])</code>	Get test data from the the data systems.
<code>get_type_map()</code>	Get the type map.
<code>reduce(key_out, key_in)</code>	Generate a new item from the reduction of another atom.

<code>compute_energy_shift</code>	
<code>get_data_dict</code>	
<code>print_summary</code>	
<code>set_sys_probs</code>	

**add**(key: str, ndof: int, atomic: bool = False, must: bool = False, high\_prec: bool = False, type\_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0)

Add a data item that to be loaded.

Parameters

key

The key of the item. The corresponding data is stored in `sys_path/set.*/key.npy`

ndof

The number of dof

atomic

The item is an atomic property. If False, the size of the data should be `nframes x ndof` If True, the size of data should be `nframes x natoms x ndof`

must

The data file `sys_path/set.*/key.npy` must exist. If must is False and the data file does not exist, the `data_dict[find_key]` is set to 0.0

high\_prec

Load the data and store in float64, otherwise in float32

type\_sel

Select certain type of atoms

repeat

The data will be repeated repeat times.

default, default=0.  
Default value of data

**add\_dict**(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

```
adict[key] = {
    "ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec, "type_sel":
    type_sel, "repeat": repeat,
}
```

For the explanation of the keys see add

**compute\_energy\_shift**(rcond=None, key='energy')

**property default\_mesh:** List[ndarray]

Mesh for each system.

**get\_batch**(sys\_idx: Optional[int] = None) → dict

Get a batch of data from the data systems.

Parameters

sys\_idx  
[int] The index of system from which the batch is get. If sys\_idx is not None, sys\_probs and auto\_prob\_style are ignored If sys\_idx is None, automatically determine the system according to sys\_probs or auto\_prob\_style, see the following. This option does not work for mixed systems.

Returns

dict  
The batch data

**get\_batch\_mixed**() → dict

Get a batch of data from the data systems in the mixed way.

Returns

dict  
The batch data

**get\_batch\_size**() → int

Get the batch size.

**get\_batch\_standard**(sys\_idx: Optional[int] = None) → dict

Get a batch of data from the data systems in the standard way.

Parameters

sys\_idx  
[int] The index of system from which the batch is get. If sys\_idx is not None, sys\_probs and auto\_prob\_style are ignored If sys\_idx is None, automatically determine the system according to sys\_probs or auto\_prob\_style, see the following.

Returns

dict  
The batch data

**get\_data\_dict**(ii: int = 0) → dict

`get_nbatches()` → `int`

Get the total number of batches.

`get_nsystems()` → `int`

Get the number of data systems.

`get_ntypes()` → `int`

Get the number of types.

`get_sys(idx: int)` → `DeepmdData`

Get a certain data system.

`get_sys_ntest(sys_idx=None)`

Get number of tests for the currently selected system, or one defined by `sys_idx`.

`get_test(sys_idx: Optional[int] = None, n_test: int = -1)`

Get test data from the the data systems.

Parameters

`sys_idx`

The test dat of system with index `sys_idx` will be returned. If is None, the currently selected system will be returned.

`n_test`

Number of test data. If set to -1 all test data will be get.

`get_type_map()` → `List[str]`

Get the type map.

`print_summary(name)`

`reduce(key_out, key_in)`

Generate a new item from the reduction of another atom.

Parameters

`key_out`

The name of the reduced item

`key_in`

The name of the data item to be reduced

`set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')`

`deepmd.utils.data_system.prob_sys_size_ext(keywords, nsystems, nbatch)`

`deepmd.utils.data_system.process_sys_probs(sys_probs, nbatch)`

## deepmd.utils.errors module

`exception deepmd.utils.errors.GraphTooLargeError`

Bases: `Exception`

The graph is too large, exceeding protobuf's hard limit of 2GB.

`exception deepmd.utils.errors.GraphWithoutTensorError`

Bases: `Exception`

`exception deepmd.utils.errors.OutOfMemoryError`

Bases: `Exception`

This error is caused by out-of-memory (OOM).

### `deepmd.utils.finetune` module

`deepmd.utils.finetune.replace_model_params_with_pretrained_model(jdata: Dict[str, Any],  
pretrained_model: str)`

Replace the model params in input script according to pretrained model.

Parameters

`jdata`

[Dict[str, Any]] input script

`pretrained_model`

[str] filename of the pretrained model

### `deepmd.utils.graph` module

`deepmd.utils.graph.get_attention_layer_nodes_from_graph_def(graph_def: GraphDef, suffix: str =  
"") → Dict`

Get the attention layer nodes with the given tf.GraphDef object.

Parameters

`graph_def`

The input tf.GraphDef object

`suffix`

[str, optional] The scope suffix

Returns

`Dict`

The attention layer nodes within the given tf.GraphDef object

`deepmd.utils.graph.get_attention_layer_variables_from_graph_def(graph_def: GraphDef, suffix:  
str = "") → Dict`

Get the attention layer variables with the given tf.GraphDef object.

Parameters

`graph_def`

[tf.GraphDef] The input tf.GraphDef object

`suffix`

[str, optional] The suffix of the scope

Returns

`Dict`

The attention layer variables within the given tf.GraphDef object

`deepmd.utils.graph.get_embedding_net_nodes(model_file: str, suffix: str = "") → Dict`

Get the embedding net nodes with the given frozen model(model\_file).

Parameters

`model_file`  
The input frozen model path

`suffix`  
[`str`, `optional`] The suffix of the scope

Returns

`Dict`  
The embedding net nodes with the given frozen model

`deepmd.utils.graph.get_embedding_net_nodes_from_graph_def`(`graph_def`: `GraphDef`, `suffix`: `str` = "")  
→ `Dict`

Get the embedding net nodes with the given `tf.GraphDef` object.

Parameters

`graph_def`  
The input `tf.GraphDef` object

`suffix`  
[`str`, `optional`] The scope suffix

Returns

`Dict`  
The embedding net nodes within the given `tf.GraphDef` object

`deepmd.utils.graph.get_embedding_net_variables`(`model_file`: `str`, `suffix`: `str` = "") → `Dict`

Get the embedding net variables with the given frozen model(`model_file`).

Parameters

`model_file`  
The input frozen model path

`suffix`  
[`str`, `optional`] The suffix of the scope

Returns

`Dict`  
The embedding net variables within the given frozen model

`deepmd.utils.graph.get_embedding_net_variables_from_graph_def`(`graph_def`: `GraphDef`, `suffix`: `str` = "") → `Dict`

Get the embedding net variables with the given `tf.GraphDef` object.

Parameters

`graph_def`  
The input `tf.GraphDef` object

`suffix`  
[`str`, `optional`] The suffix of the scope

Returns

`Dict`  
The embedding net variables within the given `tf.GraphDef` object

`deepmd.utils.graph.get_extra_embedding_net_suffix`(`type_one_side`: `bool`)

Get the extra embedding net suffix according to the value of `type_one_side`.

Parameters

type\_one\_side  
The value of type\_one\_side

Returns

str  
The extra embedding net suffix

```
deepmd.utils.graph.get_extra_embedding_net_variables_from_graph_def(graph_def: GraphDef,  
                                                                    suffix: str, extra_suffix:  
                                                                    str, layer_size: int)
```

Get extra embedding net variables from the given tf.GraphDef object. The “extra embedding net” means the embedding net with only type embeddings input, which occurs in “se\_atten\_v2” and “se\_a\_ebd\_v2” descriptor.

Parameters

graph\_def  
The input tf.GraphDef object

suffix  
[str] The “common” suffix in the descriptor

extra\_suffix  
[str] This value depends on the value of “type\_one\_side”. It should always be “\_one\_side\_ebd” or “\_two\_side\_ebd”

layer\_size  
[int] The layer size of the embedding net

Returns

Dict  
The extra embedding net variables within the given tf.GraphDef object

```
deepmd.utils.graph.get_fitting_net_nodes(model_file: str) → Dict
```

Get the fitting net nodes with the given frozen model(model\_file).

Parameters

model\_file  
The input frozen model path

Returns

Dict  
The fitting net nodes with the given frozen model

```
deepmd.utils.graph.get_fitting_net_nodes_from_graph_def(graph_def: GraphDef, suffix: str = "")  
                                                         → Dict
```

Get the fitting net nodes with the given tf.GraphDef object.

Parameters

graph\_def  
The input tf.GraphDef object

suffix  
suffix of the scope

Returns

**Dict**

The fitting net nodes within the given tf.GraphDef object

`deepmd.utils.graph.get_fitting_net_variables(model_file: str, suffix: str = "") → Dict`

Get the fitting net variables with the given frozen model(model\_file).

Parameters

`model_file`

The input frozen model path

`suffix`

suffix of the scope

Returns

**Dict**

The fitting net variables within the given frozen model

`deepmd.utils.graph.get_fitting_net_variables_from_graph_def(graph_def: GraphDef, suffix: str = "") → Dict`

Get the fitting net variables with the given tf.GraphDef object.

Parameters

`graph_def`

The input tf.GraphDef object

`suffix`

suffix of the scope

Returns

**Dict**

The fitting net variables within the given tf.GraphDef object

`deepmd.utils.graph.get_pattern_nodes_from_graph_def(graph_def: GraphDef, pattern: str) → Dict`

Get the pattern nodes with the given tf.GraphDef object.

Parameters

`graph_def`

The input tf.GraphDef object

`pattern`

The node pattern within the graph\_def

Returns

**Dict**

The fitting net nodes within the given tf.GraphDef object

`deepmd.utils.graph.get_tensor_by_name(model_file: str, tensor_name: str) → Tensor`

Load tensor value from the frozen model(model\_file).

Parameters

`model_file``[str]` The input frozen model path`tensor_name``[str]` Indicates which tensor which will be loaded from the frozen model

Returns

`tf.Tensor`

The tensor which was loaded from the frozen model

Raises

**GraphWithoutTensorError**

Whether the tensor\_name is within the frozen model

`deepmd.utils.graph.get_tensor_by_name_from_graph(graph: Graph, tensor_name: str) → Tensor`

Load tensor value from the given tf.Graph object.

Parameters

graph

[`tf.Graph`] The input TensorFlow graph

tensor\_name

[`str`] Indicates which tensor which will be loaded from the frozen model

Returns

`tf.Tensor`

The tensor which was loaded from the frozen model

Raises

**GraphWithoutTensorError**

Whether the tensor\_name is within the frozen model

`deepmd.utils.graph.get_tensor_by_type(node, data_type: dtype) → Tensor`

Get the tensor value within the given node according to the input data\_type.

Parameters

node

The given tensorflow graph node

data\_type

The data type of the node

Returns

`tf.Tensor`

The tensor value of the given node

`deepmd.utils.graph.get_type_embedding_net_nodes_from_graph_def(graph_def: GraphDef, suffix: str = '') → Dict`

Get the type embedding net nodes with the given tf.GraphDef object.

Parameters

graph\_def

The input tf.GraphDef object

suffix

[`str`, optional] The scope suffix

Returns

`Dict`

The type embedding net nodes within the given tf.GraphDef object



```
deepmd.utils.graph.get_type_embedding_net_variables_from_graph_def(graph_def: GraphDef,
                                                                    suffix: str = '') → Dict
```

Get the type embedding net variables with the given tf.GraphDef object.

Parameters

`graph_def`  
[tf.GraphDef] The input tf.GraphDef object

`suffix`  
[str, optional] The suffix of the scope

Returns

**Dict**  
The embedding net variables within the given tf.GraphDef object

```
deepmd.utils.graph.get_variables_from_graph_def_as_numpy_array(graph_def: GraphDef, pattern:
                                                                str)
```

Get variables from the given tf.GraphDef object, with numpy array returns.

Parameters

`graph_def`  
The input tf.GraphDef object

`pattern`  
[str] The name of variable

Returns

**np.ndarray**  
The numpy array of the variable

```
deepmd.utils.graph.load_graph_def(model_file: str) → Tuple[Graph, GraphDef]
```

Load graph as well as the graph\_def from the frozen model(model\_file).

Parameters

`model_file`  
[str] The input frozen model path

Returns

**tf.Graph**  
The graph loaded from the frozen model

**tf.GraphDef**  
The graph\_def loaded from the frozen model

## deepmd.utils.learning\_rate module

```
class deepmd.utils.learning_rate.LearningRateExp(start_lr: float, stop_lr: float = 5e-08,
                                                  decay_steps: int = 5000, decay_rate: float =
                                                  0.95)
```

Bases: **object**

The exponentially decaying learning rate.

The learning rate at step  $t$  is given by

$$\alpha(t) = \alpha_0 \lambda^{t/\tau}$$

where  $\alpha$  is the learning rate,  $\alpha_0$  is the starting learning rate,  $\lambda$  is the decay rate, and  $\tau$  is the decay steps.

#### Parameters

- `start_lr`  
Starting learning rate  $\alpha_0$
- `stop_lr`  
Stop learning rate  $\alpha_1$
- `decay_steps`  
Learning rate decay every this number of steps  $\tau$
- `decay_rate`  
The decay rate  $\lambda$ . If `stop_step` is provided in `build`, then it will be determined automatically and overwritten.

#### Methods

<code>build(global_step[, stop_step])</code>	Build the learning rate.
<code>start_lr()</code>	Get the start lr.
<code>value(step)</code>	Get the lr at a certain step.

`build(global_step: Tensor, stop_step: Optional[int] = None) → Tensor`

Build the learning rate.

#### Parameters

- `global_step`  
The tf Tensor providing the global training step
- `stop_step`  
The stop step. If provided, the `decay_rate` will be determined automatically and overwritten.

#### Returns

- learning\_rate**  
The learning rate

`start_lr() → float`

Get the start lr.

`value(step: int) → float`

Get the lr at a certain step.

#### deepmd.utils.multi\_init module

`deepmd.utils.multi_init.replace_model_params_with_frz_multi_model(jdata: Dict[str, Any], pretrained_model: str)`

Replace the model params in input script according to pretrained frozen multi-task united model.

#### Parameters

- `jdata`  
[Dict[str, Any]] input script

pretrained\_model  
 [str] filename of the pretrained frozen multi-task united model

## deepmd.utils.neighbor\_stat module

**class** deepmd.utils.neighbor\_stat.NeighborStat(ntypes: int, rcut: float, one\_type: bool = False)

Bases: object

Class for getting training data information.

It loads data from DeepmdData object, and measures the data info, including nearest nbor distance between atoms, max nbor size of atoms and the output data range of the environment matrix.

Parameters

ntypes

The num of atom types

rcut

The cut-off radius

one\_type

[bool, optional, default=False] Treat all types as a single type.

## Methods

<i>get_stat</i> (data)	Get the data statistics of the training data, including nearest nbor distance between atoms, max nbor size of atoms.
------------------------	--

**get\_stat**(data: DeepmdDataSystem) → Tuple[float, List[int]]

Get the data statistics of the training data, including nearest nbor distance between atoms, max nbor size of atoms.

Parameters

data

Class for manipulating many data systems. It is implemented with the help of DeepmdData.

Returns

**min\_nbor\_dist**

The nearest distance between neighbor atoms

**max\_nbor\_size**

A list with ntypes integers, denotes the actual achieved max sel

## deepmd.utils.network module

```
deepmd.utils.network.embedding_net(xx, network_size, precision, activation_fn=<function tanh>,
                                   resnet_dt=False, name_suffix='', stddev=1.0, bavg=0.0,
                                   seed=None, trainable=True, uniform_seed=False,
                                   initial_variables=None, mixed_prec=None)
```

The embedding network.

The embedding network function  $\mathcal{N}$  is constructed by is the composition of multiple layers  $\mathcal{L}^{(i)}$ :

$$\mathcal{N} = \mathcal{L}^{(n)} \circ \mathcal{L}^{(n-1)} \circ \dots \circ \mathcal{L}^{(1)}$$

A layer  $\mathcal{L}$  is given by one of the following forms, depending on the number of nodes: [1]

$$y = \mathcal{L}(x; w, b) = \begin{cases} \phi(x^T w + b) + x, & N_2 = N_1 \\ \phi(x^T w + b) + (x, x), & N_2 = 2N_1 \\ \phi(x^T w + b), & \text{otherwise} \end{cases}$$

where  $x \in \mathbb{R}^{N_1}$  is the input vector and  $y \in \mathbb{R}^{N_2}$  is the output vector.  $w \in \mathbb{R}^{N_1 \times N_2}$  and  $b \in \mathbb{R}^{N_2}$  are weights and biases, respectively, both of which are trainable if trainable is True.  $\phi$  is the activation function.

Parameters

`xx`

[Tensor] Input tensor x of shape [-1,1]

`network_size`

[list of int] Size of the embedding network. For example [16,32,64]

`precision`:

Precision of network weights. For example, tf.float64

`activation_fn`:

Activation function  $\phi$

`resnet_dt`

[bool] Using time-step in the ResNet construction

`name_suffix`

[str] The name suffix append to each variable.

`stddev`

[float] Standard deviation of initializing network parameters

`bavg`

[float] Mean of network initial bias

`seed`

[int] Random seed for initializing network parameters

`trainable`

[bool] If the network is trainable

`uniform_seed`

[bool] Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

`initial_variables`

[dict] The input dict which stores the embedding net variables

`mixed_prec`  
The input dict which stores the mixed precision setting for the embedding net

## References

[1]

```
deepmd.utils.network.embedding_net_rand_seed_shift(network_size)

deepmd.utils.network.one_layer(inputs, outputs_size, activation_fn=<function tanh>,
                                precision=tf.float64, stddev=1.0, bavg=0.0, name='linear', scope='',
                                reuse=None, seed=None, use_timestep=False, trainable=True,
                                useBN=False, uniform_seed=False, initial_variables=None,
                                mixed_prec=None, final_layer=False)

deepmd.utils.network.one_layer_rand_seed_shift()

deepmd.utils.network.variable_summaries(var: VariableV1, name: str)
    Attach a lot of summaries to a Tensor (for TensorBoard visualization).

    Parameters
    var
        [tf.Variable] [description]
    name
        [str] variable name
```

## deepmd.utils.pair\_tab module

Alias for backward compatibility.

```
class deepmd.utils.pair_tab.PairTab(filename: str)
```

Bases: `object`

Pairwise tabulated potential.

Parameters

`filename`  
File name for the short-range tabulated potential. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columes. The first columen is the distance between atoms. The second to the last columes are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columes from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

## Methods

<code>get()</code>	Get the serialized table.
<code>reinit(filename)</code>	Initialize the tabulated interaction.

`get()` → `Tuple[array, array]`  
Get the serialized table.

`reinit(filename: str) → None`

Initialize the tabulated interaction.

Parameters

filename

File name for the short-range tabulated potential. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

## deepmd.utils.parallel\_op module

```
class deepmd.utils.parallel_op.ParallelOp(builder: Callable[[...], Tuple[Dict[str, Tensor],
                                                                    Tuple[Tensor]]], nthreads: Optional[int] = None, config:
                                                                    Optional[ConfigProto] = None)
```

Bases: `object`

Run an op with data parallelism.

Parameters

builder

`[Callable[...], Tuple[Dict[str, tf.Tensor], Tuple[tf.Tensor]]]` returns two objects: a dict which stores placeholders by key, and a tuple with the final op(s)

nthreads

`[int, optional]` the number of threads

config

`[tf.ConfigProto, optional]` `tf.ConfigProto`

## Examples

```
>>> from deepmd.env import tf
>>> from deepmd.utils.parallel_op import ParallelOp
>>> def builder():
...     x = tf.placeholder(tf.int32, [1])
...     return {"x": x}, (x + 1)
...
>>> p = ParallelOp(builder, nthreads=4)
>>> def feed():
...     for ii in range(10):
...         yield {"x": [ii]}
...
>>> print(*p.generate(tf.Session(), feed()))
[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
```

## Methods

<i>generate</i> (sess, feed)	Returns a generator.
------------------------------	----------------------

---

**generate**(sess: Session, feed: Generator[Dict[str, Any], None, None]) → Generator[Tuple, None, None]

Returns a generator.

Parameters

sess  
[tf.Session] TensorFlow session

feed  
[Generator[dict, None, None]] generator which yields feed\_dict

Yields

Generator[Tuple, None, None]  
generator which yields session returns

## deepmd.utils.path module

Alias for backward compatibility.

**class** deepmd.utils.path.DPH5Path(path: str)

Bases: *DPPath*

The path class to data system (DeepmdData) for HDF5 files.

Parameters

path  
[str] path

## Notes

OS - HDF5 relationship:  
directory - Group file - Dataset

## Methods

<i>glob</i> (pattern)	Search path using the glob pattern.
<i>is_dir</i> ()	Check if self is directory.
<i>is_file</i> ()	Check if self is file.
<i>load_numpy</i> ()	Load NumPy array.
<i>load_txt</i> ([dtype])	Load NumPy array from text.
<i>rglob</i> (pattern)	This is like calling <i>DPPath.glob()</i> with <i>**/</i> added in front of the given relative pattern.

`glob(pattern: str) → List[DPPath]`

Search path using the glob pattern.

Parameters

pattern  
[str] glob pattern

Returns

List[DPPath]  
list of paths

`is_dir() → bool`

Check if self is directory.

`is_file() → bool`

Check if self is file.

`load_numpy() → ndarray`

Load NumPy array.

Returns

np.ndarray  
loaded NumPy array

`load_txt(dtype: Optional[dtype] = None, **kwargs) → ndarray`

Load NumPy array from text.

Returns

np.ndarray  
loaded NumPy array

`rglob(pattern: str) → List[DPPath]`

This is like calling `DPPath.glob()` with `**/` added in front of the given relative pattern.

Parameters

pattern  
[str] glob pattern

Returns

List[DPPath]  
list of paths

`class deepmd.utils.path.DPOSPath(path: str)`

Bases: `DPPath`

The OS path class to data system (DeepmdData) for real directories.

Parameters

path  
[str] path



## Methods

<code>glob(pattern)</code>	Search path using the glob pattern.
<code>is_dir()</code>	Check if self is directory.
<code>is_file()</code>	Check if self is file.
<code>load_numpy()</code>	Load NumPy array.
<code>load_txt(**kwargs)</code>	Load NumPy array from text.
<code>rglob(pattern)</code>	This is like calling <code>DPPath.glob()</code> with <code>**/</code> added in front of the given relative pattern.

`glob(pattern: str) → List[DPPath]`

Search path using the glob pattern.

Parameters

pattern  
[`str`] glob pattern

Returns

`List[DPPath]`  
list of paths

`is_dir() → bool`

Check if self is directory.

`is_file() → bool`

Check if self is file.

`load_numpy() → ndarray`

Load NumPy array.

Returns

`np.ndarray`  
loaded NumPy array

`load_txt(**kwargs) → ndarray`

Load NumPy array from text.

Returns

`np.ndarray`  
loaded NumPy array

`rglob(pattern: str) → List[DPPath]`

This is like calling `DPPath.glob()` with `**/` added in front of the given relative pattern.

Parameters

pattern  
[`str`] glob pattern

Returns

`List[DPPath]`  
list of paths

```
class deepmd.utils.path.DPPath(path: str)
```

Bases: `ABC`

The path class to data system (DeepmdData).

Parameters

path  
[`str`] path

## Methods

<code>glob(pattern)</code>	Search path using the glob pattern.
<code>is_dir()</code>	Check if self is directory.
<code>is_file()</code>	Check if self is file.
<code>load_numpy()</code>	Load NumPy array.
<code>load_txt(**kwargs)</code>	Load NumPy array from text.
<code>rglob(pattern)</code>	This is like calling <code>DPPath.glob()</code> with <code>**/</code> added in front of the given relative pattern.

```
abstract glob(pattern: str) → List[DPPath]
```

Search path using the glob pattern.

Parameters

pattern  
[`str`] glob pattern

Returns

`List[DPPath]`  
list of paths

```
abstract is_dir() → bool
```

Check if self is directory.

```
abstract is_file() → bool
```

Check if self is file.

```
abstract load_numpy() → ndarray
```

Load NumPy array.

Returns

`np.ndarray`  
loaded NumPy array

```
abstract load_txt(**kwargs) → ndarray
```

Load NumPy array from text.

Returns

`np.ndarray`  
loaded NumPy array

```
abstract rglob(pattern: str) → List[DPPath]
```

This is like calling `DPPath.glob()` with `**/` added in front of the given relative pattern.

Parameters

pattern  
     [str] glob pattern  
 Returns  
     List[DPPath]  
     list of paths

### deepmd.utils.plugin module

Alias for backward compatibility.

**class** deepmd.utils.plugin.Plugin

Bases: `object`

A class to register and restore plugins.

### Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
def xxx():
    pass
>>> print(plugin.plugins['xx'])
```

Attributes

plugins  
     [Dict[str, object]] plugins

### Methods

<code>get_plugin(key)</code>	Visit a plugin by key.
<code>register(key)</code>	Register a plugin.

**get\_plugin**(key) → `object`

Visit a plugin by key.

Parameters

key  
     [str] key of the plugin

Returns

`object`  
     the plugin

**register**(key: str) → `Callable[[object], object]`

Register a plugin.

Parameters

key  
     [str] key of the plugin

Returns

`Callable[[object], object]`  
decorator

`class deepmd.utils.plugin.PluginVariant(*args, **kwargs)`

Bases: `object`

A class to remove type from input arguments.

`class deepmd.utils.plugin.VariantABCMeta(name, bases, namespace, **kwargs)`

Bases: `VariantMeta`, `ABCMeta`

### Methods

<code>__call__(*args, **kwargs)</code>	Remove type and keys that starts with under-line.
<code>mro()</code>	Return a type's method resolution order.
<code>register(subclass)</code>	Register a virtual subclass of an ABC.

`class deepmd.utils.plugin.VariantMeta`

Bases: `object`

### Methods

<code>__call__(*args, **kwargs)</code>	Remove type and keys that starts with under-line.
--	---

## deepmd.utils.random module

Alias for backward compatibility.

`deepmd.utils.random.choice(a: ndarray, p: Optional[ndarray] = None)`

Generates a random sample from a given 1-D array.

Parameters

`a`  
`[np.ndarray]` A random sample is generated from its elements.

`p`  
`[np.ndarray]` The probabilities associated with each entry in `a`.

Returns

`np.ndarray`  
arrays with results and their shapes

`deepmd.utils.random.random(size=None)`

Return random floats in the half-open interval `[0.0, 1.0)`.

Parameters

`size`  
Output shape.

Returns

`np.ndarray`  
Arrays with results and their shapes.

`deepmd.utils.random.seed(val: Optional[int] = None)`

Seed the generator.

Parameters

`val`  
`[int]` Seed.

`deepmd.utils.random.shuffle(x: ndarray)`

Modify a sequence in-place by shuffling its contents.

Parameters

`x`  
`[np.ndarray]` The array or list to be shuffled.

### deepmd.utils.sess module

`deepmd.utils.sess.run_sess(sess: Session, *args, **kwargs)`

Run session with errors caught.

Parameters

`sess`  
`[tf.Session]` TensorFlow Session

`*args`  
Variable length argument list.

`**kwargs`  
Arbitrary keyword arguments.

Returns

`Any`  
the result of `sess.run()`

### deepmd.utils.spin module

`class deepmd.utils.spin.Spin(use_spin: Optional[List[bool]] = None, spin_norm: Optional[List[float]] = None, virtual_len: Optional[List[float]] = None)`

Bases: `object`

Class for spin.

Parameters

`use_spin`  
Whether to use atomic spin model for each atom type

`spin_norm`  
The magnitude of atomic spin for each atom type with spin

`virtual_len`

The distance between virtual atom representing spin and its corresponding real atom for each atom type with spin

## Methods

<code>build([reuse, suffix])</code>	Build the computational graph for the spin.
<code>get_ntypes_spin()</code>	Returns the number of atom types which contain spin.
<code>get_spin_norm()</code>	Returns the list of magnitude of atomic spin for each atom type.
<code>get_use_spin()</code>	Returns the list of whether to use spin for each atom type.
<code>get_virtual_len()</code>	Returns the list of distance between real atom and virtual atom for each atom type.

`build(reuse=None, suffix='')`

Build the computational graph for the spin.

Parameters

`reuse`

The weights in the networks should be reused when get the variable.

`suffix`

Name suffix to identify this descriptor

Returns

`embedded_types`

The computational graph for embedded types

`get_ntypes_spin() → int`

Returns the number of atom types which contain spin.

`get_spin_norm() → List[float]`

Returns the list of magnitude of atomic spin for each atom type.

`get_use_spin() → List[bool]`

Returns the list of whether to use spin for each atom type.

`get_virtual_len() → List[float]`

Returns the list of distance between real atom and virtual atom for each atom type.

## deepmd.utils.tabulate module

```
class deepmd.utils.tabulate.DPTabulate(descrpt: ~deepmd.descriptor.descriptor.Descriptor, neuron:
    ~typing.List[int], graph:
    ~tensorflow.python.framework.ops.Graph, graph_def:
    ~tensorflow.core.framework.graph_pb2.GraphDef,
    type_one_side: bool = False, exclude_types:
    ~typing.List[~typing.List[int]] = [], activation_fn: ~typ-
    ing.Callable[~tensorflow.python.framework.tensor.Tensor],
    ~tensorflow.python.framework.tensor.Tensor] = <function
    tanh>, suffix: str = '')
```

Bases: `object`

Class for tabulation.

Compress a model, which including tabulating the embedding-net. The table is composed of fifth-order polynomial coefficients and is assembled from two sub-tables. The first table takes the `stride(parameter)` as it's uniform stride, while the second table takes  $10 * \text{stride}$  as it's uniform stride. The range of the first table is automatically detected by deepmd-kit, while the second table ranges from the first table's upper boundary(`upper`) to the `extrapolate(parameter) * upper`.

Parameters

`descript`

Descriptor of the original model

`neuron`

Number of neurons in each hidden layers of the embedding net  
*mathcal{N}*

`graph`

`[tf.Graph]` The graph of the original model

`graph_def`

`[tf.GraphDef]` The graph\_def of the original model

`type_one_side`

Try to build  $N_{\text{types}}$  tables. Otherwise, building  $N_{\text{types}}^2$  tables

`exclude_types`

`[List[List[int]]]` The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

`activation_function`

The activation function in the embedding net. Supported options are {"tanh", "gelu"} in common.ACTIVATION\_FN\_DICT.

`suffix`

`[str, optional]` The suffix of the scope

## Methods

---

`build(min_nbor_dist, extrapolate, stride0, ...)` Build the tables for model compression.

---

`build(min_nbor_dist: float, extrapolate: float, stride0: float, stride1: float) → Tuple[Dict[str, int], Dict[str, int]]`

Build the tables for model compression.

Parameters

`min_nbor_dist`

The nearest distance between neighbor atoms

`extrapolate`

The scale of model extrapolation

`stride0`

The uniform stride of the first table

stride1  
The uniform stride of the second table

Returns

lower  
[dict[str, int]] The lower boundary of environment matrix by net

upper  
[dict[str, int]] The upper boundary of environment matrix by net

### deepmd.utils.type\_embed module

```
class deepmd.utils.type_embed.TypeEmbedNet(neuron: List[int] = [], resnet_dt: bool = False,
                                             activation_function: Optional[str] = 'tanh', precision:
                                             str = 'default', trainable: bool = True, seed:
                                             Optional[int] = None, uniform_seed: bool = False,
                                             padding: bool = False, **kwargs)
```

Bases: `object`

Type embedding network.

Parameters

neuron  
[list[int]] Number of neurons in each hidden layers of the embedding net

resnet\_dt  
Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

activation\_function  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

precision  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

trainable  
If the weights of embedding net are trainable.

seed  
Random seed for initializing the network parameters.

uniform\_seed  
Only for the purpose of backward compatibility, retrieves the old behavior of using the random seed

padding  
Concat the zero padding to the output, as the default embedding of empty type.



## Methods

<code>build(ntypes[, reuse, suffix])</code>	Build the computational graph for the descriptor.
<code>init_variables(graph, graph_def[, suffix, ...])</code>	Init the type embedding net variables with the given dict.

**build**(ntypes: int, reuse=None, suffix='')

Build the computational graph for the descriptor.

Parameters

ntypes

Number of atom types.

reuse

The weights in the networks should be reused when get the variable.

suffix

Name suffix to identify this descriptor

Returns

**embedded\_types**

The computational graph for embedded types

**init\_variables**(graph: Graph, graph\_def: GraphDef, suffix='', model\_type='original\_model') →  
None

Init the type embedding net variables with the given dict.

Parameters

graph

[tf.Graph] The input frozen model graph

graph\_def

[tf.GraphDef] The input frozen model graph\_def

suffix

Name suffix to identify this descriptor

model\_type

Indicator of whether this model is a compressed model

**deepmd.utils.type\_embed.embed\_atom\_type**(ntypes: int, natoms: Tensor, type\_embedding: Tensor)

Make the embedded type for the atoms in system. The atoms are assumed to be sorted according to the type, thus their types are described by a tf.Tensor natoms, see explanation below.

Parameters

ntypes:

Number of types.

natoms:

The number of atoms. This tensor has the length of Ntypes + 2  
natoms[0]: number of local atoms  
natoms[1]: total number of atoms held by this processor  
natoms[i]: 2 ≤ i < Ntypes+2, number of type i atoms

type\_embedding:

The type embedding. It has the shape of [ntypes, embedding\_dim]

Returns

**atom\_embedding**

The embedded type of each atom. It has the shape of [numb\_atoms, embedding\_dim]

### deepmd.utils.weight\_avg module

Alias for backward compatibility.

deepmd.utils.weight\_avg.weighted\_average(errors: List[Dict[str, Tuple[float, float]]]) → Dict

Compute wighted average of prediction errors (MAE or RMSE) for model.

Parameters

errors

[List[Dict[str, Tuple[float, float]]]] List: the error of systems Dict: the error of quantities, name given by the key str: the name of the quantity, must starts with 'mae' or 'rmse' Tuple: (error, weight)

Returns

Dict

weighted averages

## 18.2.2 Submodules

### 18.2.3 deepmd.calculator module

ASE calculator interface module.

```
class deepmd.calculator.DP(model: Union[str, Path], label: str = 'DP', type_dict: Optional[Dict[str, int]] = None, neighbor_list=None, **kwargs)
```

Bases: Calculator

Implementation of ASE deepmd calculator.

Implemented propertie are energy, forces and stress

Parameters

model

[Union[str, Path]] path to the model

label

[str, optional] calculator label, by default "DP"

type\_dict

[Dict[str, int], optional] mapping of element types and their numbers, best left None and the calculator will infer this information from model, by default None

neighbor\_list

[ase.neighborlist.NeighborList, optional] The neighbor list object. If None, then build the native neighbor list.

## Examples

Compute potential energy

```
>>> from ase import Atoms
>>> from deepmd.calculator import DP
>>> water = Atoms('H2O',
>>>               positions=[(0.7601, 1.9270, 1),
>>>                           (1.9575, 1, 1),
>>>                           (1., 1., 1.)],
>>>               cell=[100, 100, 100],
>>>               calculator=DP(model="frozen_model.pb"))
>>> print(water.get_potential_energy())
>>> print(water.get_forces())
```

Run BFGS structure optimization

```
>>> from ase.optimize import BFGS
>>> dyn = BFGS(water)
>>> dyn.run(fmax=1e-6)
>>> print(water.get_positions())
```

Attributes

directory  
label

## Methods

<code>band_structure()</code>	Create band-structure object for plotting.
<code>calculate([atoms, properties, system_changes])</code>	Run calculation with deepmd model.
<code>calculate_numerical_forces(atoms[, d])</code>	Calculate numerical forces using finite difference.
<code>calculate_numerical_stress(atoms[, d, voigt])</code>	Calculate numerical stress using finite difference.
<code>calculate_properties(atoms, properties)</code>	This method is experimental; currently for internal use.
<code>check_state(atoms[, tol])</code>	Check for any system changes since last calculation.
<code>get_magnetic_moments([atoms])</code>	Calculate magnetic moments projected onto atoms.
<code>get_property(name[, atoms, allow_calculation])</code>	Get the named property.
<code>get_stresses([atoms])</code>	the calculator should return intensive stresses, i.e., such that <code>stresses.sum(axis=0) == stress</code>
<code>read(label)</code>	Read atoms, parameters and calculated properties from output file.
<code>reset()</code>	Clear all information from old calculation.
<code>set(**kwargs)</code>	Set parameters like <code>set(key1=value1, key2=value2, ...)</code> .
<code>set_label(label)</code>	Set label and convert label to directory and prefix.

calculation_required	
export_properties	
get_atoms	
get_charges	
get_default_parameters	
get_dipole_moment	
get_forces	
get_magnetic_moment	
get_potential_energies	
get_potential_energy	
get_stress	
read_atoms	
todict	

```
calculate(atoms: Optional[Atoms] = None, properties: List[str] = ['energy', 'forces', 'virial'],
          system_changes: List[str] = ['positions', 'numbers', 'cell', 'pbc', 'initial_charges',
          'initial_magmoms'])
```

Run calculation with deepmd model.

Parameters

atoms  
[Optional[Atoms], optional] atoms object to run the calculation on, by default None

properties  
[List[str], optional] unused, only for function signature compatibility, by default ["energy", "forces", "stress"]

system\_changes  
[List[str], optional] unused, only for function signature compatibility, by default all\_changes

```
implemented_properties: ClassVar[List[str]] = ['energy', 'free_energy', 'forces',
'virial', 'stress']
```

Properties calculator can handle (energy, forces, ...)

```
name = 'DP'
```

### 18.2.4 deepmd.common module

Collection of functions and classes used throughout the whole package.

```
deepmd.common.add_data_requirement(key: str, ndof: int, atomic: bool = False, must: bool = False,
                                   high_prec: bool = False, type_sel: Optional[bool] = None, repeat:
                                   int = 1, default: float = 0.0, dtype: Optional[dtype] = None)
```

Specify data requirements for training.

Parameters

key  
[str] type of data stored in corresponding \*.npz file e.g. forces or energy

ndof  
[int] number of the degrees of freedom, this is tied to atomic parameter e.g. forces have atomic=True and ndof=3

atomic  
 [bool, optional] specifies whether the ndof keyword applies to per atom quantity or not, by default False

must  
 [bool, optional] specifies if the \*.npy data file must exist, by default False

high\_prec  
 [bool, optional] if true load data to np.float64 else np.float32, by default False

type\_sel  
 [bool, optional] select only certain type of atoms, by default None

repeat  
 [int, optional] if specify repeat data repeat times, by default 1

default  
 [float, optional, default=0.] default value of data

dtype  
 [np.dtype, optional] the dtype of data, overwrites high\_prec if provided

`deepmd.common.cast_precision(func: Callable) → Callable`

A decorator that casts and casts back the input and output tensor of a method.

The decorator should be used in a classmethod.

The decorator will do the following thing: (1) It casts input Tensors from GLOBAL\_TF\_FLOAT\_PRECISION to precision defined by property precision. (2) It casts output Tensors from precision to GLOBAL\_TF\_FLOAT\_PRECISION. (3) It checks inputs and outputs and only casts when input or output is a Tensor and its dtype matches GLOBAL\_TF\_FLOAT\_PRECISION and precision, respectively. If it does not match (e.g. it is an integer), the decorator will do nothing on it.

Returns

**Callable**

a decorator that casts and casts back the input and output tensor of a method

## Examples

```
>>> class A:
...     @property
...     def precision(self):
...         return tf.float32
...
...     @cast_precision
...     def f(x: tf.Tensor, y: tf.Tensor) -> tf.Tensor:
...         return x ** 2 + y
```

`deepmd.common.clear_session()`

Reset all state generated by DeePMD-kit.

`deepmd.common.expand_sys_str(root_dir: Union[str, Path]) → List[str]`

Recursively iterate over directories taking those that contain type.raw file.

Parameters

root\_dir  
 [Union[str, Path]] starting directory

Returns

`List[str]`  
list of string pointing to system directories

`deepmd.common.gelu(x: Tensor) → Tensor`

Gaussian Error Linear Unit.

This is a smoother version of the RELU, implemented by custom operator.

Parameters

`x`  
`[tf.Tensor]` float Tensor to perform activation

Returns

`tf.Tensor`  
x with the GELU activation applied

## References

Original paper <https://arxiv.org/abs/1606.08415>

`deepmd.common.gelu_tf(x: Tensor) → Tensor`

Gaussian Error Linear Unit.

This is a smoother version of the RELU, implemented by TF.

Parameters

`x`  
`[tf.Tensor]` float Tensor to perform activation

Returns

`tf.Tensor`  
x with the GELU activation applied

## References

Original paper <https://arxiv.org/abs/1606.08415>

`deepmd.common.get_activation_func(activation_fn: Optional[_ACTIVATION]) →`  
`Optional[Callable[[Tensor], Tensor]]`

Get activation function callable based on string name.

Parameters

`activation_fn`  
`[_ACTIVATION]` one of the defined activation functions

Returns

`Callable[[tf.Tensor], tf.Tensor]`  
correspondingg TF callable

Raises

`RuntimeError`  
if unknown activation function is specified

`deepmd.common.get_np_precision(precision: _PRECISION) → dtype`

Get numpy precision constant from string.

Parameters

precision  
[\_PRECISION] string name of numpy constant or default

Returns

**np.dtype**  
numpy precision constant

Raises

**RuntimeError**  
if string is invalid

`deepmd.common.get_precision(precision: _PRECISION) → Any`

Convert str to TF DType constant.

Parameters

precision  
[\_PRECISION] one of the allowed precisions

Returns

**tf.python.framework.dtypes.DType**  
appropriate TF constant

Raises

**RuntimeError**  
if supplied precision string does not have a corresponding TF constant

`deepmd.common.j_loader(filename: Union[str, Path]) → Dict[str, Any]`

Load yaml or json settings file.

Parameters

filename  
[Union[str, Path]] path to file

Returns

**Dict[str, Any]**  
loaded dictionary

Raises

**TypeError**  
if the supplied file is of unsupported type

`deepmd.common.j_must_have(jdata: Dict[str, _DICT_VAL], key: str, deprecated_key: List[str] = []) → _DICT_VAL`

Assert that supplied dictionary contains specified key.

Returns

**\_DICT\_VAL**  
value that was stored under supplied key

Raises

`RuntimeError`

if the key is not present

`deepmd.common.make_default_mesh(pbc: bool, mixed_type: bool) → ndarray`

Make mesh.

Only the size of mesh matters, not the values: \* 6 for PBC, no mixed types \* 0 for no PBC, no mixed types \* 7 for PBC, mixed types \* 1 for no PBC, mixed types

Parameters

`pbc`

[`bool`] if True, the mesh will be made for periodic boundary conditions

`mixed_type`

[`bool`] if True, the mesh will be made for mixed types

Returns

`np.ndarray`

mesh

`deepmd.common.safe_cast_tensor(input: Tensor, from_precision: DType, to_precision: DType) → Tensor`

Convert a Tensor from a precision to another precision.

If input is not a Tensor or without the specific precision, the method will not cast it.

Parameters

`input`

[`tf.Tensor`] input tensor

`from_precision`

[`tf.DType`] Tensor data type that is casted from

`to_precision`

[`tf.DType`] Tensor data type that casts to

Returns

`tf.Tensor`

casted Tensor

`deepmd.common.select_idx_map(atom_types: ndarray, select_types: ndarray) → ndarray`

Build map of indices for element supplied element types from all atoms list.

Parameters

`atom_types`

[`np.ndarray`] array specifying type for each atoms as integer

`select_types`

[`np.ndarray`] types of atoms you want to find indices for

Returns

`np.ndarray`

indices of types of atoms defined by `select_types` in `atom_types` array

Warning: `select_types` array will be sorted before finding indices in `atom_types`



### 18.2.5 deepmd.env module

Module that sets tensorflow working environment and exports important constants.

`deepmd.env.GLOBAL_ENER_FLOAT_PRECISION`

alias of `float64`

`deepmd.env.GLOBAL_NP_FLOAT_PRECISION`

alias of `float64`

`deepmd.env.global_cvt_2_ener_float(xx: Tensor) → Tensor`

Cast tensor to globally set energy precision.

Parameters

`xx`

[`tf.Tensor`] input tensor

Returns

`tf.Tensor`

output tensor cast to `GLOBAL_ENER_FLOAT_PRECISION`

`deepmd.env.global_cvt_2_tf_float(xx: Tensor) → Tensor`

Cast tensor to globally set TF precision.

Parameters

`xx`

[`tf.Tensor`] input tensor

Returns

`tf.Tensor`

output tensor cast to `GLOBAL_TF_FLOAT_PRECISION`

`deepmd.env.reset_default_tf_session_config(cpu_only: bool)`

Limit tensorflow session to CPU or not.

Parameters

`cpu_only`

[`bool`] If enabled, no GPU device is visible to the TensorFlow Session.

### 18.2.6 deepmd.lmp module

Register entry points for lammmps-wheel.

`deepmd.lmp.get_env(paths: List[Optional[str]]) → str`

Get the environment variable from given paths.

`deepmd.lmp.get_library_path(module: str, filename: str) → List[str]`

Get library path from a module.

Parameters

`module`

[`str`] The module name.

`filename`

[`str`] The library filename pattern.

Returns

`list[str]`

The library path.

`deepmd.lmp.get_op_dir()` → `str`

Get the directory of the deepmd-kit OP library.

## 18.3 deepmd\_utils package

Utilization methods for DeePMD-kit.

The `__init__` module should not import any modules for performance.

### 18.3.1 Subpackages

**deepmd\_utils.entrpoints package**

**Submodules**

**deepmd\_utils.entrpoints.doc module**

Module that prints train input arguments docstrings.

`deepmd_utils.entrpoints.doc.doc_train_input(*, out_type: str = 'rst', **kwargs)`

Print out trining input arguments to console.

**deepmd\_utils.entrpoints.gui module**

DP-GUI entrypoint.

`deepmd_utils.entrpoints.gui.start_dpgui(*, port: int, bind_all: bool, **kwargs)`

Host DP-GUI server.

Parameters

`port`

`[int]` The port to serve DP-GUI on.

`bind_all`

`[bool]` Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available).

`**kwargs`

additional arguments

Raises

`ModuleNotFoundError`

The dpgui package is not installed

## deepmd\_utils.loggers package

Module taking care of logging duties.

`deepmd_utils.loggers.set_log_handles`(level: `int`, log\_path: `Optional[Path]` = None, mpi\_log: `Optional[str]` = None)

Set desired level for package loggers and add file handlers.

Parameters

level

[`int`] logging level

log\_path

[`Optional[str]`] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi\_log

[`Optional[str]`, `optional`] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

`RuntimeError`

If the argument mpi\_log is specified, package mpi4py is not installed.

## Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

## References

<https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U>      <https://stackoverflow.com/questions/35869137/avoid-tensorflow-print-on-standard-error>  
[avoid-tensorflow-print-on-standard-error](https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U)      <https://stackoverflow.com/questions/56085015/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu>  
[suppress-openmp-debug-messages-when-running-tensorflow-on-cpu](https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U)

## Submodules

### deepmd\_utils.loggers.loggers module

Logger initialization for package.

`deepmd_utils.loggers.loggers.set_log_handles`(level: `int`, log\_path: `Optional[Path]` = None, mpi\_log: `Optional[str]` = None)

Set desired level for package loggers and add file handlers.

Parameters

level

[`int`] logging level

log\_path

[`Optional[str]`] path to log file, if None logs will be send only to console. If the parent directory does not exist it will be automatically created, by default None

mpi\_log

[`Optional[str]`, `optional`] mpi log type. Has three options. master will output logs to file and console only from rank==0. collect will write messages from all ranks to one file opened under rank==0 and to console. workers will open one log file for each worker designated by its rank, console behaviour is the same as for collect. If this argument is specified, package 'mpi4py' must be already installed. by default None

Raises

`RuntimeError`

If the argument mpi\_log is specified, package mpi4py is not installed.

## Notes

Logging levels:

	our notation	python logging	tensorflow cpp	OpenMP
debug	10	10	0	1/on/true/yes
info	20	20	1	0/off/false/no
warning	30	30	2	0/off/false/no
error	40	40	3	0/off/false/no

## References

<https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U>

[avoid-tensorflow-print-on-standard-error](https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U/avoid-tensorflow-print-on-standard-error)

[suppress-openmp-debug-messages-when-running-tensorflow-on-cpu](https://groups.google.com/g/mpi4py/c/SaNzc8bdj6U/suppress-openmp-debug-messages-when-running-tensorflow-on-cpu)

<https://stackoverflow.com/questions/35869137/>

<https://stackoverflow.com/questions/56085015/>

## deepmd\_utils.model\_format package

```
class deepmd_utils.model_format.DescriptSeA(rcut: float, rcut_smth: float, sel: List[str], neuron:
    List[int] = [24, 48, 96], axis_neuron: int = 8, resnet_dt:
    bool = False, trainable: bool = True, type_one_side:
    bool = True, exclude_types: List[List[int]] = [],
    set_davg_zero: bool = False, activation_function: str =
    'tanh', precision: str = 'float64', spin: Optional[Any] =
    None)
```

Bases: *NativeOP*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor  $\mathcal{D}^i \in \mathbb{R}^{M_1 \times M_2}$  is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i$$

where  $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$  is the coordinate matrix, and each row of  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{s(r_{ji})x_{ji}} \\ \frac{r_{ji}^{j_i}}{s(r_{ji})y_{ji}} \\ \frac{r_{ji}^{j_i}}{s(r_{ji})z_{ji}} \\ \frac{r_{ji}^{j_i}}{r_{ji}} \end{bmatrix}$$

where  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$  consists of outputs of a embedding network  $\mathcal{N}$  of  $s(r_{ji})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}_{<}^i \in \mathbb{R}^{N \times M_2}$  takes first  $M_2$  columns of  $\mathcal{G}^i$ . The equation of embedding network  $\mathcal{N}$  can be found at `deepmd_utils.network.embedding_net()`.

Parameters

`rcut`

The cut-off radius  $r_c$

`rcut_smth`

From where the environment matrix should be smoothed  $r_s$

`sel`

`[list[str]]` `sel[i]` specifies the maximum number of type *i* atoms in the cut-off radius

`neuron`

`[list[int]]` Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

`axis_neuron`

Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

`resnet_dt`  
Time-step  $dt$  in the resnet construction:  $y = x + dt * \phi(Wx + b)$

`trainable`  
If the weights of embedding net are trainable.

`type_one_side`  
Try to build  $N_{\text{types}}$  embedding nets. Otherwise, building  $N_{\text{types}}^2$  embedding nets

`exclude_types`  
[[`List[List[int]]`]] The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

`set_davg_zero`  
Set the shift of embedding net input to zero.

`activation_function`  
The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

`precision`  
The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

`multi_task`  
If the model has multi fitting nets to train.

`spin`  
The deepspin object.

## References

[1]

## Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(coord_ext, atype_ext, nlist)</code>	Compute the descriptor.

<code>cal_g</code>	
<code>deserialize</code>	
<code>serialize</code>	

`cal_g(ss, ll)`

`call(coord_ext, atype_ext, nlist)`

Compute the descriptor.

Parameters

`coord_ext`

The extended coordinates of atoms. shape:  $nf \times (nall \times 3)$

`atype_ext`

The extended atom types. shape:  $nf \times nall$

**nlist**  
The neighbor list. shape: nf x nloc x nnei

Returns

**descriptor**  
The descriptor. shape: nf x nloc x ng x axis\_neuron

**classmethod deserialize**(data: dict) → DescriptSeA

**serialize**() → dict

deepmd\_utils.model\_format.EmbeddingNet  
alias of EN

**class** deepmd\_utils.model\_format.EnvMat(rcut, rcut\_smth)  
Bases: *NativeOP*

### Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(coord_ext, atype_ext, nlist[, davg, dstd])</code>	Compute the environment matrix.

deserialize	
serialize	

**call**(coord\_ext: ndarray, atype\_ext: ndarray, nlist: ndarray, davg: Optional[ndarray] = None, dstd: Optional[ndarray] = None) → ndarray

Compute the environment matrix.

Parameters

**nlist**  
The neighbor list. shape: nf x nloc x nnei

**coord\_ext**  
The extended coordinates of atoms. shape: nf x (nallx3)

**atype\_ext**  
The extended atom types. shape: nf x nall

**davg**  
The data avg. shape: nt x nnei x 4

**dstd**  
The inverse of data std. shape: nt x nnei x 4

Returns

**env\_mat**  
The environment matrix. shape: nf x nloc x nnei x 4

**switch**  
The value of switch function. shape: nf x nloc x nnei

**classmethod deserialize**(data: dict) → EnvMat

`serialize()` → dict

`deepmd_utils.model_format.FittingNet`

alias of FN

`class deepmd_utils.model_format.FittingOutputDef` (var\_defs: List[OutputVariableDef])

Bases: `object`

Defines the shapes and other properties of the fitting network outputs.

It is assume that the fitting network output variables for each local atom. This class defines all the outputs.

Parameters

var\_defs

List of output variable definitions.

## Methods

get_data	
keys	

`get_data()` → Dict[str, OutputVariableDef]

`keys()`

`class deepmd_utils.model_format.ModelOutputDef` (fit\_defs: FittingOutputDef)

Bases: `object`

Defines the shapes and other properties of the model outputs.

The model reduce and differentiate fitting outputs if applicable. If a variable is named by foo, then the reduced variable is called foo\_redu, the derivative w.r.t. coordinates is called foo\_derv\_r and the derivative w.r.t. cell is called foo\_derv\_c.

Parameters

fit\_defs

Definition for the fitting net output

## Methods

get_data	
keys	
keys_derv_c	
keys_derv_r	
keys_outp	
keys_redu	

`get_data(key: str)` → Dict[str, OutputVariableDef]

`keys()`



`keys_derv_c()`

`keys_derv_r()`

`keys_outp()`

`keys_redu()`

```
class deepmd_utils.model_format.NativeLayer(num_in, num_out, bias: bool = True, use_timestep:
    bool = False, activation_function: Optional[str] =
    None, resnet: bool = False, precision: str = 'float64')
```

Bases: *NativeOP*

Native representation of a layer.

Parameters

`w`  
[`np.ndarray`, optional] The weights of the layer.

`b`  
[`np.ndarray`, optional] The biases of the layer.

`idt`  
[`np.ndarray`, optional] The identity matrix of the layer.

`activation_function`  
[`str`, optional] The activation function of the layer.

`resnet`  
[`bool`, optional] Whether the layer is a residual layer.

## Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(x)</code>	Forward pass.
<code>deserialize(data)</code>	Deserialize the layer from a dict.
<code>serialize()</code>	Serialize the layer to a dict.

<code>check_shape_consistency</code>	
<code>check_type_consistency</code>	
<code>dim_in</code>	
<code>dim_out</code>	

`call(x: ndarray) → ndarray`

Forward pass.

Parameters

`x`  
[`np.ndarray`] The input.

Returns

`np.ndarray`  
The output.

`check_shape_consistency()`

`check_type_consistency()`

`classmethod deserialize(data: dict) → NativeLayer`

Deserialize the layer from a dict.

Parameters

`data`

[dict] The dict to deserialize from.

`dim_in() → int`

`dim_out() → int`

`serialize() → dict`

Serialize the layer to a dict.

Returns

dict

The serialized layer.

`deepmd_utils.model_format.NativeNet`

alias of NN

`class deepmd_utils.model_format.NativeOP`

Bases: ABC

The unit operation of a native model.

## Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(*args, **kwargs)</code>	Forward pass in NumPy implementation.

`call(*args, **kwargs)`

Forward pass in NumPy implementation.

`class deepmd_utils.model_format.NetworkCollection(ndim: int, ntypes: int, network_type: str =`

'network', networks: ~typing.

List[~typing.Union[~deepmd\_utils.model\_format.network.m  
dict]] = [])

Bases: object

A collection of networks for multiple elements.

The number of dimesions for types might be 0, 1, or 2. - 0: embedding or fitting with type embedding, in () - 1: embedding with type\_one\_side, or fitting, in (type\_i) - 2: embedding without type\_one\_side, in (type\_i, type\_j)

Parameters

`ndim`

[int] The number of dimensions.

`network_type`

[str, optional] The type of the network.

networks  
 [dict, optional] The networks to initialize with.

## Methods

<code>check_completeness()</code>	Check whether the collection is complete.
<code>deserialize(data)</code>	Deserialize the networks from a dict.
<code>serialize()</code>	Serialize the networks to a dict.

```
NETWORK_TYPE_MAP: ClassVar[Dict[str, type]] = {'embedding_network': <class
'deepmd_utils.model_format.network.make_embedding_network.<locals>.EN'>,
'fitting_network': <class
'deepmd_utils.model_format.network.make_fitting_network.<locals>.FN'>, 'network':
<class 'deepmd_utils.model_format.network.make_multilayer_network.<locals>.NN'>}
```

`check_completeness()`

Check whether the collection is complete.

Raises

`RuntimeError`

If the collection is incomplete.

`classmethod deserialize(data: dict) → NetworkCollection`

Deserialize the networks from a dict.

Parameters

data

[dict] The dict to deserialize from.

`serialize() → dict`

Serialize the networks to a dict.

Returns

dict

The serialized networks.

```
class deepmd_utils.model_format.OutputVariableDef(name: str, shape: List[int], reducible: bool =
False, differentiable: bool = False, atomic: bool
= True)
```

Bases: `object`

Defines the shape and other properties of the one output variable.

It is assume that the fitting network output variables for each local atom. This class defines one output variable, including its name, shape, reducibility and differentiability.

Parameters

name

Name of the output variable. Notice that the xxxx\_redu, xxxx\_derv\_c, xxxx\_derv\_r are reserved names that should not be used to define variables.

shape

The shape of the variable. e.g. energy should be [1], dipole should be [3], polarizability should be [3,3].

reducible

If the variable is reduced.

differentiable

If the variable is differentiated with respect to coordinates of atoms and cell tensor (pbc case). Only reducible variable are differentiable.

`deepmd_utils.model_format.fitting_check_output(cls)`

Check if the output of the Fitting is consistent with the definition.

Two methods are assumed to be provided by the Fitting: 1. `Fitting.output_def` that gives the output definition. 2. `Fitting.__call__` defines the forward path of the fitting.

`deepmd_utils.model_format.get_deriv_name(name: str) → Tuple[str, str]`

`deepmd_utils.model_format.get_reduce_name(name: str) → str`

`deepmd_utils.model_format.load_dp_model(filename: str) → dict`

Load a DP model from a file in the native format.

Parameters

filename

[str] The filename to load from.

Returns

dict

The loaded model dict, including meta information.

`deepmd_utils.model_format.make_embedding_network(T_Network, T_NetworkLayer)`

`deepmd_utils.model_format.make_fitting_network(T_EmbeddingNet, T_Network, T_NetworkLayer)`

`deepmd_utils.model_format.make_multilayer_network(T_NetworkLayer, ModuleBase)`

`deepmd_utils.model_format.model_check_output(cls)`

Check if the output of the Model is consistent with the definition.

Two methods are assumed to be provided by the Model: 1. `Model.output_def` that gives the output definition. 2. `Model.__call__` that defines the forward path of the model.

`deepmd_utils.model_format.save_dp_model(filename: str, model_dict: dict, extra_info: Optional[dict] = None)`

Save a DP model to a file in the native format.

Parameters

filename

[str] The filename to save to.

model\_dict

[dict] The model dict to save.

extra\_info

[dict, optional] Extra meta information to save.

`deepmd_utils.model_format.traverse_model_dict(model_obj, callback: callable, is_variable: bool = False)`

Traverse a model dict and call callback on each variable.

Parameters

`model_obj`  
 [`object`] The model object to traverse.  
`callback`  
 [`callable()`] The callback function to call on each variable.  
`is_variable`  
 [`bool`, `optional`] Whether the current node is a variable.  
 Returns  
`object`  
 The model object after traversing.

## Submodules

### deepmd\_utils.model\_format.common module

`class deepmd_utils.model_format.common.NativeOP`

Bases: `ABC`

The unit operation of a native model.

#### Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(*args, **kwargs)</code>	Forward pass in NumPy implementation.

`call(*args, **kwargs)`  
 Forward pass in NumPy implementation.

### deepmd\_utils.model\_format.env\_mat module

`class deepmd_utils.model_format.env_mat.EnvMat(rcut, rcut_smth)`

Bases: `NativeOP`

#### Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(coord_ext, atype_ext, nlist[, davg, dstd])</code>	Compute the environment matrix.

deserialize	
serialize	

`call(coord_ext: ndarray, atype_ext: ndarray, nlist: ndarray, davg: Optional[ndarray] = None, dstd: Optional[ndarray] = None) → ndarray`

Compute the environment matrix.

Parameters

**nlist**  
The neighbor list. shape:  $nf \times nloc \times nnei$

**coord\_ext**  
The extended coordinates of atoms. shape:  $nf \times (nall \times 3)$

**atype\_ext**  
The extended atom types. shape:  $nf \times nall$

**davg**  
The data avg. shape:  $nt \times nnei \times 4$

**dstd**  
The inverse of data std. shape:  $nt \times nnei \times 4$

Returns

**env\_mat**  
The environment matrix. shape:  $nf \times nloc \times nnei \times 4$

**switch**  
The value of switch function. shape:  $nf \times nloc \times nnei$

**classmethod deserialize**(data: dict) → EnvMat

**serialize**() → dict

`deepmd_utils.model_format.env_mat.compute_smooth_weight`(distance: ndarray, rmin: float, rmax: float)

Compute smooth weight for descriptor elements.

### **deepmd\_utils.model\_format.network module**

Native DP model format for multiple backends.

See issue #2982 for more information.

**class** `deepmd_utils.model_format.network.Counter`

Bases: `object`

A callable counter.

### **Examples**

```
>>> counter = Counter()
>>> counter()
0
>>> counter()
1
```

## Methods

<code>__call__()</code>	Call self as a function.
-------------------------	--------------------------

`deepmd_utils.model_format.network.EmbeddingNet`

alias of EN

`deepmd_utils.model_format.network.FittingNet`

alias of FN

```
class deepmd_utils.model_format.network.NativeLayer(num_in, num_out, bias: bool = True,
                                                    use_timestep: bool = False,
                                                    activation_function: Optional[str] = None,
                                                    resnet: bool = False, precision: str =
                                                    'float64')
```

Bases: *NativeOP*

Native representation of a layer.

Parameters

`w`  
[`np.ndarray`, optional] The weights of the layer.

`b`  
[`np.ndarray`, optional] The biases of the layer.

`idt`  
[`np.ndarray`, optional] The identity matrix of the layer.

`activation_function`  
[`str`, optional] The activation function of the layer.

`resnet`  
[`bool`, optional] Whether the layer is a residual layer.

## Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(x)</code>	Forward pass.
<code>deserialize(data)</code>	Deserialize the layer from a dict.
<code>serialize()</code>	Serialize the layer to a dict.

<code>check_shape_consistency</code>	
<code>check_type_consistency</code>	
<code>dim_in</code>	
<code>dim_out</code>	

`call(x: ndarray) → ndarray`

Forward pass.

Parameters

`x`  
[`np.ndarray`] The input.

Returns

`np.ndarray`  
The output.

`check_shape_consistency()`

`check_type_consistency()`

**classmethod** `deserialize(data: dict) → NativeLayer`

Deserialize the layer from a dict.

Parameters

`data`  
[dict] The dict to deserialize from.

`dim_in() → int`

`dim_out() → int`

**serialize()** `→ dict`

Serialize the layer to a dict.

Returns

`dict`  
The serialized layer.

`deepmd_utils.model_format.network.NativeNet`

alias of `NN`

**class** `deepmd_utils.model_format.network.NetworkCollection(ndim: int, ntypes: int, network_type: str = 'network', networks: ~typing.List[~typing.Union[~deepmd_utils.model_format.network.NativeNet, ~deepmd_utils.model_format.network.NativeLayer]]) = []`

Bases: `object`

A collection of networks for multiple elements.

The number of dimesions for types might be 0, 1, or 2. - 0: embedding or fitting with type embedding, in () - 1: embedding with type\_one\_side, or fitting, in (type\_i) - 2: embedding without type\_one\_side, in (type\_i, type\_j)

Parameters

`ndim`  
[int] The number of dimensions.

`network_type`  
[str, optional] The type of the network.

`networks`  
[dict, optional] The networks to initialize with.



## Methods

<code>check_completeness()</code>	Check whether the collection is complete.
<code>deserialize(data)</code>	Deserialize the networks from a dict.
<code>serialize()</code>	Serialize the networks to a dict.

```
NETWORK_TYPE_MAP: ClassVar[Dict[str, type]] = {'embedding_network': <class
'deepmd_utils.model_format.network.make_embedding_network.<locals>.EN'>,
'fitting_network': <class
'deepmd_utils.model_format.network.make_fitting_network.<locals>.FN'>, 'network':
<class 'deepmd_utils.model_format.network.make_multilayer_network.<locals>.NN'>}
```

`check_completeness()`

Check whether the collection is complete.

Raises

`RuntimeError`

If the collection is incomplete.

`classmethod deserialize(data: dict) → NetworkCollection`

Deserialize the networks from a dict.

Parameters

`data`

[`dict`] The dict to deserialize from.

`serialize() → dict`

Serialize the networks to a dict.

Returns

`dict`

The serialized networks.

`deepmd_utils.model_format.network.load_dp_model(filename: str) → dict`

Load a DP model from a file in the native format.

Parameters

`filename`

[`str`] The filename to load from.

Returns

`dict`

The loaded model dict, including meta information.

`deepmd_utils.model_format.network.make_embedding_network(T_Network, T_NetworkLayer)`

`deepmd_utils.model_format.network.make_fitting_network(T_EmbeddingNet, T_Network,`  
`T_NetworkLayer)`

`deepmd_utils.model_format.network.make_multilayer_network(T_NetworkLayer, ModuleBase)`

`deepmd_utils.model_format.network.save_dp_model(filename: str, model_dict: dict, extra_info:`  
`Optional[dict] = None)`

Save a DP model to a file in the native format.

## Parameters

filename  
[[str](#)] The filename to save to.

model\_dict  
[[dict](#)] The model dict to save.

extra\_info  
[[dict](#), [optional](#)] Extra meta information to save.

`deepmd_utils.model_format.network.traverse_model_dict(model_obj, callback: callable, is_variable: bool = False)`

Traverse a model dict and call callback on each variable.

## Parameters

model\_obj  
[[object](#)] The model object to traverse.

callback  
[[callable\(\)](#)] The callback function to call on each variable.

is\_variable  
[[bool](#), [optional](#)] Whether the current node is a variable.

## Returns

[object](#)  
The model object after traversing.

**deepmd\_utils.model\_format.output\_def module**

`class deepmd_utils.model_format.output_def.FittingOutputDef(var_defs: List[OutputVariableDef])`

Bases: [object](#)

Defines the shapes and other properties of the fitting network outputs.

It is assume that the fitting network output variables for each local atom. This class defines all the outputs.

## Parameters

var\_defs  
List of output variable definitions.

**Methods**

<a href="#">get_data</a>	
<a href="#">keys</a>	

`get_data()` → [Dict](#)[[str](#), [OutputVariableDef](#)]

`keys()`

```
class deepmd_utils.model_format.output_def.ModelOutputDef(fit_defs: FittingOutputDef)
```

Bases: `object`

Defines the shapes and other properties of the model outputs.

The model reduce and differentiate fitting outputs if applicable. If a variable is named by foo, then the reduced variable is called foo\_redu, the derivative w.r.t. coordinates is called foo\_derv\_r and the derivative w.r.t. cell is called foo\_derv\_c.

Parameters

fit\_defs

Definition for the fitting net output

## Methods

get_data	
keys	
keys_derv_c	
keys_derv_r	
keys_outp	
keys_redu	

`get_data(key: str) → Dict[str, OutputVariableDef]`

`keys()`

`keys_derv_c()`

`keys_derv_r()`

`keys_outp()`

`keys_redu()`

```
class deepmd_utils.model_format.output_def.OutputVariableDef(name: str, shape: List[int],
                                                             reducible: bool = False,
                                                             differentiable: bool = False,
                                                             atomic: bool = True)
```

Bases: `object`

Defines the shape and other properties of the one output variable.

It is assume that the fitting network output variables for each local atom. This class defines one output variable, including its name, shape, reducibility and differentiability.

Parameters

name

Name of the output variable. Notice that the xxxx\_redu, xxxx\_derv\_c, xxxx\_derv\_r are reserved names that should not be used to define variables.

shape

The shape of the variable. e.g. energy should be [1], dipole should be [3], polarizability should be [3,3].

reducible

If the variable is reduced.

differentiable

If the variable is differentiated with respect to coordinates of atoms and cell tensor (pbc case). Only reducible variable are differentiable.

`deepmd_utils.model_format.output_def.check_shape(shape: List[int], def_shape: List[int])`

Check if the shape satisfies the defined shape.

`deepmd_utils.model_format.output_def.check_var(var, var_def)`

`deepmd_utils.model_format.output_def.do_derivative(def_outp: FittingOutputDef) → Tuple[Dict[str, OutputVariableDef], Dict[str, OutputVariableDef]]`

`deepmd_utils.model_format.output_def.do_reduce(def_outp: FittingOutputDef) → Dict[str, OutputVariableDef]`

`deepmd_utils.model_format.output_def.fitting_check_output(cls)`

Check if the output of the Fitting is consistent with the definition.

Two methods are assumed to be provided by the Fitting: 1. Fitting.output\_def that gives the output definition. 2. Fitting.\_\_call\_\_ defines the forward path of the fitting.

`deepmd_utils.model_format.output_def.get_deriv_name(name: str) → Tuple[str, str]`

`deepmd_utils.model_format.output_def.get_reduce_name(name: str) → str`

`deepmd_utils.model_format.output_def.model_check_output(cls)`

Check if the output of the Model is consistent with the definition.

Two methods are assumed to be provided by the Model: 1. Model.output\_def that gives the output definition. 2. Model.\_\_call\_\_ that defines the forward path of the model.

## deepmd\_utils.model\_format.se\_e2\_a module

```
class deepmd_utils.model_format.se_e2_a.DescriptSeA(rcut: float, rcut_smth: float, sel: List[str],
    neuron: List[int] = [24, 48, 96], axis_neuron:
    int = 8, resnet_dt: bool = False, trainable:
    bool = True, type_one_side: bool = True,
    exclude_types: List[List[int]] = [],
    set_davg_zero: bool = False,
    activation_function: str = 'tanh', precision:
    str = 'float64', spin: Optional[Any] = None)
```

Bases: *NativeOP*

DeepPot-SE constructed from all information (both angular and radial) of atomic configurations. The embedding takes the distance between atoms as input.

The descriptor  $\mathcal{D}^i \in \mathbb{R}^{M_1 \times M_2}$  is given by [1]

$$\mathcal{D}^i = (\mathcal{G}^i)^T \mathcal{R}^i (\mathcal{R}^i)^T \mathcal{G}_{<}^i$$

where  $\mathcal{R}^i \in \mathbb{R}^{N \times 4}$  is the coordinate matrix, and each row of  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{r_{ji}} \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{bmatrix}$$

where  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Each row of the embedding matrix  $\mathcal{G}^i \in \mathbb{R}^{N \times M_1}$  consists of outputs of a embedding network  $\mathcal{N}$  of  $s(r_{ji})$ :

$$(\mathcal{G}^i)_j = \mathcal{N}(s(r_{ji}))$$

$\mathcal{G}^i_{<} \in \mathbb{R}^{N \times M_2}$  takes first  $M_2$  columns of  $\mathcal{G}^i$ . The equation of embedding network  $\mathcal{N}$  can be found at `deepmd.utils.network.embedding_net()`.

Parameters

rcut

The cut-off radius  $r_c$

rcut\_smth

From where the environment matrix should be smoothed  $r_s$

sel

`[list[str]]` sel[i] specifies the maximum number of type i atoms in the cut-off radius

neuron

`[list[int]]` Number of neurons in each hidden layers of the embedding net  $\mathcal{N}$

axis\_neuron

Number of the axis neuron  $M_2$  (number of columns of the sub-matrix of the embedding matrix)

resnet\_dt

Time-step dt in the resnet construction:  $y = x + dt * \phi(Wx + b)$

trainable

If the weights of embedding net are trainable.

type\_one\_side

Try to build  $N_{types}$  embedding nets. Otherwise, building  $N_{types}^2$  embedding nets

exclude\_types

`[List[List[int]]]` The excluded pairs of types which have no interaction with each other. For example, `[[0, 1]]` means no interaction between type 0 and type 1.

set\_davg\_zero

Set the shift of embedding net input to zero.

activation\_function

The activation function in the embedding net. Supported options are “relu”, “relu6”, “softplus”, “sigmoid”, “tanh”, “gelu”, “gelu\_tf”, “None”, “none”.

precision

The precision of the embedding net parameters. Supported options are “default”, “float16”, “float32”, “float64”, “bfloat16”.

multi\_task

If the model has multi fitting nets to train.

spin

The deepspin object.

## References

[1]

## Methods

<code>__call__(*args, **kwargs)</code>	Forward pass in NumPy implementation.
<code>call(coord_ext, atype_ext, nlist)</code>	Compute the descriptor.

<code>cal_g</code>	
<code>deserialize</code>	
<code>serialize</code>	

`cal_g(ss, ll)`

`call(coord_ext, atype_ext, nlist)`

Compute the descriptor.

Parameters

`coord_ext`

The extended coordinates of atoms. shape:  $nf \times (nall \times 3)$

`atype_ext`

The extended atom types. shape:  $nf \times nall$

`nlist`

The neighbor list. shape:  $nf \times nloc \times nnei$

Returns

**descriptor**

The descriptor. shape:  $nf \times nloc \times ng \times axis\_neuron$

**classmethod** `deserialize(data: dict) → DescriptSeA`

**serialize()** → dict

## deepmd\_utils.utils package

### Submodules

#### deepmd\_utils.utils.argcheck module

**class** `deepmd_utils.utils.argcheck.ArgsPlugin`

Bases: `object`

## Methods

<code>get_all_argument([exclude_hybrid])</code>	Get all arguments.
<code>register(name[, alias])</code>	Register a descriptor argument plugin.

**get\_all\_argument** (exclude\_hybrid: bool = False) → List[Argument]

Get all arguments.

Parameters

exclude\_hybrid  
[bool] exclude hybrid descriptor to prevent circular calls

Returns

List[Argument]  
all arguments

**register** (name: str, alias: Optional[List[str]] = None) → Callable[[], List[Argument]]

Register a descriptor argument plugin.

Parameters

name  
[str] the name of a descriptor  
alias  
[List[str], optional] the list of aliases of this descriptor

Returns

Callable[[], List[Argument]]  
the registered descriptor argument method

## Examples

```
>>> some_plugin = ArgsPlugin()
>>> @some_plugin.register("some_descrpt")
def descrpt_some_descrpt_args():
    return []
```

```
deepmd_utils.utils.argcheck.descrpt_hybrid_args()
```

```
deepmd_utils.utils.argcheck.descrpt_local_frame_args()
```

```
deepmd_utils.utils.argcheck.descrpt_se_a_args()
```

```
deepmd_utils.utils.argcheck.descrpt_se_a_ebd_v2_args()
```

```
deepmd_utils.utils.argcheck.descrpt_se_a_mask_args()
```

```
deepmd_utils.utils.argcheck.descrpt_se_a_tpe_args()
```

```
deepmd_utils.utils.argcheck.descrpt_se_atten_args()
```

```
deepmd_utils.utils.argcheck.descrpt_se_atten_common_args()
```

```
deepmd_utils.utils.argcheck.descrpt_se_atten_v2_args()
```

```
deepmd_utils.utils.argcheck.descript_se_r_args()
deepmd_utils.utils.argcheck.descript_se_t_args()
deepmd_utils.utils.argcheck.descript_variant_type_args(exclude_hybrid: bool = False) → Variant
deepmd_utils.utils.argcheck.fitting_dipole()
deepmd_utils.utils.argcheck.fitting_dos()
deepmd_utils.utils.argcheck.fitting_ener()
deepmd_utils.utils.argcheck.fitting_polar()
deepmd_utils.utils.argcheck.fitting_variant_type_args()
deepmd_utils.utils.argcheck.frozen_model_args() → Argument
deepmd_utils.utils.argcheck.gen_args(**kwargs) → List[Argument]
deepmd_utils.utils.argcheck.gen_doc(*, make_anchor=True, make_link=True, **kwargs)
deepmd_utils.utils.argcheck.gen_json(**kwargs)
deepmd_utils.utils.argcheck.learning_rate_args()
deepmd_utils.utils.argcheck.learning_rate_dict_args()
deepmd_utils.utils.argcheck.learning_rate_exp()
deepmd_utils.utils.argcheck.learning_rate_variant_type_args()
deepmd_utils.utils.argcheck.limit_pref(item)
deepmd_utils.utils.argcheck.linear_ener_model_args() → Argument
deepmd_utils.utils.argcheck.list_to_doc(xx)
deepmd_utils.utils.argcheck.loss_args()
deepmd_utils.utils.argcheck.loss_dict_args()
deepmd_utils.utils.argcheck.loss_dos()
deepmd_utils.utils.argcheck.loss_ener()
deepmd_utils.utils.argcheck.loss_ener_spin()
deepmd_utils.utils.argcheck.loss_tensor()
deepmd_utils.utils.argcheck.loss_variant_type_args()
deepmd_utils.utils.argcheck.make_index(keys)
deepmd_utils.utils.argcheck.make_link(content, ref_key)
deepmd_utils.utils.argcheck.mixed_precision_args()
deepmd_utils.utils.argcheck.model_args(exclude_hybrid=False)
deepmd_utils.utils.argcheck.model_compression()
```



```

deepmd_utils.utils.argcheck.model_compression_type_args()
deepmd_utils.utils.argcheck.modifier_dipole_charge()
deepmd_utils.utils.argcheck.modifier_variant_type_args()
deepmd_utils.utils.argcheck.multi_model_args() → Argument
deepmd_utils.utils.argcheck.normalize(data)
deepmd_utils.utils.argcheck.normalize_data_dict(data_dict)
deepmd_utils.utils.argcheck.normalize_fitting_net_dict(fitting_net_dict)
deepmd_utils.utils.argcheck.normalize_fitting_weight(fitting_keys, data_keys,
                                                       fitting_weight=None)
deepmd_utils.utils.argcheck.normalize_learning_rate_dict(fitting_keys, learning_rate_dict)
deepmd_utils.utils.argcheck.normalize_learning_rate_dict_with_single_learning_rate(fitting_keys,
                                                                                     learn-
                                                                                     ing_rate)

deepmd_utils.utils.argcheck.normalize_loss_dict(fitting_keys, loss_dict)
deepmd_utils.utils.argcheck.normalize_multi_task(data)
deepmd_utils.utils.argcheck.pairtab_model_args() → Argument
deepmd_utils.utils.argcheck.pairwise_dprc() → Argument
deepmd_utils.utils.argcheck.spin_args()
deepmd_utils.utils.argcheck.standard_model_args() → Argument
deepmd_utils.utils.argcheck.start_pref(item, label=None, abbr=None)
deepmd_utils.utils.argcheck.training_args()
deepmd_utils.utils.argcheck.training_data_args()
deepmd_utils.utils.argcheck.type_embedding_args()
deepmd_utils.utils.argcheck.validation_data_args()

```

#### **deepmd\_utils.utils.argcheck\_nvnmd module**

```

deepmd_utils.utils.argcheck_nvnmd.nvnmd_args()

```

**deepmd\_utils.utils.batch\_size module**

```
class deepmd_utils.utils.batch_size.AutoBatchSize(initial_batch_size: int = 1024, factor: float = 2.0)
```

Bases: ABC

This class allows DeePMD-kit to automatically decide the maximum batch size that will not cause an OOM error.

Parameters

`initial_batch_size`  
 [`int`, default: 1024] initial batch size (number of total atoms) when DP\_INFER\_BATCH\_SIZE is not set

`factor`  
 [`float`, default: 2.] increased factor

**Notes**

In some CPU environments, the program may be directly killed when OOM. In this case, by default the batch size will not be increased for CPUs. The environment variable DP\_INFER\_BATCH\_SIZE can be set as the batch size.

In other cases, we assume all OOM error will raise `OutOfMemoryError`.

Attributes

`current_batch_size`  
 [`int`] current batch size (number of total atoms)

`maximum_working_batch_size`  
 [`int`] maximum working batch size

`minimal_not_working_batch_size`  
 [`int`] minimal not working batch size

**Methods**

<code>execute(callable, start_index, natoms)</code>	Excuate a method with given batch size.
<code>execute_all(callable, total_size, natoms, ...)</code>	Excuate a method with all given data.
<code>is_gpu_available()</code>	Check if GPU is available.
<code>is_oom_error(e)</code>	Check if the exception is an OOM error.

**execute**(callable: Callable, start\_index: int, natoms: int) → Tuple[int, tuple]

Excuate a method with given batch size.

Parameters

`callable`  
 [Callable] The method should accept the batch size and start\_index as parameters, and returns executed batch size and data.

`start_index`  
 [int] start index

```

    natoms
        [int] natoms
Returns
    int
        executed batch size * number of atoms
    tuple
        result from callable, None if failing to execute
Raises
    OutOfMemoryError
        OOM when batch size is 1
execute_all(callable: Callable, total_size: int, natoms: int, *args, **kwargs) → Tuple[ndarray]
    Excuate a method with all given data.
Parameters
    callable
        [Callable] The method should accept *args and **kwargs as input and return the
        similiar array.
    total_size
        [int] Total size
    natoms
        [int] The number of atoms
    *args
        Variable length argument list.
    **kwargs
        If 2D np.ndarray, assume the first axis is batch; otherwise do nothing.
abstract is_gpu_available() → bool
    Check if GPU is available.
Returns
    bool
        True if GPU is available
abstract is_oom_error(e: Exception) → bool
    Check if the exception is an OOM error.
Parameters
    e
        [Exception] Exception
Returns
    bool
        True if the exception is an OOM error

```

**deepmd\_utils.utils.compat module**

Module providing compatibility between 0.x.x and 1.x.x input versions.

`deepmd_utils.utils.compat.convert_input_v0_v1(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]`

Convert input from v0 format to v1.

Parameters

`jdata`

[Dict[str, Any]] loaded json/yaml file

`warning`

[bool, optional] whether to show deprecation warning, by default True

`dump`

[Optional[Union[str, Path]], optional] whether to dump converted file, by default None

Returns

Dict[str, Any]

converted output

`deepmd_utils.utils.compat.convert_input_v1_v2(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]`

`deepmd_utils.utils.compat.deprecate_numb_test(jdata: Dict[str, Any], warning: bool = True, dump: Optional[Union[str, Path]] = None) → Dict[str, Any]`

Deprecate `numb_test` since v2.1. It has taken no effect since v2.0.

See [#1243](#).

Parameters

`jdata`

[Dict[str, Any]] loaded json/yaml file

`warning`

[bool, optional] whether to show deprecation warning, by default True

`dump`

[Optional[Union[str, Path]], optional] whether to dump converted file, by default None

Returns

Dict[str, Any]

converted output

`deepmd_utils.utils.compat.remove_decay_rate(jdata: Dict[str, Any])`

Convert `decay_rate` to `stop_lr`.

Parameters

`jdata`

[Dict[str, Any]] input data

---

```
deepmd_utils.utils.compat.update_deepmd_input(jdata: Dict[str, Any], warning: bool = True, dump:
Optional[Union[str, Path]] = None) → Dict[str,
Any]
```

### deepmd\_utils.utils.data module

```
class deepmd_utils.utils.data.DeepmdData(sys_path: str, set_prefix: str = 'set', shuffle_test: bool =
True, type_map: Optional[List[str]] = None,
optional_type_map: bool = True, modifier=None,
trn_all_set: bool = False, sort_atoms: bool = True)
```

Bases: `object`

Class for a data system.

It loads data from hard disk, and maintains the data as a `data_dict`

Parameters

`sys_path`  
Path to the data system

`set_prefix`  
Prefix for the directories of different sets

`shuffle_test`  
If the test data are shuffled

`type_map`  
Gives the name of different atom types

`optional_type_map`  
If the `type_map.raw` in each system is optional

`modifier`  
Data modifier that has the method `modify_data`

`trn_all_set`  
Use all sets as training dataset. Otherwise, if the number of sets is more than 1, the last set is left for test.

`sort_atoms`  
[`bool`] Sort atoms by atom types. Required to enable when the data is directly feeded to descriptors except mixed types.

## Methods

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded.
<code>avg(key)</code>	Return the average value of an item.
<code>check_batch_size(batch_size)</code>	Check if the system can get a batch of data with <code>batch_size</code> frames.
<code>check_test_size(test_size)</code>	Check if the system can get a test dataset with <code>test_size</code> frames.
<code>get_atom_type()</code>	Get atom types.
<code>get_batch(batch_size)</code>	Get a batch of data with <code>batch_size</code> frames.
<code>get_data_dict()</code>	Get the <code>data_dict</code> .
<code>get_natoms()</code>	Get number of atoms.
<code>get_natoms_vec(ntypes)</code>	Get number of atoms and number of atoms in different types.
<code>get_ntypes()</code>	Number of atom types in the system.
<code>get_num_batch(batch_size, set_idx)</code>	Get the number of batches in a set.
<code>get_num_set()</code>	Get number of training sets.
<code>get_sys_num_batch(batch_size)</code>	Get the number of batches in the data system.
<code>get_test([ntests])</code>	Get the test data with <code>ntests</code> frames.
<code>get_type_map()</code>	Get the type map.
<code>reduce(key_out, key_in)</code>	Generate a new item from the reduction of another atom.

reset_get_batch	
-----------------	--

**add**(key: str, ndof: int, atomic: bool = False, must: bool = False, high\_prec: bool = False, type\_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0, dtype: Optional[dtype] = None)  
 Add a data item that to be loaded.

## Parameters

**key**  
 The key of the item. The corresponding data is stored in `sys_path/set.*/key.npy`

**ndof**  
 The number of dof

**atomic**  
 The item is an atomic property. If False, the size of the data should be `nframes x ndof` If True, the size of data should be `nframes x natoms x ndof`

**must**  
 The data file `sys_path/set.*/key.npy` must exist. If `must` is False and the data file does not exist, the `data_dict[find_key]` is set to 0.0

**high\_prec**  
 Load the data and store in float64, otherwise in float32

**type\_sel**  
 Select certain type of atoms

**repeat**  
 The data will be repeated `repeat` times.

**default**  
 [float, default=0.] default value of data

`dtype`  
`[np.dtype, optional]` the dtype of data, overwrites `high_prec` if provided

**avg(key)**  
 Return the average value of an item.

**check\_batch\_size(batch\_size)**  
 Check if the system can get a batch of data with `batch_size` frames.

**check\_test\_size(test\_size)**  
 Check if the system can get a test dataset with `test_size` frames.

**get\_atom\_type()** → `List[int]`  
 Get atom types.

**get\_batch(batch\_size: int)** → `dict`  
 Get a batch of data with `batch_size` frames. The frames are randomly picked from the data system.

Parameters

`batch_size`  
 size of the batch

**get\_data\_dict()** → `dict`  
 Get the `data_dict`.

**get\_natoms()**  
 Get number of atoms.

**get\_natoms\_vec(ntypes: int)**  
 Get number of atoms and number of atoms in different types.

Parameters

`ntypes`  
 Number of types (may be larger than the actual number of types in the system).

Returns

**natoms**  
`natoms[0]`: number of local atoms `natoms[1]`: total number of atoms held by this processor `natoms[i]`:  $2 \leq i < Ntypes+2$ , number of type `i` atoms

**get\_ntypes()** → `int`  
 Number of atom types in the system.

**get\_numb\_batch(batch\_size: int, set\_idx: int)** → `int`  
 Get the number of batches in a set.

**get\_numb\_set()** → `int`  
 Get number of training sets.

**get\_sys\_numb\_batch(batch\_size: int)** → `int`  
 Get the number of batches in the data system.

**get\_test(ntests: int = -1)** → `dict`  
 Get the test data with `ntests` frames.

Parameters

`ntests`  
 Size of the test data set. If `ntests` is -1, all test data will be get.

`get_type_map()` → `List[str]`

Get the type map.

`reduce(key_out: str, key_in: str)`

Generate a new item from the reduction of another atom.

Parameters

`key_out`

The name of the reduced item

`key_in`

The name of the data item to be reduced

`reset_get_batch()`

### `deepmd_utils.utils.data_system` module

```
class deepmd_utils.utils.data_system.DeepmdDataSystem(systems: List[str], batch_size: int,
    test_size: int, rcut: Optional[float] = None,
    set_prefix: str = 'set', shuffle_test: bool =
    True, type_map: Optional[List[str]] =
    None, optional_type_map: bool = True,
    modifier=None, trn_all_set=False,
    sys_probs=None,
    auto_prob_style='prob_sys_size',
    sort_atoms: bool = True)
```

Bases: `object`

Class for manipulating many data systems.

It is implemented with the help of `DeepmdData`

Attributes

`default_mesh`

Mesh for each system.



## Methods

<code>add(key, ndof[, atomic, must, high_prec, ...])</code>	Add a data item that to be loaded.
<code>add_dict(adict)</code>	Add items to the data system by a dict.
<code>get_batch([sys_idx])</code>	Get a batch of data from the data systems.
<code>get_batch_mixed()</code>	Get a batch of data from the data systems in the mixed way.
<code>get_batch_size()</code>	Get the batch size.
<code>get_batch_standard([sys_idx])</code>	Get a batch of data from the data systems in the standard way.
<code>get_nbatches()</code>	Get the total number of batches.
<code>get_nsystems()</code>	Get the number of data systems.
<code>get_ntypes()</code>	Get the number of types.
<code>get_sys(idx)</code>	Get a certain data system.
<code>get_sys_ntest([sys_idx])</code>	Get number of tests for the currently selected system, or one defined by <code>sys_idx</code> .
<code>get_test([sys_idx, n_test])</code>	Get test data from the the data systems.
<code>get_type_map()</code>	Get the type map.
<code>reduce(key_out, key_in)</code>	Generate a new item from the reduction of another atom.

<code>compute_energy_shift</code>	
<code>get_data_dict</code>	
<code>print_summary</code>	
<code>set_sys_probs</code>	

**add**(key: str, ndof: int, atomic: bool = False, must: bool = False, high\_prec: bool = False, type\_sel: Optional[List[int]] = None, repeat: int = 1, default: float = 0.0)

Add a data item that to be loaded.

Parameters

key

The key of the item. The corresponding data is stored in `sys_path/set.*/key.npy`

ndof

The number of dof

atomic

The item is an atomic property. If False, the size of the data should be `nframes x ndof` If True, the size of data should be `nframes x natoms x ndof`

must

The data file `sys_path/set.*/key.npy` must exist. If must is False and the data file does not exist, the `data_dict[find_key]` is set to 0.0

high\_prec

Load the data and store in float64, otherwise in float32

type\_sel

Select certain type of atoms

repeat

The data will be repeated repeat times.

default, default=0.  
Default value of data

**add\_dict**(adict: dict) → None

Add items to the data system by a dict. adict should have items like .. code-block:: python.

```
adict[key] = {
    "ndof": ndof, "atomic": atomic, "must": must, "high_prec": high_prec, "type_sel":
    type_sel, "repeat": repeat,
}
```

For the explanation of the keys see add

**compute\_energy\_shift**(rcond=None, key='energy')

**property default\_mesh:** List[ndarray]

Mesh for each system.

**get\_batch**(sys\_idx: Optional[int] = None) → dict

Get a batch of data from the data systems.

Parameters

sys\_idx  
[int] The index of system from which the batch is get. If sys\_idx is not None, sys\_probs and auto\_prob\_style are ignored If sys\_idx is None, automatically determine the system according to sys\_probs or auto\_prob\_style, see the following. This option does not work for mixed systems.

Returns

dict  
The batch data

**get\_batch\_mixed**() → dict

Get a batch of data from the data systems in the mixed way.

Returns

dict  
The batch data

**get\_batch\_size**() → int

Get the batch size.

**get\_batch\_standard**(sys\_idx: Optional[int] = None) → dict

Get a batch of data from the data systems in the standard way.

Parameters

sys\_idx  
[int] The index of system from which the batch is get. If sys\_idx is not None, sys\_probs and auto\_prob\_style are ignored If sys\_idx is None, automatically determine the system according to sys\_probs or auto\_prob\_style, see the following.

Returns

dict  
The batch data

**get\_data\_dict**(ii: int = 0) → dict

`get_nbatches()` → `int`

Get the total number of batches.

`get_nsystems()` → `int`

Get the number of data systems.

`get_ntypes()` → `int`

Get the number of types.

`get_sys(idx: int)` → `DeepmdData`

Get a certain data system.

`get_sys_ntest(sys_idx=None)`

Get number of tests for the currently selected system, or one defined by `sys_idx`.

`get_test(sys_idx: Optional[int] = None, n_test: int = -1)`

Get test data from the the data systems.

Parameters

`sys_idx`

The test dat of system with index `sys_idx` will be returned. If is `None`, the currently selected system will be returned.

`n_test`

Number of test data. If set to -1 all test data will be get.

`get_type_map()` → `List[str]`

Get the type map.

`print_summary(name)`

`reduce(key_out, key_in)`

Generate a new item from the reduction of another atom.

Parameters

`key_out`

The name of the reduced item

`key_in`

The name of the data item to be reduced

`set_sys_probs(sys_probs=None, auto_prob_style: str = 'prob_sys_size')`

`deepmd_utils.utils.data_system.prob_sys_size_ext(keywords, nsystems, nbatch)`

`deepmd_utils.utils.data_system.process_sys_probs(sys_probs, nbatch)`

### **deepmd\_utils.utils.errors module**

**exception** `deepmd_utils.utils.errors.OutOfMemoryError`

Bases: `Exception`

This error is caused by out-of-memory (OOM).

**deepmd\_utils.utils.model\_stat module**

`deepmd_utils.utils.model_stat.make_stat_input(data, nbatches, merge_sys=True)`

Pack data for statistics.

Parameters

`data`

The data

`nbatches`

[`int`] The number of batches

`merge_sys`

[`bool` (`True`)] Merge system data

Returns

`all_stat`:

A dictionary of list of list storing data for stat. if `merge_sys == False` data can be accessed by

`all_stat[key][sys_idx][batch_idx][frame_idx]`

else `merge_sys == True` can be accessed by

`all_stat[key][batch_idx][frame_idx]`

`deepmd_utils.utils.model_stat.merge_sys_stat(all_stat)`

**deepmd\_utils.utils.pair\_tab module**

`class deepmd_utils.utils.pair_tab.PairTab(filename: str)`

Bases: `object`

Pairwise tabulated potential.

Parameters

`filename`

File name for the short-range tabulated potential. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  colums. The first colum is the distance between atoms. The second to the last colums are energies for pairs of certain types. For example we have two atom types, 0 and 1. The colums from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

**Methods**

<code>get()</code>	Get the serialized table.
<code>reinit(filename)</code>	Initialize the tabulated interaction.

`get()` → `Tuple`[`array`, `array`]

Get the serialized table.

`reinit(filename: str) → None`

Initialize the tabulated interaction.

Parameters

filename

File name for the short-range tabulated potential. The table is a text data file with  $(N_t + 1) * N_t / 2 + 1$  columns. The first column is the distance between atoms. The second to the last columns are energies for pairs of certain types. For example we have two atom types, 0 and 1. The columns from 2nd to 4th are for 0-0, 0-1 and 1-1 correspondingly.

## deepmd\_utils.utils.path module

`class deepmd_utils.utils.path.DPH5Path(path: str)`

Bases: *DPPath*

The path class to data system (DeepmdData) for HDF5 files.

Parameters

path

[str] path

## Notes

OS - HDF5 relationship:

directory - Group file - Dataset

## Methods

<i>glob</i> (pattern)	Search path using the glob pattern.
<i>is_dir</i> ()	Check if self is directory.
<i>is_file</i> ()	Check if self is file.
<i>load_numpy</i> ()	Load NumPy array.
<i>load_txt</i> ([dtype])	Load NumPy array from text.
<i>rglob</i> (pattern)	This is like calling <i>DPPath.glob()</i> with <code>**/</code> added in front of the given relative pattern.

`glob(pattern: str) → List[DPPath]`

Search path using the glob pattern.

Parameters

pattern

[str] glob pattern

Returns

List[DPPath]

list of paths

`is_dir() → bool`

Check if self is directory.

`is_file()` → `bool`

Check if self is file.

`load_numpy()` → `ndarray`

Load NumPy array.

Returns

`np.ndarray`

loaded NumPy array

`load_txt(dtype: Optional[dtype] = None, **kwargs)` → `ndarray`

Load NumPy array from text.

Returns

`np.ndarray`

loaded NumPy array

`rglob(pattern: str)` → `List[DPPPath]`

This is like calling `DPPPath.glob()` with `**/` added in front of the given relative pattern.

Parameters

pattern

`[str]` glob pattern

Returns

`List[DPPPath]`

list of paths

`class deepmd_utils.utils.path.DPOSPath(path: str)`

Bases: `DPPPath`

The OS path class to data system (DeepmdData) for real directories.

Parameters

path

`[str]` path

## Methods

<code>glob(pattern)</code>	Search path using the glob pattern.
<code>is_dir()</code>	Check if self is directory.
<code>is_file()</code>	Check if self is file.
<code>load_numpy()</code>	Load NumPy array.
<code>load_txt(**kwargs)</code>	Load NumPy array from text.
<code>rglob(pattern)</code>	This is like calling <code>DPPPath.glob()</code> with <code>**/</code> added in front of the given relative pattern.

`glob(pattern: str)` → `List[DPPPath]`

Search path using the glob pattern.

Parameters

pattern

`[str]` glob pattern

Returns

`List[DPPath]`  
list of paths

`is_dir()` → `bool`

Check if self is directory.

`is_file()` → `bool`

Check if self is file.

`load_numpy()` → `ndarray`

Load NumPy array.

Returns

`np.ndarray`  
loaded NumPy array

`load_txt(**kwargs)` → `ndarray`

Load NumPy array from text.

Returns

`np.ndarray`  
loaded NumPy array

`rglob(pattern: str)` → `List[DPPath]`

This is like calling `DPPath.glob()` with `**/` added in front of the given relative pattern.

Parameters

pattern  
[`str`] glob pattern

Returns

`List[DPPath]`  
list of paths

`class deepmd_utils.utils.path.DPPath(path: str)`

Bases: `ABC`

The path class to data system (DeepmdData).

Parameters

path  
[`str`] path

## Methods

<code>glob(pattern)</code>	Search path using the glob pattern.
<code>is_dir()</code>	Check if self is directory.
<code>is_file()</code>	Check if self is file.
<code>load_numpy()</code>	Load NumPy array.
<code>load_txt(**kwargs)</code>	Load NumPy array from text.
<code>rglob(pattern)</code>	This is like calling <code>DPPath.glob()</code> with <code>**/</code> added in front of the given relative pattern.

**abstract** `glob(pattern: str) → List[DPPPath]`

Search path using the glob pattern.

Parameters

pattern  
[`str`] glob pattern

Returns

`List[DPPPath]`  
list of paths

**abstract** `is_dir() → bool`

Check if self is directory.

**abstract** `is_file() → bool`

Check if self is file.

**abstract** `load_numpy() → ndarray`

Load NumPy array.

Returns

`np.ndarray`  
loaded NumPy array

**abstract** `load_txt(**kwargs) → ndarray`

Load NumPy array from text.

Returns

`np.ndarray`  
loaded NumPy array

**abstract** `rglob(pattern: str) → List[DPPPath]`

This is like calling `DPPPath.glob()` with `**/` added in front of the given relative pattern.

Parameters

pattern  
[`str`] glob pattern

Returns

`List[DPPPath]`  
list of paths

### **deepmd\_utils.utils.plugin module**

Base of plugin systems.

**class** `deepmd_utils.utils.plugin.Plugin`

Bases: `object`

A class to register and restore plugins.



## Examples

```
>>> plugin = Plugin()
>>> @plugin.register("xx")
    def xxx():
        pass
>>> print(plugin.plugins['xx'])
```

### Attributes

plugins  
[Dict[str, object]] plugins

## Methods

<code>get_plugin(key)</code>	Visit a plugin by key.
<code>register(key)</code>	Register a plugin.

`get_plugin(key) → object`

Visit a plugin by key.

### Parameters

key  
[str] key of the plugin

### Returns

object  
the plugin

`register(key: str) → Callable[[object], object]`

Register a plugin.

### Parameters

key  
[str] key of the plugin

### Returns

Callable[[object], object]  
decorator

`class deepmd_utils.utils.plugin.PluginVariant(*args, **kwargs)`

Bases: object

A class to remove type from input arguments.

`class deepmd_utils.utils.plugin.VariantABCMeta(name, bases, namespace, **kwargs)`

Bases: VariantMeta, ABCMeta

## Methods

<code>__call__(*args, **kwargs)</code>	Remove type and keys that starts with underline.
<code>mro()</code>	Return a type's method resolution order.
<code>register(subclass)</code>	Register a virtual subclass of an ABC.

`class deepmd_utils.utils.plugin.VariantMeta`

Bases: `object`

## Methods

<code>__call__(*args, **kwargs)</code>	Remove type and keys that starts with underline.
--	--

## deepmd\_utils.utils.random module

`deepmd_utils.utils.random.choice(a: ndarray, p: Optional[ndarray] = None)`

Generates a random sample from a given 1-D array.

Parameters

- a  
[`np.ndarray`] A random sample is generated from its elements.
- p  
[`np.ndarray`] The probabilities associated with each entry in a.

Returns

`np.ndarray`  
arrays with results and their shapes

`deepmd_utils.utils.random.random(size=None)`

Return random floats in the half-open interval [0.0, 1.0).

Parameters

- size  
Output shape.

Returns

`np.ndarray`  
Arrays with results and their shapes.

`deepmd_utils.utils.random.seed(val: Optional[int] = None)`

Seed the generator.

Parameters

- val  
[`int`] Seed.

```
deepmd_utils.utils.random.shuffle(x: ndarray)
```

Modify a sequence in-place by shuffling its contents.

Parameters

`x`  
`[np.ndarray]` The array or list to be shuffled.

### deepmd\_utils.utils.weight\_avg module

```
deepmd_utils.utils.weight_avg.weighted_average(errors: List[Dict[str, Tuple[float, float]]]) → Dict
```

Compute wighted average of prediction errors (MAE or RMSE) for model.

Parameters

`errors`  
`[List[Dict[str, Tuple[float, float]]]]` List: the error of systems Dict: the error of quantities, name given by the key str: the name of the quantity, must starts with 'mae' or 'rmse' Tuple: (error, weight)

Returns

`Dict`  
 weighted averages

## 18.3.2 Submodules

### 18.3.3 deepmd\_utils.common module

```
deepmd_utils.common.add_data_requirement(key: str, ndof: int, atomic: bool = False, must: bool = False, high_prec: bool = False, type_sel: Optional[bool] = None, repeat: int = 1, default: float = 0.0, dtype: Optional[dtype] = None)
```

Specify data requirements for training.

Parameters

`key`  
`[str]` type of data stored in corresponding \*.npv file e.g. forces or energy

`ndof`  
`[int]` number of the degrees of freedom, this is tied to atomic parameter e.g. forces have atomic=True and ndof=3

`atomic`  
`[bool, optional]` specifies whwther the ndof keyworrd applies to per atom quantity or not, by default False

`must`  
`[bool, optional]` specifi if the \*.npv data file must exist, by default False

`high_prec`  
`[bool, optional]` if true load data to np.float64 else np.float32, by default False

`type_sel`  
`[bool, optional]` select only certain type of atoms, by default None

`repeat`  
    [`int`, optional] if specify repaeat data repeat times, by default 1  
`default`  
    [`float`, optional, default=0.] default value of data  
`dtype`  
    [`np.dtype`, optional] the dtype of data, overwrites `high_prec` if provided

`deepmd_utils.common.expand_sys_str`(`root_dir`: `Union[str, Path]`) → `List[str]`

Recursively iterate over directories taking those that contain `type.raw` file.

Parameters

`root_dir`  
    [`Union[str, Path]`] starting directory

Returns

`List[str]`  
    list of string pointing to system directories

`deepmd_utils.common.get_np_precision`(`precision`: `_PRECISION`) → `dtype`

Get numpy precision constant from string.

Parameters

`precision`  
    [`_PRECISION`] string name of numpy constant or default

Returns

`np.dtype`  
    numpy presicion constant

Raises

`RuntimeError`  
    if string is invalid

`deepmd_utils.common.j_loader`(`filename`: `Union[str, Path]`) → `Dict[str, Any]`

Load yaml or json settings file.

Parameters

`filename`  
    [`Union[str, Path]`] path to file

Returns

`Dict[str, Any]`  
    loaded dictionary

Raises

`TypeError`  
    if the supplied file is of unsupported type

`deepmd_utils.common.j_must_have`(`jdata`: `Dict[str, _DICT_VAL]`, `key`: `str`, `deprecated_key`: `List[str] = []`) → `_DICT_VAL`

Assert that supplied dictionary conaines specified key.

Returns

`_DICT_VAL`  
value that was store unde supplied key

Raises

`RuntimeError`  
if the key is not present

`deepmd_utils.common.make_default_mesh(pbc: bool, mixed_type: bool) → ndarray`

Make mesh.

Only the size of mesh matters, not the values: \* 6 for PBC, no mixed types \* 0 for no PBC, no mixed types \* 7 for PBC, mixed types \* 1 for no PBC, mixed types

Parameters

`pbc`  
[bool] if True, the mesh will be made for periodic boundary conditions

`mixed_type`  
[bool] if True, the mesh will be made for mixed types

Returns

`np.ndarray`  
mesh

`deepmd_utils.common.select_idx_map(atom_types: ndarray, select_types: ndarray) → ndarray`

Build map of indices for element supplied element types from all atoms list.

Parameters

`atom_types`  
[`np.ndarray`] array specifying type for each atoms as integer

`select_types`  
[`np.ndarray`] types of atoms you want to find indices for

Returns

`np.ndarray`  
indices of types of atoms defined by select\_types in atom\_types array

Warning: select\_types array will be sorted before finding indices in atom\_types

### 18.3.4 deepmd\_utils.env module

`deepmd_utils.env.GLOBAL_ENER_FLOAT_PRECISION`  
alias of float64

`deepmd_utils.env.GLOBAL_NP_FLOAT_PRECISION`  
alias of float64

### 18.3.5 deepmd\_utils.main module

The entry points for DeePMD-kit.

If only printing the help message, this module does not call the main DeePMD-kit module to avoid the slow import of TensorFlow.

```
class deepmd_utils.main.RawTextArgumentDefaultsHelpFormatter(prog, indent_increment=2,
                                                             max_help_position=24,
                                                             width=None)
```

Bases: `RawTextHelpFormatter`, `ArgumentDefaultsHelpFormatter`

This formatter is used to print multile-line help message with default value.

#### Methods

---

`format_help()`

---

`start_section(heading)`

---

<code>add_argument</code>	
<code>add_arguments</code>	
<code>add_text</code>	
<code>add_usage</code>	
<code>end_section</code>	

`deepmd_utils.main.get_ll(log_level: str) → int`

Convert string to python logging level.

Parameters

`log_level`

`[str]` allowed input values are: DEBUG, INFO, WARNING, ERROR, 3, 2, 1, 0

Returns

`int`

one of python logging module log levels - 10, 20, 30 or 40

`deepmd_utils.main.main()`

DeePMD-kit new entry point.

Raises

`RuntimeError`

if no command was input

`deepmd_utils.main.main_parser() → ArgumentParser`

DeePMD-Kit commandline options argument parser.

Returns

`argparse.ArgumentParser`

main parser of DeePMD-kit

`deepmd_utils.main.parse_args(args: Optional[List[str]] = None) → Namespace`

Parse arguments and convert argument strings to objects.

Parameters

`args`

`[List[str]]` list of command line arguments, main purpose is testing default option  
None takes arguments from `sys.argv`

Returns

`argparse.Namespace`

the populated namespace





## 19.1 op\_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

```
deepmd.env.op_module.AddFltNvnmd(x, w, name=None)
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

```
class deepmd.env.op_module.Annotated(*args, **kwargs)
```

Bases: `object`

Add context specific metadata to a type.

Example: `Annotated[int, runtime_check.Unsigned]` indicates to the hypothetical `runtime_check` module that this type is an unsigned int. Every other consumer of this type can ignore this metadata and treat this type as int.

The first argument to `Annotated` must be a valid type.

Details:

- It's an error to call `Annotated` with less than two arguments.
- Nested `Annotated` are flattened:

```
Annotated[Annotated[T, Ann1, Ann2], Ann3] == Annotated[T, Ann1, Ann2, Ann3]
```

- Instantiating an annotated type is equivalent to instantiating the underlying type:

```
Annotated[C, Ann1](5) == C(5)
```

- `Annotated` can be used as a generic type alias:

```

Optimized = Annotated[T, runtime.Optimize()]
Optimized[int] == Annotated[int, runtime.Optimize()]

OptimizedList = Annotated[List[T], runtime.Optimize()]
OptimizedList[int] == Annotated[List[int], runtime.Optimize()]

```

`deepmd.env.op_module.ConvertForwardMap(sub_forward_map, sub_natoms, natoms, name=None)`

TODO: add doc.

Parameters

- **sub\_forward\_map** – A Tensor of type int32.
- **sub\_natoms** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward\_map, backward\_map, new\_natoms, mesh).

forward\_map: A Tensor of type int32. backward\_map: A Tensor of type int32.

new\_natoms: A Tensor of type int32. mesh: A Tensor of type int32.

`deepmd.env.op_module.CopyFltNvnmd(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (y1, y2).

y1: A Tensor. Has the same type as x. y2: A Tensor. Has the same type as x.

`deepmd.env.op_module.Descript(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, sel_a, sel_r, axis_rule, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **sel\_a** – A list of ints.

- **sel\_r** – A list of ints.
- **axis\_rule** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist, axis, rot\_mat).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. axis: A Tensor of type int32. rot\_mat: A Tensor. Has the same type as coord.

```
deepmd.env.op_module.DescriptNorot(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,
                                   rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

#### Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.DescriptSeA(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,
                                  rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

#### Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.

- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt\_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.DescriptSeAEf(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r,
                                     rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt\_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.DescriptSeAEfPara(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r,
                                         rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DescriptSeAEfVert(coord, type, natoms, box, mesh, ef, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DescriptSeAMask(coord, type, mask, box, natoms, mesh, name=None)`

TODO: add doc.

## Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **mask** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **mesh** – A Tensor of type int32.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DescriptSeR(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel, name=None)`

TODO: add doc.

## Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut\_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.DotmulFltNvnmd(x, w, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.DprcPairwiseIdx(idxs, natoms, name=None)`

TODO: add doc.

Parameters

- **idxs** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward\_qm\_map, backward\_qm\_map, forward\_qmmm\_map, backward\_qmmm\_map, natoms\_qm, natoms\_qmmm, qmmm\_frame\_idx).

forward\_qm\_map: A Tensor of type int32. backward\_qm\_map: A Tensor of type int32. forward\_qmmm\_map: A Tensor of type int32. backward\_qmmm\_map: A Tensor of type int32. natoms\_qm: A Tensor of type int32. natoms\_qmmm: A Tensor of type int32. qmmm\_frame\_idx: A Tensor of type int32.

`deepmd.env.op_module.EwaldRecp(coord, charge, natoms, box, ewald_beta, ewald_h, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **charge** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **ewald\_beta** – A float.
- **ewald\_h** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

`deepmd.env.op_module.FltNvnmd(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.Gelu(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.GeluCustom(x, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.GeluGrad(dy, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

`deepmd.env.op_module.GeluGradCustom(dy, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

`deepmd.env.op_module.GeluGradGrad(dy, dy_, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy\_** – A Tensor. Must have the same type as dy.



- **x** – A Tensor. Must have the same type as **dy**.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **dy**.

`deepmd.env.op_module.GeluGradGradCustom(dy, dy_, x, name=None)`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy\_** – A Tensor. Must have the same type as **dy**.
- **x** – A Tensor. Must have the same type as **dy**.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **dy**.

`deepmd.env.op_module.MapAparam(aparam, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **aparam** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **aparam**.

`deepmd.env.op_module.MapFltNvnmd(x, table, table_grad, table_info, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **table** – A Tensor. Must have the same type as **x**.
- **table\_grad** – A Tensor. Must have the same type as **x**.
- **table\_info** – A Tensor. Must have the same type as **x**.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **x**.

`deepmd.env.op_module.MatmulFitnetNvnmd(x, w, nbitx, nbitw, normw, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.

- **w** – A Tensor. Must have the same type as **x**.
- **nbitx** – An int.
- **nbitw** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **x**.

`deepmd.env.op_module.MatmulFlt2fixNvnmd(x, w, nbit, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as **x**.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **x**.

`deepmd.env.op_module.MatmulFltNvnmd(x, w, normx, normw, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as **x**.
- **normx** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **x**.

`deepmd.env.op_module.MulFltNvnmd(x, w, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as **x**.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as **x**.

`deepmd.env.op_module.NeighborStat(coord, type, natoms, box, mesh, rcut, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.

- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **rcut** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max\_nbor\_size, min\_nbor\_dist).

max\_nbor\_size: A Tensor of type int32. min\_nbor\_dist: A Tensor. Has the same type as coord.

```
deepmd.env.op_module.PairTab(table_info, table_data, type, rij, nlist, natoms, scale, sel_a, sel_r,
                             name=None)
```

TODO: add doc.

Parameters

- **table\_info** – A Tensor of type float64.
- **table\_data** – A Tensor of type float64.
- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **scale** – A Tensor. Must have the same type as rij.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom\_energy, force, atom\_virial).

atom\_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom\_virial: A Tensor. Has the same type as rij.

```
deepmd.env.op_module.ParallelProdForceSeA(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel,
                                           parallel=False, start_frac=0, end_frac=1, name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.

- **parallel** – An optional bool. Defaults to False.
- **start\_frac** – An optional float. Defaults to 0.
- **end\_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as `net_deriv`.

`deepmd.env.op_module.ProdEnvMatA(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r, name=None)`

Compute the environment matrix for descriptor `se_e2_a`.

Each row of the environment matrix  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{r_{ji}} \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{bmatrix}$$

In the above equation,  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Note that the environment matrix is normalized by `davg` and `dstd`.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64. The coordinates of atoms.
- **type** – A Tensor of type int32. The types of atoms.
- **natoms** – A Tensor of type int32. The number of atoms. This tensor has the length of `Ntypes + 2`. `natoms[0]`: number of local atoms. `natoms[1]`: total number of atoms held by this processor. `natoms[i]`:  $2 \leq i < \text{Ntypes}+2$ , number of type *i* atoms.
- **box** – A Tensor. Must have the same type as `coord`. The box of frames.
- **mesh** – A Tensor of type int32. For historical reasons, only the length of the Tensor matters. If size of `mesh` == 6, pbc is assumed. If size of `mesh` == 0, no-pbc is assumed.
- **davg** – A Tensor. Must have the same type as `coord`. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as `coord`. Standard deviation of the environment matrix for normalization.
- **rcut\_a** – A float. This argument is not used.
- **rcut\_r** – A float. The cutoff radius for the environment matrix.

- **rcut\_r\_smth** – A float. From where the environment matrix should be smoothed.
- **sel\_a** – A list of ints. `sel_a[i]` specifies the maximum number of type `i` atoms in the cut-off radius.
- **sel\_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (`descrpt`, `descrpt_deriv`, `rij`, `nlist`).

`descrpt`: A Tensor. Has the same type as `coord`. The environment matrix. `descrpt_deriv`: A Tensor. Has the same type as `coord`. The derivative of the environment matrix. `rij`: A Tensor. Has the same type as `coord`. The distance between the atoms. `nlist`: A Tensor of type `int32`. The neighbor list of each atom.

```
deepmd.env.op_module.ProdEnvMatAMix(coord, type, natoms, box, mesh, davg, dstd, rcut_a, rcut_r,
                                     rcut_r_smth, sel_a, sel_r, name=None)
```

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in `nlist` matrix will gather forward and thus save space for gaps of types in `ProdEnvMatA`, resulting in optimized and relative small `sel_a`.

The additional outputs are listed as following:

#### Parameters

- **coord** – A Tensor. Must be one of the following types: `float32`, `float64`.
- **type** – A Tensor of type `int32`.
- **natoms** – A Tensor of type `int32`.
- **box** – A Tensor. Must have the same type as `coord`.
- **mesh** – A Tensor of type `int32`.
- **davg** – A Tensor. Must have the same type as `coord`.
- **dstd** – A Tensor. Must have the same type as `coord`.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (`descrpt`, `descrpt_deriv`, `rij`, `nlist`, `ntype`, `nmask`).

`descrpt`: A Tensor. Has the same type as `coord`. `descrpt_deriv`: A Tensor. Has the same type as `coord`. `rij`: A Tensor. Has the same type as `coord`. `nlist`: A Tensor of type `int32`. `ntype`: A Tensor of type `int32`. The corresponding atom types in `nlist`. `nmask`: A Tensor of type `bool`. The atom mask in `nlist`.

```
deepmd.env.op_module.ProdEnvMatAMixNvnmdQuantize(coord, type, natoms, box, mesh, davg, dstd,
                                                    rcut_a, rcut_r, rcut_r_smth, sel_a, sel_r,
                                                    name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist, ntype, nmask).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. nmask: A Tensor of type bool.

```
deepmd.env.op_module.ProdEnvMatANvnmdQuantize(coord, type, natoms, box, mesh, davg, dstd, rcut_a,
                                                rcut_r, rcut_r_smth, sel_a, sel_r, name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.

- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.ProdEnvMatR(coord, type, natoms, box, mesh, davg, dstd, rcut, rcut_smth, sel,
                                  name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut\_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.ProdForce(net_deriv, in_deriv, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.ProdForceNorot(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel,
                                     name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.ProdForceSeA(net_deriv, in_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.ProdForceSeAMask(net_deriv, in_deriv, mask, nlist, total_atom_num,
                                       name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total\_atom\_num** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.



`deepmd.env.op_module.ProdForceSeR(net_deriv, in_deriv, nlist, natoms, name=None)`

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

`deepmd.env.op_module.ProdVirial(net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

`deepmd.env.op_module.ProdVirialNorot(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.

- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.ProdVirialSeA(net_deriv, in_deriv, rij, nlist, natoms, n_a_sel, n_r_sel,
                                   name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.ProdVirialSeR(net_deriv, in_deriv, rij, nlist, natoms, name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.QuantizeNvnmd(x, isround, nbit1, nbit2, nbit3, name=None)
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.

- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.SoftMinForce(du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw\_deriv** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as du.

`deepmd.env.op_module.SoftMinSwitch(type, rij, nlist, natoms, sel_a, sel_r, alpha, rmin, rmax, name=None)`

TODO: add doc.

Parameters

- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **alpha** – A float.
- **rmin** – A float.
- **rmax** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (sw\_value, sw\_deriv).

sw\_value: A Tensor. Has the same type as rij. sw\_deriv: A Tensor. Has the same type as rij.

`deepmd.env.op_module.SoftMinVirial(du, sw_deriv, rij, nlist, natoms, n_a_sel, n_r_sel, name=None)`

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw\_deriv** – A Tensor. Must have the same type as du.
- **rij** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as du. atom\_virial: A Tensor. Has the same type as du.

`deepmd.env.op_module.TabulateFusion(table, table_info, em_x, em, last_layer_size, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionGrad(table, table_info, em_x, em, dy, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionGradGrad(table, table_info, em_x, em, dz_dy_dem_x,
                                             dz_dy_dem, descriptor, name=None)
```

TODO: add doc.

## Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

## Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeA(table, table_info, em_x, em, last_layer_size, name=None)
```

TODO: add doc.

## Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **name** – A name for the operation (optional).

## Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeAGrad(table, table_info, em_x, em, dy, descriptor,
                                             name=None)
```

TODO: add doc.

## Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.

- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeAGradGrad(table, table_info, em_x, em, dz_dy_dem_x,
                                                dz_dy_dem, descriptor, is_sorted=True,
                                                name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is\_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeAtten(table, table_info, em_x, em, two_embed,
                                            last_layer_size, is_sorted=True, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two\_embed** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **is\_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeAttenGrad(table, table_info, em_x, em, two_embed, dy,
                                                descriptor, is_sorted=True, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two\_embed** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is\_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem, dy\_dtwo).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table. dy\_dtwo: A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeAttenGradGrad(table, table_info, em_x, em, two_embed,
                                                    dz_dy_dem_x, dz_dy_dem, dz_dy_dtwo,
                                                    descriptor, is_sorted=True, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two\_embed** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **dz\_dy\_dtwo** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is\_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeR(table, table_info, em, last_layer_size, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeRGrad(table, table_info, em, dy, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeRGradGrad(table, table_info, em, dz_dy_dem, descriptor, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.TabulateFusionSeT(table, table_info, em_x, em, last_layer_size, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.



- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeTGrad(table, table_info, em_x, em, dy, descriptor,
                                             name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.TabulateFusionSeTGradGrad(table, table_info, em_x, em, dz_dy_dem_x,
                                                dz_dy_dem, descriptor, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.Tanh4FltNvnmd(x, name=None)
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.UnaggregatedDy2Dx(z, w, dy_dx, dy2_dx, ybar, functype, name=None)`

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy\_dx** – A Tensor. Must have the same type as z.
- **dy2\_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as z.

`deepmd.env.op_module.UnaggregatedDy2DxS(y, dy, w, xbar, functype, name=None)`

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as y.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as y.

`deepmd.env.op_module.UnaggregatedDyDx(z, w, dy_dx, ybar, functype, name=None)`

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy\_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as z.

`deepmd.env.op_module.UnaggregatedDyDxS(y, w, xbar, functype, name=None)`

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.

- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as y.

`deepmd.env.op_module.add_flt_nvnmmd(x: Any, w: Any, name=None) → Any`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.convert_forward_map(sub_forward_map: Any, sub_natoms: Any, natoms: Any, name=None)`

TODO: add doc.

Parameters

- **sub\_forward\_map** – A Tensor of type int32.
- **sub\_natoms** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward\_map, backward\_map, new\_natoms, mesh).

forward\_map: A Tensor of type int32. backward\_map: A Tensor of type int32.  
new\_natoms: A Tensor of type int32. mesh: A Tensor of type int32.

`deepmd.env.op_module.copy_flt_nvnmmd(x: Any, name=None)`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (y1, y2).

y1: A Tensor. Has the same type as x. y2: A Tensor. Has the same type as x.

`deepmd.env.op_module.descript(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, sel_a, sel_r, axis_rule, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **axis\_rule** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist, axis, rot\_mat).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same

type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

axis: A Tensor of type int32. rot\_mat: A Tensor. Has the same type as coord.

```
deepmd.env.op_module.descript_norot(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any,  
                                     davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth:  
                                     float, sel_a, sel_r, name=None)
```

TODO: add doc.

#### Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)
```

TODO: add doc.

## Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a_ef(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, ef: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)
```

TODO: add doc.

## Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.

- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a_ef_para(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, ef: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)
```

TODO: add doc.

#### Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.descript_se_a_ef_vert(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, ef: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)
```

TODO: add doc.

## Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **ef** – A Tensor. Must have the same type as coord.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (descrpt, descrpt\_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.descrpt_se_a_mask(coord: Any, type: Any, mask: Any, box: Any, natoms: Any, mesh: Any, name=None)`

TODO: add doc.

## Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **mask** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **mesh** – A Tensor of type int32.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (descrpt, descrpt\_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.descrpt_se_r(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut: float, rcut_smth: float, sel, name=None)`

TODO: add doc.

## Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut\_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

`deepmd.env.op_module.dotmul_flt_nvnmmd(x: Any, w: Any, name=None) → Any`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.dprc_pairwise_idx(idxs: Any, natoms: Any, name=None)`

TODO: add doc.

Parameters

- **idxs** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (forward\_qm\_map, backward\_qm\_map, forward\_qmmm\_map, backward\_qmmm\_map, natoms\_qm, natoms\_qmmm, qmmm\_frame\_idx).

forward\_qm\_map: A Tensor of type int32. backward\_qm\_map: A Tensor of type int32. forward\_qmmm\_map: A Tensor of type int32. backward\_qmmm\_map: A Tensor of type int32. natoms\_qm: A Tensor of type int32. natoms\_qmmm: A Tensor of type int32. qmmm\_frame\_idx: A Tensor of type int32.



`deepmd.env.op_module.ewald_recp(coord: Any, charge: Any, natoms: Any, box: Any, ewald_beta: float, ewald_h: float, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **charge** – A Tensor. Must have the same type as coord.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **ewald\_beta** – A float.
- **ewald\_h** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (energy, force, virial).

energy: A Tensor. Has the same type as coord. force: A Tensor. Has the same type as coord. virial: A Tensor. Has the same type as coord.

`deepmd.env.op_module.flt_nvnmnd(x: Any, name=None) → Any`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.gelu(x: Any, name=None) → Any`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.gelu_custom(x: Any, name=None) → Any`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.gelu_grad(dy: Any, x: Any, name=None) → Any`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

`deepmd.env.op_module.gelu_grad_custom(dy: Any, x: Any, name=None) → Any`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

`deepmd.env.op_module.gelu_grad_grad(dy: Any, dy_: Any, x: Any, name=None) → Any`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

`deepmd.env.op_module.gelu_grad_grad_custom(dy: Any, dy_: Any, x: Any, name=None) → Any`

TODO: add doc.

Parameters

- **dy** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as dy.
- **x** – A Tensor. Must have the same type as dy.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as dy.

`deepmd.env.op_module.map_aparam(aparam: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None) → Any`

TODO: add doc.

Parameters

- **aparam** – A Tensor. Must be one of the following types: float32, float64.

- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as `aparam`.

```
deepmd.env.op_module.map_flt_nvnmd(x: Any, table: Any, table_grad: Any, table_info: Any,
                                   name=None) → Any
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **table** – A Tensor. Must have the same type as `x`.
- **table\_grad** – A Tensor. Must have the same type as `x`.
- **table\_info** – A Tensor. Must have the same type as `x`.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as `x`.

```
deepmd.env.op_module.matmul_fitnet_nvnmd(x: Any, w: Any, nbitx: int, nbitw: int, normw: int,
                                           name=None) → Any
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as `x`.
- **nbitx** – An int.
- **nbitw** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as `x`.

```
deepmd.env.op_module.matmul_flt2fix_nvnmd(x: Any, w: Any, nbit: int, name=None) → Any
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as `x`.
- **nbit** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as `x`.

`deepmd.env.op_module.matmul_flt_nvnmd(x: Any, w: Any, normx: int, normw: int, name=None) → Any`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **normx** – An int.
- **normw** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.mul_flt_nvnmd(x: Any, w: Any, name=None) → Any`

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as x.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

`deepmd.env.op_module.neighbor_stat(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, rcut: float, name=None)`

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **rcut** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (max\_nbor\_size, min\_nbor\_dist).

max\_nbor\_size: A Tensor of type int32. min\_nbor\_dist: A Tensor. Has the same type as coord.

`deepmd.env.op_module.pair_tab(table_info: Any, table_data: Any, type: Any, rij: Any, nlist: Any, natoms: Any, scale: Any, sel_a, sel_r, name=None)`

TODO: add doc.

Parameters

- **table\_info** – A Tensor of type float64.

- **table\_data** – A Tensor of type float64.
- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **scale** – A Tensor. Must have the same type as rij.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (atom\_energy, force, atom\_virial).

atom\_energy: A Tensor. Has the same type as rij. force: A Tensor. Has the same type as rij. atom\_virial: A Tensor. Has the same type as rij.

```
deepmd.env.op_module.parallel_prod_force_se_a(net_deriv: Any, in_deriv: Any, nlist: Any, natoms:
                                             Any, n_a_sel: int, n_r_sel: int, parallel: bool =
                                             False, start_frac: float = 0, end_frac: float = 1,
                                             name=None) → Any
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **parallel** – An optional bool. Defaults to False.
- **start\_frac** – An optional float. Defaults to 0.
- **end\_frac** – An optional float. Defaults to 1.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_env_mat_a(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any,
                                     davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth:
                                     float, sel_a, sel_r, name=None)
```

Compute the environment matrix for descriptor se\_e2\_a.

Each row of the environment matrix  $\mathcal{R}^i$  can be constructed as follows

$$(\mathcal{R}^i)_j = \begin{bmatrix} \frac{s(r_{ji})}{r_{ji}} \\ \frac{s(r_{ji})x_{ji}}{r_{ji}} \\ \frac{s(r_{ji})y_{ji}}{r_{ji}} \\ \frac{s(r_{ji})z_{ji}}{r_{ji}} \end{bmatrix}$$

In the above equation,  $\mathbf{R}_{ji} = \mathbf{R}_j - \mathbf{R}_i = (x_{ji}, y_{ji}, z_{ji})$  is the relative coordinate and  $r_{ji} = \|\mathbf{R}_{ji}\|$  is its norm. The switching function  $s(r)$  is defined as:

$$s(r) = \begin{cases} \frac{1}{r}, & r < r_s \\ \frac{1}{r} \left\{ \left( \frac{r-r_s}{r_c-r_s} \right)^3 \left( -6 \left( \frac{r-r_s}{r_c-r_s} \right)^2 + 15 \frac{r-r_s}{r_c-r_s} - 10 \right) + 1 \right\}, & r_s \leq r < r_c \\ 0, & r \geq r_c \end{cases}$$

Note that the environment matrix is normalized by `davg` and `dstd`.

#### Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64. The coordinates of atoms.
- **type** – A Tensor of type int32. The types of atoms.
- **natoms** – A Tensor of type int32. The number of atoms. This tensor has the length of `Ntypes + 2`. `natoms[0]`: number of local atoms. `natoms[1]`: total number of atoms held by this processor. `natoms[i]`:  $2 \leq i < \text{Ntypes} + 2$ , number of type *i* atoms.
- **box** – A Tensor. Must have the same type as `coord`. The box of frames.
- **mesh** – A Tensor of type int32. For historical reasons, only the length of the Tensor matters. If size of `mesh` == 6, pbc is assumed. If size of `mesh` == 0, no-pbc is assumed.
- **davg** – A Tensor. Must have the same type as `coord`. Average value of the environment matrix for normalization.
- **dstd** – A Tensor. Must have the same type as `coord`. Standard deviation of the environment matrix for normalization.
- **rcut\_a** – A float. This argument is not used.
- **rcut\_r** – A float. The cutoff radius for the environment matrix.
- **rcut\_r\_smth** – A float. From where the environment matrix should be smoothed.
- **sel\_a** – A list of ints. `sel_a[i]` specifies the maximum number of type *i* atoms in the cut-off radius.
- **sel\_r** – A list of ints. This argument is not used.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (`descript`, `descript_deriv`, `rij`, `nlist`).

`descript`: A Tensor. Has the same type as `coord`. The environment matrix. `descript_deriv`: A Tensor. Has the same type as `coord`. The derivative of the environment matrix. `rij`: A Tensor. Has the same type as `coord`. The distance between the atoms. `nlist`: A Tensor of type int32. The neighbor list of each atom.

```
deepmd.env.op_module.prod_env_mat_a_mix(coord: Any, type: Any, natoms: Any, box: Any, mesh:
                                         Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float,
                                         rcut_r_smth: float, sel_a, sel_r, name=None)
```

Compute the environment matrix mixing the atom types.

The sorting of neighbor atoms depends not on atom types, but on the distance and index. The atoms in nlist matrix will gather forward and thus save space for gaps of types in ProdEnvMatA, resulting in optimized and relative small sel\_a.

The additional outputs are listed as following:

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descript, descript\_deriv, rij, nlist, ntype, nmask).

descript: A Tensor. Has the same type as coord. descript\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. The corresponding atom types in nlist. nmask: A Tensor of type bool. The atom mask in nlist.

```
deepmd.env.op_module.prod_env_mat_a_mix_nvnm_quantize(coord: Any, type: Any, natoms: Any,
                                                       box: Any, mesh: Any, davg: Any, dstd:
                                                       Any, rcut_a: float, rcut_r: float,
                                                       rcut_r_smth: float, sel_a, sel_r,
                                                       name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.

- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (descrpt, descrpt\_deriv, rij, nlist, ntype, nmask).

descrpt: A Tensor. Has the same type as coord. descrpt\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32. ntype: A Tensor of type int32. nmask: A Tensor of type bool.

```
deepmd.env.op_module.prod_env_mat_a_nvnmmd_quantize(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any, davg: Any, dstd: Any, rcut_a: float, rcut_r: float, rcut_r_smth: float, sel_a, sel_r, name=None)
```

TODO: add doc.

#### Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut\_a** – A float.
- **rcut\_r** – A float.
- **rcut\_r\_smth** – A float.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **name** – A name for the operation (optional).

#### Returns

A tuple of Tensor objects (descrpt, descrpt\_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.



```
deepmd.env.op_module.prod_env_mat_r(coord: Any, type: Any, natoms: Any, box: Any, mesh: Any,
                                     davg: Any, dstd: Any, rcut: float, rcut_smth: float, sel,
                                     name=None)
```

TODO: add doc.

Parameters

- **coord** – A Tensor. Must be one of the following types: float32, float64.
- **type** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **box** – A Tensor. Must have the same type as coord.
- **mesh** – A Tensor of type int32.
- **davg** – A Tensor. Must have the same type as coord.
- **dstd** – A Tensor. Must have the same type as coord.
- **rcut** – A float.
- **rcut\_smth** – A float.
- **sel** – A list of ints.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (descrpt, descrpt\_deriv, rij, nlist).

descrpt: A Tensor. Has the same type as coord. descrpt\_deriv: A Tensor. Has the same type as coord. rij: A Tensor. Has the same type as coord. nlist: A Tensor of type int32.

```
deepmd.env.op_module.prod_force(net_deriv: Any, in_deriv: Any, nlist: Any, axis: Any, natoms: Any,
                                n_a_sel: int, n_r_sel: int, name=None) → Any
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_force_norot(net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any,
                                       n_a_sel: int, n_r_sel: int, name=None) → Any
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_force_se_a(net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any,  
                                     n_a_sel: int, n_r_sel: int, name=None) → Any
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_force_se_a_mask(net_deriv: Any, in_deriv: Any, mask: Any, nlist: Any,  
                                           total_atom_num: int, name=None) → Any
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total\_atom\_num** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_force_se_r(net_deriv: Any, in_deriv: Any, nlist: Any, natoms: Any,  
                                     name=None) → Any
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_virial(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, axis: Any,
                                natoms: Any, n_a_sel: int, n_r_sel: int, name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_virial_norot(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms:
                                       Any, n_a_sel: int, n_r_sel: int, name=None)
```

TODO: add doc.

Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_virial_se_a(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None)
```

TODO: add doc.

## Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.prod_virial_se_r(net_deriv: Any, in_deriv: Any, rij: Any, nlist: Any, natoms: Any, name=None)
```

TODO: add doc.

## Parameters

- **net\_deriv** – A Tensor. Must be one of the following types: float32, float64.
- **in\_deriv** – A Tensor. Must have the same type as net\_deriv.
- **rij** – A Tensor. Must have the same type as net\_deriv.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

## Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as net\_deriv. atom\_virial: A Tensor. Has the same type as net\_deriv.

```
deepmd.env.op_module.quantize_nvnm(x: Any, isround: int, nbit1: int, nbit2: int, nbit3: int, name=None) → Any
```

TODO: add doc.

## Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **isround** – An int.
- **nbit1** – An int.
- **nbit2** – An int.
- **nbit3** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

```
deepmd.env.op_module.soft_min_force(du: Any, sw_deriv: Any, nlist: Any, natoms: Any, n_a_sel: int,
                                     n_r_sel: int, name=None) → Any
```

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw\_deriv** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as du.

```
deepmd.env.op_module.soft_min_switch(type: Any, rij: Any, nlist: Any, natoms: Any, sel_a, sel_r,
                                     alpha: float, rmin: float, rmax: float, name=None)
```

TODO: add doc.

Parameters

- **type** – A Tensor of type int32.
- **rij** – A Tensor. Must be one of the following types: float32, float64.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **sel\_a** – A list of ints.
- **sel\_r** – A list of ints.
- **alpha** – A float.
- **rmin** – A float.
- **rmax** – A float.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (sw\_value, sw\_deriv).

sw\_value: A Tensor. Has the same type as rij. sw\_deriv: A Tensor. Has the same type as rij.

`deepmd.env.op_module.soft_min_virial(du: Any, sw_deriv: Any, rij: Any, nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int, name=None)`

TODO: add doc.

Parameters

- **du** – A Tensor. Must be one of the following types: float32, float64.
- **sw\_deriv** – A Tensor. Must have the same type as du.
- **rij** – A Tensor. Must have the same type as du.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (virial, atom\_virial).

virial: A Tensor. Has the same type as du. atom\_virial: A Tensor. Has the same type as du.

`deepmd.env.op_module.tabulate_fusion(table: Any, table_info: Any, em_x: Any, em: Any, last_layer_size: int, name=None) → Any`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

`deepmd.env.op_module.tabulate_fusion_grad(table: Any, table_info: Any, em_x: Any, em: Any, dy: Any, descriptor: Any, name=None)`

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.

- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_grad_grad(table: Any, table_info: Any, em_x: Any, em: Any,
        dz_dy_dem_x: Any, dz_dy_dem: Any, descriptor:
        Any, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_a(table: Any, table_info: Any, em_x: Any, em: Any,
        last_layer_size: int, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_a_grad(table: Any, table_info: Any, em_x: Any, em: Any,
        dy: Any, descriptor: Any, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.

- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_a_grad_grad(table: Any, table_info: Any, em_x: Any, em: Any, dz_dy_dem_x: Any, dz_dy_dem: Any, descriptor: Any, is_sorted: bool = True, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is\_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_atten(table: Any, table_info: Any, em_x: Any, em: Any, two_embed: Any, last_layer_size: int, is_sorted: bool = True, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two\_embed** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **is\_sorted** – An optional bool. Defaults to True.



- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_atten_grad(table: Any, table_info: Any, em_x: Any, em: Any, two_embed: Any, dy: Any, descriptor: Any, is_sorted: bool = True, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two\_embed** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is\_sorted** – An optional bool. Defaults to True.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem, dy\_dtwo).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table. dy\_dtwo: A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_atten_grad_grad(table: Any, table_info: Any, em_x: Any, em: Any, two_embed: Any, dz_dy_dem_x: Any, dz_dy_dem: Any, dz_dy_dtwo: Any, descriptor: Any, is_sorted: bool = True, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **two\_embed** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **dz\_dy\_dtwo** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **is\_sorted** – An optional bool. Defaults to True.

- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_r(table: Any, table_info: Any, em: Any, last_layer_size:
                                           int, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_r_grad(table: Any, table_info: Any, em: Any, dy: Any,
                                                descriptor: Any, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_r_grad_grad(table: Any, table_info: Any, em: Any,
                                                      dz_dy_dem: Any, descriptor: Any,
                                                      name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

---

```
deepmd.env.op_module.tabulate_fusion_se_t(table: Any, table_info: Any, em_x: Any, em: Any,
                                          last_layer_size: int, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **last\_layer\_size** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_t_grad(table: Any, table_info: Any, em_x: Any, em: Any,
                                                dy: Any, descriptor: Any, name=None)
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dy** – A Tensor. Must have the same type as table.
- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A tuple of Tensor objects (dy\_dem\_x, dy\_dem).

dy\_dem\_x: A Tensor. Has the same type as table. dy\_dem: A Tensor. Has the same type as table.

```
deepmd.env.op_module.tabulate_fusion_se_t_grad_grad(table: Any, table_info: Any, em_x: Any, em: Any,
                                                    dz_dy_dem_x: Any, dz_dy_dem: Any,
                                                    descriptor: Any, name=None) → Any
```

TODO: add doc.

Parameters

- **table** – A Tensor. Must be one of the following types: float32, float64.
- **table\_info** – A Tensor. Must have the same type as table.
- **em\_x** – A Tensor. Must have the same type as table.
- **em** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem\_x** – A Tensor. Must have the same type as table.
- **dz\_dy\_dem** – A Tensor. Must have the same type as table.

- **descriptor** – A Tensor. Must have the same type as table.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as table.

```
deepmd.env.op_module.tanh4_flt_nvnm(x: Any, name=None) → Any
```

TODO: add doc.

Parameters

- **x** – A Tensor. Must be one of the following types: float32, float64.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as x.

```
deepmd.env.op_module.unaggregated_dy2_dx(z: Any, w: Any, dy_dx: Any, dy2_dx: Any, ybar: Any,
                                           functype: Any, name=None) → Any
```

TODO: add doc.

Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy\_dx** – A Tensor. Must have the same type as z.
- **dy2\_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as z.

```
deepmd.env.op_module.unaggregated_dy2_dx_s(y: Any, dy: Any, w: Any, xbar: Any, functype: Any,
                                             name=None) → Any
```

TODO: add doc.

Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **dy** – A Tensor. Must have the same type as y.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as y.

```
deepmd.env.op_module.unaggregated_dy_dx(z: Any, w: Any, dy_dx: Any, ybar: Any, functype: Any,
                                           name=None) → Any
```

TODO: add doc.

## Parameters

- **z** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as z.
- **dy\_dx** – A Tensor. Must have the same type as z.
- **ybar** – A Tensor. Must have the same type as z.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

## Returns

A Tensor. Has the same type as z.

```
deepmd.env.op_module.unaggregated_dy_dx_s(y: Any, w: Any, xbar: Any, functype: Any,
                                           name=None) → Any
```

TODO: add doc.

## Parameters

- **y** – A Tensor. Must be one of the following types: float32, float64.
- **w** – A Tensor. Must have the same type as y.
- **xbar** – A Tensor. Must have the same type as y.
- **functype** – A Tensor of type int32.
- **name** – A name for the operation (optional).

## Returns

A Tensor. Has the same type as y.

## 19.2 op\_grads\_module

Python wrappers around TensorFlow ops.

This file is MACHINE GENERATED! Do not edit.

```
class deepmd.env.op_grads_module.Annotated(*args, **kwargs)
```

Bases: `object`

Add context specific metadata to a type.

Example: `Annotated[int, runtime_check.Unsigned]` indicates to the hypothetical `runtime_check` module that this type is an unsigned int. Every other consumer of this type can ignore this metadata and treat this type as `int`.

The first argument to `Annotated` must be a valid type.

Details:

- It's an error to call `Annotated` with less than two arguments.
- Nested `Annotated` are flattened:

```
Annotated[Annotated[T, Ann1, Ann2], Ann3] == Annotated[T, Ann1, Ann2, Ann3]
```

- Instantiating an annotated type is equivalent to instantiating the underlying type:

```
Annotated[C, Ann1](5) == C(5)
```

- Annotated can be used as a generic type alias:

```
Optimized = Annotated[T, runtime.Optimize()]
Optimized[int] == Annotated[int, runtime.Optimize()]

OptimizedList = Annotated[List[T], runtime.Optimize()]
OptimizedList[int] == Annotated[List[int], runtime.Optimize()]
```

```
deepmd.env.op_grads_module.ProdForceGrad(grad, net_deriv, in_deriv, nlist, axis, natoms, n_a_sel,
                                          n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdForceSeAGrad(grad, net_deriv, in_deriv, nlist, natoms, n_a_sel,
                                             n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdForceSeAMaskGrad(grad, net_deriv, in_deriv, mask, nlist,
                                                  total_atom_num, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total\_atom\_num** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdForceSeRGrad(grad, net_deriv, in_deriv, nlist, natoms, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdVirialGrad(grad, net_deriv, in_deriv, rij, nlist, axis, natoms, n_a_sel,
                                           n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdVirialSeAGrad(grad, net_deriv, in_deriv, rij, nlist, natoms, n_a_sel,
                                              n_r_sel, name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.ProdVirialSeRGrad(grad, net_deriv, in_deriv, rij, nlist, natoms,
                                              name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.SoftMinForceGrad(grad, du, sw_deriv, nlist, natoms, n_a_sel, n_r_sel,
                                              name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.



- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.SoftMinVirialGrad(grad, du, sw_deriv, rij, nlist, natoms, n_a_sel, n_r_sel,
                                              name=None)
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_force_grad(grad: Any, net_deriv: Any, in_deriv: Any, nlist: Any,
                                             axis: Any, natoms: Any, n_a_sel: int, n_r_sel: int,
                                             name=None) → Any
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_force_se_a_grad(grad: Any, net_deriv: Any, in_deriv: Any, nlist:
                                                Any, natoms: Any, n_a_sel: int, n_r_sel: int,
                                                name=None) → Any
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_force_se_a_mask_grad(grad: Any, net_deriv: Any, in_deriv: Any,
                                                       mask: Any, nlist: Any, total_atom_num:
                                                       int, name=None) → Any
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **mask** – A Tensor of type int32.
- **nlist** – A Tensor of type int32.
- **total\_atom\_num** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.prod_force_se_r_grad(grad: Any, net_deriv: Any, in_deriv: Any, nlist:
                                                  Any, natoms: Any, name=None) → Any
```

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.prod_virial_grad`(grad: *Any*, net\_deriv: *Any*, in\_deriv: *Any*, rij: *Any*, nlist: *Any*, axis: *Any*, natoms: *Any*, n\_a\_sel: int, n\_r\_sel: int, name=None) → *Any*

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **axis** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.prod_virial_se_a_grad`(grad: *Any*, net\_deriv: *Any*, in\_deriv: *Any*, rij: *Any*, nlist: *Any*, natoms: *Any*, n\_a\_sel: int, n\_r\_sel: int, name=None) → *Any*

TODO: add doc.

Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.

`deepmd.env.op_grads_module.prod_virial_se_r_grad`(grad: *Any*, net\_deriv: *Any*, in\_deriv: *Any*, rij: *Any*, nlist: *Any*, natoms: *Any*, name=None) → *Any*

TODO: add doc.

## Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **net\_deriv** – A Tensor. Must have the same type as grad.
- **in\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **name** – A name for the operation (optional).

## Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.soft_min_force_grad(grad: Any, du: Any, sw_deriv: Any, nlist: Any,  
                                                natoms: Any, n_a_sel: int, n_r_sel: int,  
                                                name=None) → Any
```

TODO: add doc.

## Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw\_deriv** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.
- **name** – A name for the operation (optional).

## Returns

A Tensor. Has the same type as grad.

```
deepmd.env.op_grads_module.soft_min_virial_grad(grad: Any, du: Any, sw_deriv: Any, rij: Any,  
                                                nlist: Any, natoms: Any, n_a_sel: int, n_r_sel: int,  
                                                name=None) → Any
```

TODO: add doc.

## Parameters

- **grad** – A Tensor. Must be one of the following types: float32, float64.
- **du** – A Tensor. Must have the same type as grad.
- **sw\_deriv** – A Tensor. Must have the same type as grad.
- **rij** – A Tensor. Must have the same type as grad.
- **nlist** – A Tensor of type int32.
- **natoms** – A Tensor of type int32.
- **n\_a\_sel** – An int.
- **n\_r\_sel** – An int.

- **name** – A name for the operation (optional).

Returns

A Tensor. Has the same type as grad.



## 20.1 Class Hierarchy

- Namespace deepmd
  - Struct deepmd\_exception
  - Struct NeighborListData
  - Struct tf\_exception
  - Class AtomMap
  - Class DeepPot
  - Class DeepPotBase
  - Class DeepPotModelDevi
  - Class DeepPotTF
  - Class DeepTensor
  - Class DeepTensorBase
  - Class DeepTensorTF
  - Class DipoleChargeModifier
  - Class DipoleChargeModifierBase
  - Class DipoleChargeModifierTF
  - Enum DPBackend

## 20.2 File Hierarchy

- dir\_source
  - dir\_source\_api\_cc
    - \*dir\_source\_api\_cc\_include
      - file\_source\_api\_cc\_include\_AtomMap.h
      - file\_source\_api\_cc\_include\_common.h
      - file\_source\_api\_cc\_include\_DataModifier.h

- `file_source_api_cc_include_DataModifierTF.h`
- `file_source_api_cc_include_DeepPot.h`
- `file_source_api_cc_include_DeepPotTF.h`
- `file_source_api_cc_include_DeepTensor.h`
- `file_source_api_cc_include_DeepTensorTF.h`
- `file_source_api_cc_include_tf_private.h`
- `file_source_api_cc_include_tf_public.h`

## 20.3 Full API

### 20.3.1 Namespaces

#### Namespace `deepmd`

##### Contents

- [Classes](#)
- [Enums](#)
- [Functions](#)
- [Typedefs](#)

#### Classes

- [Struct `deepmd\_exception`](#)
- [Struct `NeighborListData`](#)
- [Struct `tf\_exception`](#)
- [Class `AtomMap`](#)
- [Class `DeepPot`](#)
- [Class `DeepPotBase`](#)
- [Class `DeepPotModelDevi`](#)
- [Class `DeepPotTF`](#)
- [Class `DeepTensor`](#)
- [Class `DeepTensorBase`](#)
- [Class `DeepTensorTF`](#)
- [Class `DipoleChargeModifier`](#)
- [Class `DipoleChargeModifierBase`](#)
- [Class `DipoleChargeModifierTF`](#)



## Enums

- Enum DPBackend

## Functions

- Function `deepmd::check_status`
- Function `deepmd::convert_pbtxt_to_pb`
- Function `deepmd::get_env_nthreads`
- Function `deepmd::load_op_library`
- Function `deepmd::model_compatible`
- Function `deepmd::name_prefix`
- Function `deepmd::print_summary`
- Function `deepmd::read_file_to_string`
- Template Function `deepmd::select_by_type`
- Template Function `deepmd::select_map(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&, const int&, const int&, const int&)`
- Template Function `deepmd::select_map(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&, const int&, const int&, const int&)`
- Template Function `deepmd::select_map_inv(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&)`
- Template Function `deepmd::select_map_inv(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&)`
- Template Function `deepmd::select_real_atoms`
- Template Function `deepmd::select_real_atoms_coord`
- Function `deepmd::session_get_dtype`
- Template Function `deepmd::session_get_scalar`
- Template Function `deepmd::session_get_vector`
- Template Function `deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, const double&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const std::string, const bool)`
- Template Function `deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, InputNlist&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const int, const int, const std::string, const bool)`
- Template Function `deepmd::session_input_tensors_mixed_type`

## Typedefs

- Typedef `deepmd::ENERGYTYPE`
- Typedef `deepmd::STRINGTYPE`

## Namespace `tensorflow`

### 20.3.2 Classes and Structs

#### Struct `deepmd_exception`

- Defined in `file_source_api_cc_include_common.h`

#### Inheritance Relationships

#### Derived Type

- `public deepmd::tf_exception(Struct tf_exception)`

#### Struct Documentation

struct `deepmd_exception`

Subclassed by `deepmd::tf_exception`

#### Struct `NeighborListData`

- Defined in `file_source_api_cc_include_common.h`

#### Struct Documentation

struct `NeighborListData`

#### Public Functions

void `copy_from_nlist`(const `InputNlist` &`inlist`)

void `shuffle`(const `std::vector<int>` &`fwd_map`)

void `shuffle`(const `deepmd::AtomMap` &`map`)

void `shuffle_exclude_empty`(const `std::vector<int>` &`fwd_map`)

void `make_inlist`(`InputNlist` &`inlist`)

## Public Members

`std::vector<int> ilist`

Array stores the core region atom's index.

`std::vector<std::vector<int>> jlist`

Array stores the core region atom's neighbor index.

`std::vector<int> numneigh`

Array stores the number of neighbors of core region atoms.

`std::vector<int*> firstneigh`

Array stores the the location of the first neighbor of core region atoms.

## Struct `tf_exception`

- Defined in `file_source_api_cc_include_common.h`

## Inheritance Relationships

### Base Type

- `public deepmd_exception (Struct deepmd_exception)`

## Struct Documentation

`struct tf_exception : public deepmd_exception`

Throw exception if TensorFlow doesn't work.

### Public Functions

`inline tf_exception()`

`inline tf_exception(const std::string &msg)`

## Class `AtomMap`

- Defined in `file_source_api_cc_include_AtomMap.h`

## Class Documentation

class **AtomMap**

### Public Functions

**AtomMap**()

**AtomMap**(const std::vector<int>::const\_iterator in\_begin, const std::vector<int>::const\_iterator in\_end)

template<typename VALUETYPE>  
void **forward**(typename std::vector<VALUETYPE>::iterator out, const typename std::vector<VALUETYPE>::const\_iterator in, const int stride = 1, const int nframes = 1, const int nall = 0) const

template<typename VALUETYPE>  
void **backward**(typename std::vector<VALUETYPE>::iterator out, const typename std::vector<VALUETYPE>::const\_iterator in, const int stride = 1, const int nframes = 1, const int nall = 0) const

inline const std::vector<int> &**get\_type**() const

inline const std::vector<int> &**get\_fwd\_map**() const

inline const std::vector<int> &**get\_bkw\_map**() const

## Class DeepPot

- Defined in file\_source\_api\_cc\_include\_DeepPot.h

## Class Documentation

class **DeepPot**

Deep Potential to automatically switch backends.

### Unnamed Group

template<typename VALUETYPE>  
void **compute**(ENERGYTYPE &ener, std::vector<VALUETYPE> &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.

- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE>
void compute(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
            std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
            std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const
            std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

### Unnamed Group

```
template<typename VALUETYPE>
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
            std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost,
            const InputNlist &inlist, const int &ago, const std::vector<VALUETYPE> &fparam =
            std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam =
            std::vector<VALUETYPE>())
```

Evaluate the energy, force and virial by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE>
void compute(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
            std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost,
            const InputNlist &inlist, const int &ago, const std::vector<VALUETYPE> &fparam =
            std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam =
            std::vector<VALUETYPE>())
```

### Unnamed Group

```
template<typename VALUETYPE>
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy,
            std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
            const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
            std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const
            std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE>
void compute(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy,
            std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
            const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
            std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const
            std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

## Unnamed Group

```
template<typename VALUETYPE>
void compute(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
             std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy,
             std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
             const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int
             nghost, const InputNlist &imp_list, const int &ago, const std::vector<VALUETYPE>
             &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE>
             &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **imp\_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE>
void compute(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE> &force,
             std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_energy,
             std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
             const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int
             nghost, const InputNlist &imp_list, const int &ago, const std::vector<VALUETYPE>
             &fparam = std::vector<VALUETYPE>(), const std::vector<VALUETYPE>
             &aparam = std::vector<VALUETYPE>())
```

### Unnamed Group

```
template<typename VALUETYPE>
void compute_mixed_type(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
                        std::vector<VALUETYPE> &virial, const int &nframes, const
                        std::vector<VALUETYPE> &coord, const std::vector<int> &atype,
                        const std::vector<VALUETYPE> &box, const
                        std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(),
                        const std::vector<VALUETYPE> &aparam =
                        std::vector<VALUETYPE>())
```

Evaluate the energy, force, and virial with the mixed type by using this DP.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE>
void compute_mixed_type(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE>
                        &force, std::vector<VALUETYPE> &virial, const int &nframes, const
                        std::vector<VALUETYPE> &coord, const std::vector<int> &atype,
                        const std::vector<VALUETYPE> &box, const
                        std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(),
                        const std::vector<VALUETYPE> &aparam =
                        std::vector<VALUETYPE>())
```

### Unnamed Group

```
template<typename VALUETYPE>
void compute_mixed_type(ENERGYTYPE &ener, std::vector<VALUETYPE> &force,
                        std::vector<VALUETYPE> &virial, std::vector<VALUETYPE>
                        &atom_energy, std::vector<VALUETYPE> &atom_virial, const int
                        &nframes, const std::vector<VALUETYPE> &coord, const
                        std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
                        std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(),
                        const std::vector<VALUETYPE> &aparam =
                        std::vector<VALUETYPE>())
```

Evaluate the energy, force, and virial with the mixed type by using this DP.



## Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE>
void compute_mixed_type(std::vector<ENERGYTYPE> &ener, std::vector<VALUETYPE>
    &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE>
    &atom_energy, std::vector<VALUETYPE> &atom_virial, const int
    &nframes, const std::vector<VALUETYPE> &coord, const
    std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
    std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(),
    const std::vector<VALUETYPE> &aparam =
    std::vector<VALUETYPE>())
```

## Public Functions

**DeepPot()**

DP constructor without initialization.

**~DeepPot()**

**DeepPot**(const std::string &model, const int &gpu\_rank = 0, const std::string &file\_content = "")

DP constructor with initialization.

## Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

```
void init(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")
```

Initialize the DP.

## Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

void **print\_summary**(const std::string &pre) const

Print the DP summary to the screen.

Parameters

**pre** – [in] The prefix to each line.

double **cutoff**() const

Get the cutoff radius.

Returns

The cutoff radius.

int **numb\_types**() const

Get the number of types.

Returns

The number of types.

int **numb\_types\_spin**() const

Get the number of types with spin.

Returns

The number of types with spin.

int **dim\_fparam**() const

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

int **dim\_aparam**() const

Get the dimension of the atomic parameter.

Returns

The dimension of the atomic parameter.

void **get\_type\_map**(std::string &type\_map)

Get the type map (element name of the atom types) of this model.

Parameters

**type\_map** – [out] The type map of this model.

bool **is\_aparam\_nall**() const

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters

**aparam\_nall** – [out] whether the atom dimension of aparam is nall instead of fparam.

## Class DeepPotBase

- Defined in file `_source_api_cc_include_DeepPot.h`

## Inheritance Relationships

## Derived Type

- `public deepmd::DeepPotTF` ([Class DeepPotTF](#))

## Class Documentation

class **DeepPotBase**

Deep Potential.

Subclassed by [deepmd::DeepPotTF](#)

## Unnamed Group

```
virtual void compute(std::vector<double> &ener, std::vector<double> &force,
                    std::vector<double> &virial, std::vector<double> &atom_energy,
                    std::vector<double> &atom_virial, const std::vector<double> &coord, const
                    std::vector<int> &atype, const std::vector<double> &box, const
                    std::vector<double> &fparam = std::vector<double>(), const
                    std::vector<double> &aparam = std::vector<double>()) = 0
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

---

Note: The double precision interface is used by i-PI, GROMACS, ABACUS, and CP2k.

---

## Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size `nframes x natoms x 3`.
- **atype** – [in] The atom types. The list should contain `natoms` ints.
- **box** – [in] The cell of the region. The array should be of size `nframes x 9`.
- **fparam** – [in] The frame parameter. The array can be of size : `nframes x dim_fparam`. `dim_fparam`. Then all frames are assumed to be provided with the same `fparam`.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
virtual void computew(std::vector<double> &ener, std::vector<float> &force, std::vector<float>
    &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial,
    const std::vector<float> &coord, const std::vector<int> &atype, const
    std::vector<float> &box, const std::vector<float> &fparam =
    std::vector<float>(), const std::vector<float> &aparam =
    std::vector<float>()) = 0
```

### Unnamed Group

```
virtual void computew(std::vector<double> &ener, std::vector<double> &force,
    std::vector<double> &virial, std::vector<double> &atom_energy,
    std::vector<double> &atom_virial, const std::vector<double> &coord, const
    std::vector<int> &atype, const std::vector<double> &box, const int nghost,
    const InputNlist &inlist, const int &ago, const std::vector<double> &fparam
    = std::vector<double>(), const std::vector<double> &aparam =
    std::vector<double>()) = 0
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

---

Note: The double precision interface is used by LAMMPS and AMBER.

---

### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **inp\_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
virtual void computew(std::vector<double> &ener, std::vector<float> &force, std::vector<float>
    &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial,
    const std::vector<float> &coord, const std::vector<int> &atype, const
    std::vector<float> &box, const int nghost, const InputNlist &inlist, const int
    &ago, const std::vector<float> &fparam = std::vector<float>(), const
    std::vector<float> &aparam = std::vector<float>()) = 0
```

### Unnamed Group

```
virtual void computew_mixed_type(std::vector<double> &ener, std::vector<double> &force,
    std::vector<double> &virial, std::vector<double>
    &atom_energy, std::vector<double> &atom_virial, const int
    &nframes, const std::vector<double> &coord, const
    std::vector<int> &atype, const std::vector<double> &box, const
    std::vector<double> &fparam = std::vector<double>(), const
    std::vector<double> &aparam = std::vector<double>()) = 0
```

Evaluate the energy, force, and virial with the mixed type by using this DP.

---

Note: At this time, no external program uses this interface.

---

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
virtual void computew_mixed_type(std::vector<double> &ener, std::vector<float> &force,
    std::vector<float> &virial, std::vector<float> &atom_energy,
    std::vector<float> &atom_virial, const int &nframes, const
    std::vector<float> &coord, const std::vector<int> &atype, const
    std::vector<float> &box, const std::vector<float> &fparam =
    std::vector<float>(), const std::vector<float> &aparam =
    std::vector<float>()) = 0
```

## Public Functions

`inline DeepPotBase()`

DP constructor without initialization.

`inline virtual ~DeepPotBase()`

`DeepPotBase(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")`

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

`virtual void init(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "") = 0`

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

`virtual double cutoff() const = 0`

Get the cutoff radius.

Returns

The cutoff radius.

`virtual int numb_types() const = 0`

Get the number of types.

Returns

The number of types.

`virtual int numb_types_spin() const = 0`

Get the number of types with spin.

Returns

The number of types with spin.

`virtual int dim_fparam() const = 0`

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

`virtual int dim_aparam() const = 0`

Get the dimension of the atomic parameter.

Returns

The dimension of the atomic parameter.

```
virtual void get_type_map(std::string &type_map) = 0
    Get the type map (element name of the atom types) of this model.
    Parameters
        type_map – [out] The type map of this model.
virtual bool is_aparam_nall() const = 0
    Get whether the atom dimension of aparam is nall instead of fparam.
    Parameters
        aparam_nall – [out] whether the atom dimension of aparam is nall instead of fparam.
```

## Class DeepPotModelDevi

- Defined in file `source_api_cc_include_DeepPot.h`

## Class Documentation

class **DeepPotModelDevi**

### Public Functions

**DeepPotModelDevi**()

DP model deviation constructor without initialization.

**~DeepPotModelDevi**()

**DeepPotModelDevi**(const std::vector<std::string> &models, const int &gpu\_rank = 0, const std::vector<std::string> &file\_contents = std::vector<std::string>())

DP model deviation constructor with initialization.

Parameters

- **models** – [in] The names of the frozen model files.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_contents** – [in] The contents of the model files. If it is not empty, DP will read from the strings instead of the files.

void **init**(const std::vector<std::string> &models, const int &gpu\_rank = 0, const std::vector<std::string> &file\_contents = std::vector<std::string>())

Initialize the DP model deviation contrcutor.

Parameters

- **models** – [in] The names of the frozen model files.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_contents** – [in] The contents of the model files. If it is not empty, DP will read from the strings instead of the files.

template<typename VALUETYPE>

```
void compute(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>>
    &all_force, std::vector<std::vector<VALUETYPE>> &all_virial, const
    std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const
    std::vector<VALUETYPE> &box, const int nghost, const InputNlist &imp_list, const
    int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(),
    const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force and virial by using these DP models.

Parameters

- **all\_ener** – [out] The system energies of all models.
- **all\_force** – [out] The forces on each atom of all models.
- **all\_virial** – [out] The virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **imp\_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. dim\_aparam. Then all frames and atoms are provided with the same aparam.

```
template<typename VALUETYPE>
void compute(std::vector<ENERGYTYPE> &all_ener, std::vector<std::vector<VALUETYPE>>
    &all_force, std::vector<std::vector<VALUETYPE>> &all_virial,
    std::vector<std::vector<VALUETYPE>> &all_atom_energy,
    std::vector<std::vector<VALUETYPE>> &all_atom_virial, const
    std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const
    std::vector<VALUETYPE> &box, const int nghost, const InputNlist &imp_list, const
    int &ago, const std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(),
    const std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using these DP models.

Parameters

- **all\_ener** – [out] The system energies of all models.
- **all\_force** – [out] The forces on each atom of all models.
- **all\_virial** – [out] The virials of all models.
- **all\_atom\_energy** – [out] The atomic energies of all models.
- **all\_atom\_virial** – [out] The atomic virials of all models.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.



- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp\_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam. dim\_aparam. Then all frames and atoms are provided with the same aparam.

inline double **cutoff**() const

Get the cutoff radius.

Returns

The cutoff radius.

inline int **numb\_types**() const

Get the number of types.

Returns

The number of types.

inline int **numb\_types\_spin**() const

Get the number of types with spin.

Returns

The number of types with spin.

inline int **dim\_fparam**() const

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

inline int **dim\_aparam**() const

Get the dimension of the atomic parameter.

Returns

The dimension of the atomic parameter.

template<typename **VALUETYPE**>

void **compute\_avg**(**VALUETYPE** &dener, const std::vector<**VALUETYPE**> &all\_energy)

Compute the average energy.

Parameters

- **dener** – [out] The average energy.
- **all\_energy** – [in] The energies of all models.

template<typename **VALUETYPE**>

```
void compute_avg(std::vector<VALUETYPE> &avg, const
                std::vector<std::vector<VALUETYPE>> &xx)
```

Compute the average of vectors.

Parameters

- **avg** – [out] The average of vectors.
- **xx** – [in] The vectors of all models.

```
template<typename VALUETYPE>
void compute_std(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg, const
                std::vector<std::vector<VALUETYPE>> &xx, const int &stride)
```

Compute the standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **xx** – [in] The vectors of all models.
- **stride** – [in] The stride to compute the deviation.

```
template<typename VALUETYPE>
void compute_relative_std(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE>
                          &avg, const VALUETYPE eps, const int &stride)
```

Compute the relative standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **eps** – [in] The level parameter for computing the deviation.
- **stride** – [in] The stride to compute the deviation.

```
template<typename VALUETYPE>
void compute_std_e(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg,
                  const std::vector<std::vector<VALUETYPE>> &xx)
```

Compute the standard deviation of atomic energies.

Parameters

- **std** – [out] The standard deviation of atomic energies.
- **avg** – [in] The average of atomic energies.
- **xx** – [in] The vectors of all atomic energies.

```
template<typename VALUETYPE>
void compute_std_f(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg,
                  const std::vector<std::vector<VALUETYPE>> &xx)
```

Compute the standard deviation of forces.

Parameters

- **std** – [out] The standard deviation of forces.
- **avg** – [in] The average of forces.
- **xx** – [in] The vectors of all forces.

```
template<typename VALUETYPE>
void compute_relative_std_f(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE>
                           &avg, const VALUETYPE eps)
```

Compute the relative standard deviation of forces.

Parameters

- **std** – [out] The relative standard deviation of forces.
- **avg** – [in] The relative average of forces.
- **eps** – [in] The level parameter for computing the deviation.

```
inline bool is_aparam_nall() const
```

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters

**aparam\_nall** – [out] whether the atom dimension of aparam is nall instead of fparam.

## Class DeepPotTF

- Defined in file\_source\_api\_cc\_include\_DeepPotTF.h

## Inheritance Relationships

### Base Type

- public deepmd::DeepPotBase (Class DeepPotBase)

## Class Documentation

```
class DeepPotTF : public deepmd::DeepPotBase
```

TensorFlow implementation for Deep Potential.

### Public Functions

```
DeepPotTF()
```

DP constructor without initialization.

```
~DeepPotTF()
```

```
DeepPotTF(const std::string &model, const int &gpu_rank = 0, const std::string &file_content = "")
```

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

```
virtual void init(const std::string &model, const int &gpu_rank = 0, const std::string &file_content
                = "")
```

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **file\_content** – [in] The content of the model file. If it is not empty, DP will read from the string instead of the file.

```
inline virtual double cutoff() const
```

Get the cutoff radius.

Returns

The cutoff radius.

```
inline virtual int numb_types() const
```

Get the number of types.

Returns

The number of types.

```
inline virtual int numb_types_spin() const
```

Get the number of types with spin.

Returns

The number of types with spin.

```
inline virtual int dim_fparam() const
```

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

```
inline virtual int dim_aparam() const
```

Get the dimension of the atomic parameter.

Returns

The dimension of the atomic parameter.

```
virtual void get_type_map(std::string &type_map)
```

Get the type map (element name of the atom types) of this model.

Parameters

**type\_map** – [out] The type map of this model.

```
inline virtual bool is_aparam_nall() const
```

Get whether the atom dimension of aparam is nall instead of fparam.

Parameters

**aparam\_nall** – [out] whether the atom dimension of aparam is nall instead of fparam.

```
virtual void computew(std::vector<double> &ener, std::vector<double> &force,
                    std::vector<double> &virial, std::vector<double> &atom_energy,
                    std::vector<double> &atom_virial, const std::vector<double> &coord, const
                    std::vector<int> &atype, const std::vector<double> &box, const
                    std::vector<double> &fparam = std::vector<double>(), const
                    std::vector<double> &aparam = std::vector<double>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

---

Note: The double precision interface is used by i-PI, GROMACS, ABACUS, and CP2k.

---

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
virtual void computew(std::vector<double> &ener, std::vector<float> &force, std::vector<float>
    &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial,
    const std::vector<float> &coord, const std::vector<int> &atype, const
    std::vector<float> &box, const std::vector<float> &fparam =
    std::vector<float>(), const std::vector<float> &aparam =
    std::vector<float>())
```

```
virtual void computew(std::vector<double> &ener, std::vector<double> &force,
    std::vector<double> &virial, std::vector<double> &atom_energy,
    std::vector<double> &atom_virial, const std::vector<double> &coord, const
    std::vector<int> &atype, const std::vector<double> &box, const int nghost,
    const InputNlist &inlist, const int &ago, const std::vector<double> &fparam
    = std::vector<double>(), const std::vector<double> &aparam =
    std::vector<double>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

---

Note: The double precision interface is used by LAMMPS and AMBER.

---

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.

- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **nghost** – [in] The number of ghost atoms.
- **lmp\_list** – [in] The input neighbour list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
virtual void computew(std::vector<double> &ener, std::vector<float> &force, std::vector<float>
    &virial, std::vector<float> &atom_energy, std::vector<float> &atom_virial,
    const std::vector<float> &coord, const std::vector<int> &atype, const
    std::vector<float> &box, const int nghost, const InputNlist &inlist, const int
    &ago, const std::vector<float> &fparam = std::vector<float>(), const
    std::vector<float> &aparam = std::vector<float>())
```

```
virtual void computew_mixed_type(std::vector<double> &ener, std::vector<double> &force,
    std::vector<double> &virial, std::vector<double>
    &atom_energy, std::vector<double> &atom_virial, const int
    &nframes, const std::vector<double> &coord, const
    std::vector<int> &atype, const std::vector<double> &box,
    const std::vector<double> &fparam = std::vector<double>(),
    const std::vector<double> &aparam = std::vector<double>())
```

Evaluate the energy, force, and virial with the mixed type by using this DP.

---

Note: At this time, no external program uses this interface.

---

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The array should be of size nframes x natoms.

- **box** – [in] The cell of the region. The array should be of size nframes x 9.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
virtual void computew_mixed_type(std::vector<double> &ener, std::vector<float> &force,
                                std::vector<float> &virial, std::vector<float> &atom_energy,
                                std::vector<float> &atom_virial, const int &nframes, const
                                std::vector<float> &coord, const std::vector<int> &atype, const
                                std::vector<float> &box, const std::vector<float> &fparam =
                                std::vector<float>(), const std::vector<float> &aparam =
                                std::vector<float>())
```

## Class DeepTensor

- Defined in file\_source\_api\_cc\_include\_DeepTensor.h

## Class Documentation

class **DeepTensor**

Deep Tensor.

### Public Functions

**DeepTensor()**

Deep Tensor constructor without initialization.

**~DeepTensor()**

**DeepTensor**(const std::string &model, const int &gpu\_rank = 0, const std::string &name\_scope = "")

Deep Tensor constructor with initialization..

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] Name scopes of operations.

void **init**(const std::string &model, const int &gpu\_rank = 0, const std::string &name\_scope = "")

Initialize the Deep Tensor.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] Name scopes of operations.

```
void print_summary(const std::string &pre) const
```

Print the DP summary to the screen.

Parameters

**pre** – [in] The prefix to each line.

```
template<typename VALUETYPE>
```

```
void compute(std::vector<VALUETYPE> &value, const std::vector<VALUETYPE> &coord, const  
             std::vector<int> &atype, const std::vector<VALUETYPE> &box)
```

Evaluate the value by using this model.

Parameters

- **value** – [out] The value to evaluate, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

```
template<typename VALUETYPE>
```

```
void compute(std::vector<VALUETYPE> &value, const std::vector<VALUETYPE> &coord, const  
             std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost,  
             const InputNlist &inlist)
```

Evaluate the value by using this model.

Parameters

- **value** – [out] The value to evaluate, usually would be the atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

```
template<typename VALUETYPE>
```

```
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,  
             std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const  
             std::vector<int> &atype, const std::vector<VALUETYPE> &box)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

```
template<typename VALUETYPE>
```



```
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord, const
            std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int nghost,
            const InputNlist &inlist)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

```
template<typename VALUETYPE>
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor,
            std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
            const std::vector<int> &atype, const std::vector<VALUETYPE> &box)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom\_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.

```
template<typename VALUETYPE>
void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE> &force,
            std::vector<VALUETYPE> &virial, std::vector<VALUETYPE> &atom_tensor,
            std::vector<VALUETYPE> &atom_virial, const std::vector<VALUETYPE> &coord,
            const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const int
            nghost, const InputNlist &inlist)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.

- **force** – [out] The component-wise force of the global tensor, size `odim x natoms x 3`.
- **virial** – [out] The component-wise virial of the global tensor, size `odim x 9`.
- **atom\_tensor** – [out] The atomic tensor value of the model, size `natoms x odim`.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size `odim x natoms x 9`.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The list should contain `natoms` ints.
- **box** – [in] The cell of the region. The array should be of size `9`.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.

double **cutoff**() const

Get the cutoff radius.

Returns

The cutoff radius.

int **numb\_types**() const

Get the number of types.

Returns

The number of types.

int **output\_dim**() const

Get the output dimension.

Returns

The output dimension.

const std::vector<int> &**sel\_types**() const

Get the list of sel types.

Returns

The list of sel types.

void **get\_type\_map**(std::string &type\_map)

Get the type map (element name of the atom types) of this model.

Parameters

**type\_map** – [out] The type map of this model.

## Class **DeepTensorBase**

- Defined in `file_source_api_cc_include_DeepTensor.h`

## Inheritance Relationships

### Derived Type

- `public deepmd::DeepTensorTF (Class DeepTensorTF)`

### Class Documentation

class **DeepTensorBase**

Deep Tensor.

Subclassed by `deepmd::DeepTensorTF`

#### Unnamed Group

```
virtual void computew(std::vector<double> &global_tensor, std::vector<double> &force,
    std::vector<double> &virial, std::vector<double> &atom_tensor,
    std::vector<double> &atom_virial, const std::vector<double> &coord, const
    std::vector<int> &atype, const std::vector<double> &box, const bool
    request_deriv) = 0
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom\_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **request\_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

```
virtual void computew(std::vector<float> &global_tensor, std::vector<float> &force,
    std::vector<float> &virial, std::vector<float> &atom_tensor,
    std::vector<float> &atom_virial, const std::vector<float> &coord, const
    std::vector<int> &atype, const std::vector<float> &box, const bool
    request_deriv) = 0
```

## Unnamed Group

```
virtual void computew(std::vector<double> &global_tensor, std::vector<double> &force,
    std::vector<double> &virial, std::vector<double> &atom_tensor,
    std::vector<double> &atom_virial, const std::vector<double> &coord, const
    std::vector<int> &atype, const std::vector<double> &box, const int nghost,
    const InputNlist &inlist, const bool request_deriv) = 0
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom\_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.
- **request\_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

```
virtual void computew(std::vector<float> &global_tensor, std::vector<float> &force,
    std::vector<float> &virial, std::vector<float> &atom_tensor,
    std::vector<float> &atom_virial, const std::vector<float> &coord, const
    std::vector<int> &atype, const std::vector<float> &box, const int nghost,
    const InputNlist &inlist, const bool request_deriv) = 0
```

## Public Functions

```
inline DeepTensorBase()
```

Deep Tensor constructor without initialization.

```
inline virtual ~DeepTensorBase()
```

```
DeepTensorBase(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope
    = "")
```

Deep Tensor constructor with initialization..

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] Name scopes of operations.

```
virtual void init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope
                 = "") = 0
```

Initialize the Deep Tensor.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] Name scopes of operations.

```
virtual double cutoff() const = 0
```

Get the cutoff radius.

Returns

The cutoff radius.

```
virtual int numb_types() const = 0
```

Get the number of types.

Returns

The number of types.

```
virtual int output_dim() const = 0
```

Get the output dimension.

Returns

The output dimension.

```
virtual const std::vector<int> &sel_types() const = 0
```

Get the list of sel types.

Returns

The list of sel types.

```
virtual void get_type_map(std::string &type_map) = 0
```

Get the type map (element name of the atom types) of this model.

Parameters

**type\_map** – [out] The type map of this model.

## Class DeepTensorTF

- Defined in file\_source\_api\_cc\_include\_DeepTensorTF.h

## Inheritance Relationships

### Base Type

- public deepmd::DeepTensorBase (Class DeepTensorBase)

## Class Documentation

class **DeepTensorTF** : public deepmd::DeepTensorBase

Deep Tensor.

### Unnamed Group

```
virtual void computew(std::vector<double> &global_tensor, std::vector<double> &force,
                    std::vector<double> &virial, std::vector<double> &atom_tensor,
                    std::vector<double> &atom_virial, const std::vector<double> &coord, const
                    std::vector<int> &atype, const std::vector<double> &box, const bool
                    request_deriv)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.
- **force** – [out] The component-wise force of the global tensor, size odim x natoms x 3.
- **virial** – [out] The component-wise virial of the global tensor, size odim x 9.
- **atom\_tensor** – [out] The atomic tensor value of the model, size natoms x odim.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size odim x natoms x 9.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9.
- **request\_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

```
virtual void computew(std::vector<float> &global_tensor, std::vector<float> &force,
                    std::vector<float> &virial, std::vector<float> &atom_tensor,
                    std::vector<float> &atom_virial, const std::vector<float> &coord, const
                    std::vector<int> &atype, const std::vector<float> &box, const bool
                    request_deriv)
```

### Unnamed Group

```
virtual void computew(std::vector<double> &global_tensor, std::vector<double> &force,
                    std::vector<double> &virial, std::vector<double> &atom_tensor,
                    std::vector<double> &atom_virial, const std::vector<double> &coord, const
                    std::vector<int> &atype, const std::vector<double> &box, const int nghost,
                    const InputNlist &inlist, const bool request_deriv)
```

Evaluate the global tensor and component-wise force and virial.

Parameters

- **global\_tensor** – [out] The global tensor to evaluate.

- **force** – [out] The component-wise force of the global tensor, size `odim x natoms x 3`.
- **virial** – [out] The component-wise virial of the global tensor, size `odim x 9`.
- **atom\_tensor** – [out] The atomic tensor value of the model, size `natoms x odim`.
- **atom\_virial** – [out] The component-wise atomic virial of the global tensor, size `odim x natoms x 9`.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The list should contain `natoms` ints.
- **box** – [in] The cell of the region. The array should be of size `9`.
- **nghost** – [in] The number of ghost atoms.
- **inlist** – [in] The input neighbour list.
- **request\_deriv** – [in] Whether to request the derivative of the global tensor, including force and virial.

```
virtual void computew(std::vector<float> &global_tensor, std::vector<float> &force,
                    std::vector<float> &virial, std::vector<float> &atom_tensor,
                    std::vector<float> &atom_virial, const std::vector<float> &coord, const
                    std::vector<int> &atype, const std::vector<float> &box, const int nghost,
                    const InputNlist &inlist, const bool request_deriv)
```

## Public Functions

**DeepTensorTF()**

Deep Tensor constructor without initialization.

**~DeepTensorTF()**

**DeepTensorTF**(const std::string &model, const int &gpu\_rank = 0, const std::string &name\_scope =  
 "")

Deep Tensor constructor with initialization..

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] Name scopes of operations.

```
virtual void init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope =
  "")
```

Initialize the Deep Tensor.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] Name scopes of operations.

```
inline virtual double cutoff() const
    Get the cutoff radius.
    Returns
        The cutoff radius.
inline virtual int numb_types() const
    Get the number of types.
    Returns
        The number of types.
inline virtual int output_dim() const
    Get the output dimension.
    Returns
        The output dimension.
inline virtual const std::vector<int> &sel_types() const
    Get the list of sel types.
    Returns
        The list of sel types.
virtual void get_type_map(std::string &type_map)
    Get the type map (element name of the atom types) of this model.
    Parameters
        type_map – [out] The type map of this model.
```

### Class **DipoleChargeModifier**

- Defined in file\_source\_api\_cc\_include\_DataModifier.h

### Class Documentation

```
class DipoleChargeModifier
    Dipole charge modifier.
```

#### Public Functions

```
DipoleChargeModifier()
    Dipole charge modifier without initialization.
DipoleChargeModifier(const std::string &model, const int &gpu_rank = 0, const std::string
    &name_scope = "")
    Dipole charge modifier without initialization.
    Parameters
        • model – [in] The name of the frozen model file.
        • gpu_rank – [in] The GPU rank. Default is 0.
        • name_scope – [in] The name scope.
```



`~DipoleChargeModifier()`

void **init**(const std::string &model, const int &gpu\_rank = 0, const std::string &name\_scope = "")  
Initialize the dipole charge modifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] The name scope.

void **print\_summary**(const std::string &pre) const  
Print the DP summary to the screen.

Parameters

- **pre** – [in] The prefix to each line.

template<typename VALUETYPE>

void **compute**(std::vector<VALUETYPE> &dfcorr\_, std::vector<VALUETYPE> &dvcorr\_, const  
std::vector<VALUETYPE> &dcoord\_, const std::vector<int> &dtype\_, const  
std::vector<VALUETYPE> &dbox, const std::vector<std::pair<int, int>> &pairs,  
const std::vector<VALUETYPE> &delef\_, const int nghost, const [InputNlist](#)  
&imp\_list)

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr\_** – [out] The force correction on each atom.
- **dvcorr\_** – [out] The virial correction.
- **dcoord\_** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **dtype\_** – [in] The atom types. The list should contain natoms ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **delef\_** – [in] The electric field on each atom. The array should be of size natoms x 3.
- **nghost** – [in] The number of ghost atoms.
- **imp\_list** – [in] The neighbor list.

double **cutoff**() const

Get cutoff radius.

Returns

double cutoff radius.

int **numb\_types**() const

Get the number of atom types.

Returns

int number of atom types.

const std::vector<int> &**sel\_types**() const

Get the list of sel types.

Returns  
The list of sel types.

## Class `DipoleChargeModifierBase`

- Defined in file `_source_api_cc_include_DataModifier.h`

## Inheritance Relationships

## Derived Type

- `public deepmd::DipoleChargeModifierTF` (Class `DipoleChargeModifierTF`)

## Class Documentation

class `DipoleChargeModifierBase`

Dipole charge modifier. (Base class)

Subclassed by `deepmd::DipoleChargeModifierTF`

## Unnamed Group

```
virtual void computew(std::vector<double> &dfcorr_, std::vector<double> &dvcorr_, const  
    std::vector<double> &dcoord_, const std::vector<int> &dtype_, const  
    std::vector<double> &dbox, const std::vector<std::pair<int, int>> &pairs,  
    const std::vector<double> &delef_, const int nghost, const InputNlist  
    &lmplist) = 0
```

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr\_** – [out] The force correction on each atom.
- **dvcorr\_** – [out] The virial correction.
- **dcoord\_** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **dtype\_** – [in] The atom types. The list should contain `natoms` ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain `npairs` pairs of ints.
- **delef\_** – [in] The electric field on each atom. The array should be of size `natoms x 3`.
- **nghost** – [in] The number of ghost atoms.
- **lmplist** – [in] The neighbor list.

```
virtual void computew(std::vector<float> &dfcorr_, std::vector<float> &dvcorr_, const  
    std::vector<float> &dcoord_, const std::vector<int> &dtype_, const  
    std::vector<float> &dbox, const std::vector<std::pair<int, int>> &pairs,  
    const std::vector<float> &delef_, const int nghost, const InputNlist  
    &lmplist) = 0
```

## Public Functions

`inline DipoleChargeModifierBase()`

Dipole charge modifier without initialization.

`DipoleChargeModifierBase(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

Dipole charge modifier without initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] The name scope.

`inline virtual ~DipoleChargeModifierBase()`

`virtual void init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "") = 0`

Initialize the dipole charge modifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] The name scope.

`virtual double cutoff() const = 0`

Get cutoff radius.

Returns

double cutoff radius.

`virtual int numb_types() const = 0`

Get the number of atom types.

Returns

int number of atom types.

`virtual const std::vector<int> &sel_types() const = 0`

Get the list of sel types.

Returns

The list of sel types.

## Class DipoleChargeModifierTF

- Defined in file `_source_api_cc_include_DataModifierTF.h`

## Inheritance Relationships

### Base Type

- `public deepmd::DipoleChargeModifierBase` ([Class DipoleChargeModifierBase](#))

## Class Documentation

`class DipoleChargeModifierTF : public deepmd::DipoleChargeModifierBase`

Dipole charge modifier.

### Public Functions

`DipoleChargeModifierTF()`

Dipole charge modifier without initialization.

`DipoleChargeModifierTF(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

Dipole charge modifier without initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] The name scope.

`~DipoleChargeModifierTF()`

`virtual void init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

Initialize the dipole charge modifier.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank. Default is 0.
- **name\_scope** – [in] The name scope.

`inline virtual double cutoff() const`

Get cutoff radius.

Returns

double cutoff radius.

```
inline virtual int numb_types() const
```

Get the number of atom types.

Returns

int number of atom types.

```
inline virtual const std::vector<int> &sel_types() const
```

Get the list of sel types.

Returns

The list of sel types.

```
virtual void computew(std::vector<double> &dfcorr_, std::vector<double> &dvcorr_, const  
std::vector<double> &dcoord_, const std::vector<int> &dtype_, const  
std::vector<double> &dbox, const std::vector<std::pair<int, int>> &pairs,  
const std::vector<double> &delef_, const int nghost, const InputNlist  
&lmp_list)
```

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr\_** – [out] The force correction on each atom.
- **dvcorr\_** – [out] The virial correction.
- **dcoord\_** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **dtype\_** – [in] The atom types. The list should contain natoms ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **delef\_** – [in] The electric field on each atom. The array should be of size natoms x 3.
- **nghost** – [in] The number of ghost atoms.
- **lmp\_list** – [in] The neighbor list.

```
virtual void computew(std::vector<float> &dfcorr_, std::vector<float> &dvcorr_, const  
std::vector<float> &dcoord_, const std::vector<int> &dtype_, const  
std::vector<float> &dbox, const std::vector<std::pair<int, int>> &pairs,  
const std::vector<float> &delef_, const int nghost, const InputNlist  
&lmp_list)
```

### 20.3.3 Enums

#### Enum DPBackend

- Defined in `file_source_api_cc_include_common.h`

## Enum Documentation

enum deepmd : DPBackend

Values:

enumerator TensorFlow

enumerator PyTorch

enumerator Paddle

enumerator Unknown

## 20.3.4 Functions

### Function deepmd::check\_status

- Defined in file\_source\_api\_cc\_include\_common.h

### Function Documentation

void deepmd::check\_status(const tensorflow::Status &status)

Check TensorFlow status. Exit if not OK.

Parameters

**status** – [in] TensorFlow status.

### Function deepmd::convert\_pbtxt\_to\_pb

- Defined in file\_source\_api\_cc\_include\_common.h

### Function Documentation

void deepmd::convert\_pbtxt\_to\_pb(std::string fn\_pb\_txt, std::string fn\_pb)

Convert pbtxt to pb.

Parameters

- **fn\_pb\_txt** – [in] Filename of the pb txt file.
- **fn\_pb** – [in] Filename of the pb file.

**Function `deepmd::get_env_nthreads`**

- Defined in file `_source_api_cc_include_common.h`

**Function Documentation**

`void deepmd::get_env_nthreads(int &num_intra_nthreads, int &num_inter_nthreads)`

Get the number of threads from the environment variable.

A warning will be thrown if environmental variables are not set.

Parameters

- **`num_intra_nthreads`** – [out] The number of intra threads. Read from `TF_INTRA_OP_PARALLELISM_THREADS`.
- **`num_inter_nthreads`** – [out] The number of inter threads. Read from `TF_INTER_OP_PARALLELISM_THREADS`.

**Function `deepmd::load_op_library`**

- Defined in file `_source_api_cc_include_common.h`

**Function Documentation**

`void deepmd::load_op_library()`

Dynamically load OP library. This should be called before loading graphs.

**Function `deepmd::model_compatible`**

- Defined in file `_source_api_cc_include_common.h`

**Function Documentation**

`bool deepmd::model_compatible(std::string &model_version)`

Check if the model version is supported.

Parameters

- **`model_version`** – [in] The model version.

Returns

Whether the model is supported (true or false).

### Function `deepmd::name_prefix`

- Defined in `file_source_api_cc_include_common.h`

### Function Documentation

`std::string deepmd::name_prefix(const std::string &name_scope)`

### Function `deepmd::print_summary`

- Defined in `file_source_api_cc_include_common.h`

### Function Documentation

`void deepmd::print_summary(const std::string &pre)`

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

**pre** – [in] The prefix to each line.

### Function `deepmd::read_file_to_string`

- Defined in `file_source_api_cc_include_common.h`

### Function Documentation

`void deepmd::read_file_to_string(std::string model, std::string &file_content)`

Read model file to a string.

Parameters

- **model** – [in] Path to the model.
- **file\_content** – [out] Content of the model file.

### Template Function `deepmd::select_by_type`

- Defined in `file_source_api_cc_include_common.h`

### Function Documentation

`template<typename VALUETYPE>`

`void deepmd::select_by_type(std::vector<int> &fwd_map, std::vector<int> &bkwd_map, int  
                            &nghost_real, const std::vector<VALUETYPE> &dcoord_, const  
                            std::vector<int> &dtype_, const int &nghost, const std::vector<int>  
                            &sel_type_)`

Get forward and backward map of selected atoms by atom types.

Parameters



- **fwd\_map** – [out] The forward map with size `natoms`.
- **bkw\_map** – [out] The backward map with size `nreal`.
- **nghost\_real** – [out] The number of selected ghost atoms.
- **dcoord\_** – [in] The coordinates of all atoms. Reserved for compatibility.
- **dtype\_** – [in] The atom types of all atoms.
- **nghost** – [in] The number of ghost atoms.
- **sel\_type\_** – [in] The selected atom types.

**Template Function** `deepmd::select_map(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&, const int&, const int&, const int&)`

- Defined in `file_source_api_cc_include_common.h`

## Function Documentation

```
template<typename VT>
void deepmd::select_map(std::vector<VT> &out, const std::vector<VT> &in, const std::vector<int>
                        &fwd_map, const int &stride, const int &nframes = 1, const int &nall1 = 0,
                        const int &nall2 = 0)
```

Apply the given map to a vector.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd\_map** – [in] The map.
- **stride** – [in] The stride of the input vector.
- **nframes** – [in] The number of frames.
- **nall1** – [in] The number of atoms in the input vector.
- **nall2** – [in] The number of atoms in the output vector.

**Template Function** `deepmd::select_map(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&, const int&, const int&, const int&)`

- Defined in `file_source_api_cc_include_common.h`

## Function Documentation

```
template<typename VT>
void deepmd::select_map(typename std::vector<VT>::iterator out, const typename
                        std::vector<VT>::const_iterator in, const std::vector<int> &fwd_map, const
                        int &stride, const int &nframes = 1, const int &nall1 = 0, const int &nall2 = 0)
```

Apply the given map to a vector.

Parameters

- **out** – [out] The output vector.
- **in** – [in] The input vector.
- **fwd\_map** – [in] The map.
- **stride** – [in] The stride of the input vector.
- **nframes** – [in] The number of frames.
- **nall1** – [in] The number of atoms in the input vector.
- **nall2** – [in] The number of atoms in the output vector.

**Template Function** `deepmd::select_map_inv(std::vector<VT>&, const std::vector<VT>&, const std::vector<int>&, const int&)`

- Defined in `file_source_api_cc_include_common.h`

## Function Documentation

```
template<typename VT>
void deepmd::select_map_inv(std::vector<VT> &out, const std::vector<VT> &in, const
                           std::vector<int> &fwd_map, const int &stride)
```

**Template Function** `deepmd::select_map_inv(typename std::vector<VT>::iterator, const typename std::vector<VT>::const_iterator, const std::vector<int>&, const int&)`

- Defined in `file_source_api_cc_include_common.h`

## Function Documentation

```
template<typename VT>
void deepmd::select_map_inv(typename std::vector<VT>::iterator out, const typename
                           std::vector<VT>::const_iterator in, const std::vector<int> &fwd_map,
                           const int &stride)
```

### Template Function `deepmd::select_real_atoms`

- Defined in `file_source_api_cc_include_common.h`

#### Function Documentation

```
template<typename VALUETYPE>
void deepmd::select_real_atoms(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int
                               &nghost_real, const std::vector<VALUETYPE> &dcoord_, const
                               std::vector<int> &dtype_, const int &nghost, const int &ntypes)
```

### Template Function `deepmd::select_real_atoms_coord`

- Defined in `file_source_api_cc_include_common.h`

#### Function Documentation

```
template<typename VALUETYPE>
void deepmd::select_real_atoms_coord(std::vector<VALUETYPE> &dcoord, std::vector<int>
                                     &dtype, std::vector<VALUETYPE> &aparam, int
                                     &nghost_real, std::vector<int> &fwd_map, std::vector<int>
                                     &bkw_map, int &nall_real, int &nloc_real, const
                                     std::vector<VALUETYPE> &dcoord_, const std::vector<int>
                                     &dtype_, const std::vector<VALUETYPE> &aparam_, const
                                     int &nghost, const int &ntypes, const int &nframes, const int
                                     &daparam, const int &nall, const bool aparam_nall = false)
```

### Function `deepmd::session_get_dtype`

- Defined in `file_source_api_cc_include_common.h`

#### Function Documentation

```
int deepmd::session_get_dtype(tensorflow::Session *session, const std::string name, const std::string
                              scope = "")
```

Get the type of a tensor.

Parameters

- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Returns

The type of the tensor as int.

### Template Function `deepmd::session_get_scalar`

- Defined in `file_source_api_cc_include_common.h`

#### Function Documentation

```
template<typename VT>
VT deepmd::session_get_scalar(tensorflow::Session *session, const std::string name, const std::string
                             scope = "")
```

Get the value of a tensor.

Parameters

- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

Returns

The value of the tensor.

### Template Function `deepmd::session_get_vector`

- Defined in `file_source_api_cc_include_common.h`

#### Function Documentation

```
template<typename VT>
void deepmd::session_get_vector(std::vector<VT> &o_vec, tensorflow::Session *session, const
                                std::string name_, const std::string scope = "")
```

Get the vector of a tensor.

Parameters

- **o\_vec** – [out] The output vector.
- **session** – [in] TensorFlow session.
- **name** – [in] The name of the tensor.
- **scope** – [in] The scope of the tensor.

**Template Function `deepmd::session_input_tensors`**(`std::vector<std::pair<std::string, tensorflow::Tensor>>&`, `const std::vector<VALUETYPE>&`, `const int&`, `const std::vector<int>&`, `const std::vector<VALUETYPE>&`, `const double&`, `const std::vector<VALUETYPE>&`, `const std::vector<VALUETYPE>&`, `const deepmd::AtomMap&`, `const std::string`, `const bool`)

- Defined in `file_source_api_cc_include_common.h`

## Function Documentation

```
template<typename MODELTYPE, typename VALUETYPE>
int deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>
                                &input_tensors, const std::vector<VALUETYPE> &dcoord_, const
                                int &ntypes, const std::vector<int> &dtype_, const
                                std::vector<VALUETYPE> &dbox, const double &cell_size, const
                                std::vector<VALUETYPE> &fparam_, const
                                std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap
                                &atommap, const std::string scope = "", const bool aparam_null =
                                false)
```

Get input tensors.

Parameters

- **input\_tensors** – [out] Input tensors.
- **dcoord\_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **dtype\_** – [in] Atom types.
- **dbox** – [in] Box matrix.
- **cell\_size** – [in] Cell size.
- **fparam\_** – [in] Frame parameters.
- **aparam\_** – [in] Atom parameters.
- **atommap** – [in] Atom map.
- **scope** – [in] The scope of the tensors.
- **aparam\_null** – [in] Whether the atomic dimesion of atomic parameters is null.

**Template Function** `deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>&, const std::vector<VALUETYPE>&, const int&, const std::vector<int>&, const std::vector<VALUETYPE>&, InputNlist&, const std::vector<VALUETYPE>&, const std::vector<VALUETYPE>&, const deepmd::AtomMap&, const int, const int, const std::string, const bool)`

- Defined in file\_source\_api\_cc\_include\_common.h

## Function Documentation

```
template<typename MODELTYPE, typename VALUETYPE>
int deepmd::session_input_tensors(std::vector<std::pair<std::string, tensorflow::Tensor>>
                                &input_tensors, const std::vector<VALUETYPE> &dcoord_, const
                                int &ntypes, const std::vector<int> &dtype_, const
                                std::vector<VALUETYPE> &dbox, InputNlist &dlist, const
                                std::vector<VALUETYPE> &fparam_, const
                                std::vector<VALUETYPE> &aparam_, const deepmd::AtomMap
                                &atommap, const int nghost, const int ago, const std::string scope =
                                "", const bool aparam_null = false)
```

Get input tensors.

## Parameters

- **input\_tensors** – [out] Input tensors.
- **dcoord\_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **dtype\_** – [in] Atom types.
- **dlist** – [in] Neighbor list.
- **fparam\_** – [in] Frame parameters.
- **aparam\_** – [in] Atom parameters.
- **atommap** – [in] Atom map.
- **nghost** – [in] Number of ghost atoms.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **scope** – [in] The scope of the tensors.
- **aparam\_nall** – [in] Whether the atomic dimension of atomic parameters is null.

**Template Function `deepmd::session_input_tensors_mixed_type`**

- Defined in `file_source_api_cc_include_common.h`

**Function Documentation**

```
template<typename MODELTYPE, typename VALUETYPE>
int deepmd::session_input_tensors_mixed_type(std::vector<std::pair<std::string,
    tensorflow::Tensor>> &input_tensors, const int
    &nframes, const std::vector<VALUETYPE>
    &dcoord_, const int &ntypes, const std::vector<int>
    &dtype_, const std::vector<VALUETYPE> &dbox,
    const double &cell_size, const
    std::vector<VALUETYPE> &fparam_, const
    std::vector<VALUETYPE> &aparam_, const
    deepmd::AtomMap &atommap, const std::string scope
    = "", const bool aparam_nall = false)
```

Get input tensors for mixed type.

## Parameters

- **input\_tensors** – [out] Input tensors.
- **nframes** – [in] Number of frames.
- **dcoord\_** – [in] Coordinates of atoms.
- **ntypes** – [in] Number of atom types.
- **dtype\_** – [in] Atom types.
- **dlist** – [in] Neighbor list.
- **fparam\_** – [in] Frame parameters.
- **aparam\_** – [in] Atom parameters.

- **atommap** – [in] Atom map.
- **nghost** – [in] Number of ghost atoms.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **scope** – [in] The scope of the tensors.
- **aparam\_nall** – [in] Whether the atomic dimesion of atomic parameters is nall.

### 20.3.5 Typedefs

#### Typedef deepmd::ENERGYTYPE

- Defined in file\_source\_api\_cc\_include\_common.h

#### Typedef Documentation

```
typedef double deepmd::ENERGYTYPE
```

#### Typedef deepmd::STRINGTYPE

- Defined in file\_source\_api\_cc\_include\_tf\_private.h

#### Typedef Documentation

```
typedef std::string deepmd::STRINGTYPE
```





## 21.1 Class Hierarchy

- Namespace deepmd
  - Namespace deepmd::hpp
    - \* Struct deepmd\_exception
    - \* Struct InputNlist
    - \* Class DeepPot
    - \* Class DeepPotModelDevi
    - \* Class DeepTensor
    - \* Class DipoleChargeModifier
- Struct DP\_DeepPot
- Struct DP\_DeepPotModelDevi
- Struct DP\_DeepTensor
- Struct DP\_DipoleChargeModifier
- Struct DP\_Nlist

## 21.2 File Hierarchy

- dir\_source
  - dir\_source\_api\_c
    - \* dir\_source\_api\_c\_include
      - file\_source\_api\_c\_include\_c\_api.h
      - file\_source\_api\_c\_include\_c\_api\_internal.h
      - file\_source\_api\_c\_include\_deepmd.hpp

## 21.3 Full API

### 21.3.1 Namespaces

#### Namespace `deepmd`

##### Contents

- [Namespaces](#)

#### Namespaces

- [Namespace `deepmd::hpp`](#)

#### Namespace `deepmd::hpp`

##### Contents

- [Classes](#)
- [Functions](#)

#### Classes

- [Struct `deepmd\_exception`](#)
- [Struct `InputNlist`](#)
- [Class `DeepPot`](#)
- [Class `DeepPotModelDevi`](#)
- [Class `DeepTensor`](#)
- [Class `DipoleChargeModifier`](#)

#### Functions

- [Function `deepmd::hpp::convert\_nlist`](#)
- [Function `deepmd::hpp::convert\_pbtxt\_to\_pb`](#)
- [Function `deepmd::hpp::read\_file\_to\_string`](#)
- [Template Function `deepmd::hpp::select\_by\_type`](#)
- [Template Function `deepmd::hpp::select\_map`](#)

Namespace `std`

### 21.3.2 Classes and Structs

#### Struct `deepmd_exception`

- Defined in `file_source_api_c_include_deepmd.hpp`

#### Inheritance Relationships

#### Base Type

- `public std::runtime_error`

#### Struct Documentation

struct `deepmd_exception` : `public std::runtime_error`  
 General DeePMD-kit exception. Throw if anything doesn't work.

#### Public Functions

```
inline deepmd_exception()
inline deepmd_exception(const std::string &msg)
```

#### Struct `InputNlist`

- Defined in `file_source_api_c_include_deepmd.hpp`

#### Struct Documentation

struct `InputNlist`  
 Neighbor list.

#### Public Functions

```
inline InputNlist()
inline InputNlist(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)
inline ~InputNlist()
```

### Public Members

`DP_Nlist *nl`

C API neighbor list.

`int inum`

Number of core region atoms.

`int *ilist`

Array stores the core region atom's index.

`int *numneigh`

Array stores the core region atom's neighbor atom number.

`int **firstneigh`

Array stores the core region atom's neighbor index.

### Struct DP\_DeepPot

- Defined in `file_source_api_c_include_c_api_internal.h`

### Struct Documentation

`struct DP_DeepPot`

### Public Functions

`DP_DeepPot()`

`DP_DeepPot(deepmd::DeepPot &dp)`

### Public Members

`deepmd::DeepPot dp`

`std::string exception`

`int dfparam`

`int daparam`

`bool aparam_nall`

**Struct DP\_DeepPotModelDevi**

- Defined in file `_source_api_c_include_c_api_internal.h`

**Struct Documentation**

```
struct DP_DeepPotModelDevi
```

**Public Functions**

```
DP_DeepPotModelDevi()
```

```
DP_DeepPotModelDevi(deepmd::DeepPotModelDevi &dp)
```

**Public Members**

```
deepmd::DeepPotModelDevi dp
```

```
std::string exception
```

```
int dfparam
```

```
int daparam
```

```
bool aparam_nall
```

**Struct DP\_DeepTensor**

- Defined in file `_source_api_c_include_c_api_internal.h`

**Struct Documentation**

```
struct DP_DeepTensor
```

**Public Functions**

```
DP_DeepTensor()
```

```
DP_DeepTensor(deepmd::DeepTensor &dt)
```

### Public Members

deepmd::DeepTensor **dt**

std::string **exception**

### Struct DP\_DipoleChargeModifier

- Defined in file\_source\_api\_c\_include\_c\_api\_internal.h

### Struct Documentation

struct DP\_DipoleChargeModifier

### Public Functions

DP\_DipoleChargeModifier()

DP\_DipoleChargeModifier(deepmd::DipoleChargeModifier &dcm)

### Public Members

deepmd::DipoleChargeModifier **dcm**

std::string **exception**

### Struct DP\_Nlist

- Defined in file\_source\_api\_c\_include\_c\_api\_internal.h

### Struct Documentation

struct DP\_Nlist

## Public Functions

`DP_Nlist()`

`DP_Nlist(deepmd::InputNlist &nl)`

## Public Members

`deepmd::InputNlist nl`

`std::string exception`

## Class DeepPot

- Defined in file `_source_api_c_include_deepmd.hpp`

## Class Documentation

class **DeepPot**

Deep Potential.

## Public Functions

`inline DeepPot()`

DP constructor without initialization.

`inline ~DeepPot()`

`inline DeepPot(const std::string &model, const int &gpu_rank = 0, const std::string &file_content =  
 "")`

DP constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank.
- **file\_content** – [in] The content of the frozen model file.

`inline void init(const std::string &model, const int &gpu_rank = 0, const std::string &file_content =  
 "")`

Initialize the DP.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank.
- **file\_content** – [in] The content of the frozen model file.

`template<typename VALUETYPE, typename ENERGYVTYPE>`

```
inline void compute(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force,
                  std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord,
                  const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
                  std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const
                  std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force and virial by using this DP.

Warning: Natoms should not be zero when computing multiple frames.

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE, typename ENERGYVTYPE>
inline void compute(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force,
                  std::vector<VALUETYPE> &virial, std::vector<VALUETYPE>
                  &atom_energy, std::vector<VALUETYPE> &atom_virial, const
                  std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const
                  std::vector<VALUETYPE> &box, const std::vector<VALUETYPE> &fparam
                  = std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam =
                  std::vector<VALUETYPE>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP.

Warning: Natoms should not be zero when computing multiple frames.

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.



- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE, typename ENERGYVTYPE>
inline void compute(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force,
    std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE> &coord,
    const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
    int nghost, const InputNlist &imp_list, const int &ago, const
    std::vector<VALUETYPE> &fparam = std::vector<VALUETYPE>(), const
    std::vector<VALUETYPE> &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force and virial by using this DP with the neighbor list.

Warning: Natoms should not be zero when computing multiple frames.

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **ngghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE, typename ENERGYVTYPE>
```

```
inline void compute(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force,
    std::vector<VALUETYPE> &virial, std::vector<VALUETYPE>
    &atom_energy, std::vector<VALUETYPE> &atom_virial, const
    std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const
    std::vector<VALUETYPE> &box, const int nghost, const InputNlist &imp_list,
    const int &ago, const std::vector<VALUETYPE> &fparam =
    std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam =
    std::vector<VALUETYPE>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP with the neighbor list.

Warning: Natoms should not be zero when computing multiple frames.

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE, typename ENERGYVTYPE>
inline void compute_mixed_type(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force,
    std::vector<VALUETYPE> &virial, const int &nframes, const
    std::vector<VALUETYPE> &coord, const std::vector<int>
    &atype, const std::vector<VALUETYPE> &box, const
    std::vector<VALUETYPE> &fparam =
    std::vector<VALUETYPE>(), const std::vector<VALUETYPE>
    &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force and virial by using this DP with the mixed type.

#### Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.
- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

```
template<typename VALUETYPE, typename ENERGYVTYPE>
inline void compute_mixed_type(ENERGYVTYPE &ener, std::vector<VALUETYPE> &force,
                               std::vector<VALUETYPE> &virial, std::vector<VALUETYPE>
                               &atom_energy, std::vector<VALUETYPE> &atom_virial, const
                               int &nframes, const std::vector<VALUETYPE> &coord, const
                               std::vector<int> &atype, const std::vector<VALUETYPE> &box,
                               const std::vector<VALUETYPE> &fparam =
                               std::vector<VALUETYPE>(), const std::vector<VALUETYPE>
                               &aparam = std::vector<VALUETYPE>())
```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP with the mixed type.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **nframes** – [in] The number of frames.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **fparam** – [in] The frame parameter. The array can be of size : nframes x dim\_fparam. dim\_fparam. Then all frames are assumed to be provided with the same fparam.

- **aparam** – [in] The atomic parameter The array can be of size : nframes x natoms x dim\_aparam. natoms x dim\_aparam. Then all frames are assumed to be provided with the same aparam.

inline double **cutoff**() const

Get the cutoff radius.

Returns

The cutoff radius.

inline int **numb\_types**() const

Get the number of types.

Returns

The number of types.

inline int **numb\_types\_spin**() const

Get the number of types with spin.

Returns

The number of types with spin.

inline void **get\_type\_map**(std::string &type\_map)

Get the type map (element name of the atom types) of this model.

Parameters

**type\_map** – [out] The type map of this model.

inline void **print\_summary**(const std::string &pre) const

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

**pre** – [in] The prefix to each line.

inline int **dim\_fparam**() const

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

inline int **dim\_aparam**() const

Get the dimension of the atomic parameter.

Returns

The dimension of the atomic parameter.

## Class DeepPotModelDevi

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

## Class Documentation

class **DeepPotModelDevi**

Deep Potential model deviation.

### Public Functions

inline **DeepPotModelDevi**()

DP model deviation constructor without initialization.

inline **~DeepPotModelDevi**()

inline **DeepPotModelDevi**(const std::vector<std::string> &models)

DP model deviation constructor with initialization.

Parameters

**models** – [in] The names of the frozen model file.

inline void **init**(const std::vector<std::string> &models, const int &gpu\_rank = 0, const std::vector<std::string> &file\_content = std::vector<std::string>())

Initialize the DP model deviation.

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The GPU rank.
- **file\_content** – [in] The content of the frozen model file.

template<typename **VALUETYPE**>

inline void **compute**(std::vector<double> &ener, std::vector<std::vector<**VALUETYPE**>> &force, std::vector<std::vector<**VALUETYPE**>> &virial, const std::vector<**VALUETYPE**> &coord, const std::vector<int> &atype, const std::vector<**VALUETYPE**> &box, const int nghost, const **InputNlist** &imp\_list, const int &ago, const std::vector<**VALUETYPE**> &fparam = std::vector<**VALUETYPE**>(), const std::vector<**VALUETYPE**> &aparam = std::vector<**VALUETYPE**>())

Evaluate the energy, force and virial by using this DP model deviation.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename **VALUETYPE**>

```

inline void compute(std::vector<double> &ener, std::vector<std::vector<VALUETYPE>> &force,
    std::vector<std::vector<VALUETYPE>> &virial,
    std::vector<std::vector<VALUETYPE>> &atom_energy,
    std::vector<std::vector<VALUETYPE>> &atom_virial, const
    std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const
    std::vector<VALUETYPE> &box, const int nghost, const InputNlist &imp_list,
    const int &ago, const std::vector<VALUETYPE> &fparam =
    std::vector<VALUETYPE>(), const std::vector<VALUETYPE> &aparam =
    std::vector<VALUETYPE>())

```

Evaluate the energy, force, virial, atomic energy, and atomic virial by using this DP model deviation.

Parameters

- **ener** – [out] The system energy.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_energy** – [out] The atomic energy.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

```
inline double cutoff() const
```

Get the cutoff radius.

Returns

The cutoff radius.

```
inline int numb_types() const
```

Get the number of types.

Returns

The number of types.

```
inline int numb_types_spin() const
```

Get the number of types with spin.

Returns

The number of types with spin.

```
inline int dim_fparam() const
```

Get the dimension of the frame parameter.

Returns

The dimension of the frame parameter.

```
inline int dim_aparam() const
```

Get the dimension of the atomic parameter.

Returns

The dimension of the atomic parameter.

```
template<typename VALUETYPE>
inline void compute_avg(std::vector<VALUETYPE> &avg, const
                        std::vector<std::vector<VALUETYPE>> &xx)
```

Compute the average of vectors.

Parameters

- **avg** – [out] The average of vectors.
- **xx** – [in] The vectors of all models.

```
template<typename VALUETYPE>
inline void compute_std(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE> &avg,
                        const std::vector<std::vector<VALUETYPE>> &xx, const int &stride)
```

Compute the standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **xx** – [in] The vectors of all models.
- **stride** – [in] The stride to compute the deviation.

```
template<typename VALUETYPE>
inline void compute_relative_std(std::vector<VALUETYPE> &std, const
                                std::vector<VALUETYPE> &avg, const VALUETYPE eps,
                                const int &stride)
```

Compute the relative standard deviation of vectors.

Parameters

- **std** – [out] The standard deviation of vectors.
- **avg** – [in] The average of vectors.
- **eps** – [in] The level parameter for computing the deviation.
- **stride** – [in] The stride to compute the deviation.

```
template<typename VALUETYPE>
inline void compute_std_f(std::vector<VALUETYPE> &std, const std::vector<VALUETYPE>
                           &avg, const std::vector<std::vector<VALUETYPE>> &xx)
```

Compute the standard deviation of forces.

Parameters

- **std** – [out] The standard deviation of forces.
- **avg** – [in] The average of forces.
- **xx** – [in] The vectors of all forces.

```
template<typename VALUETYPE>
inline void compute_relative_std_f(std::vector<VALUETYPE> &std, const
                                    std::vector<VALUETYPE> &avg, const VALUETYPE eps)
```

Compute the relative standard deviation of forces.

Parameters

- **std** – [out] The relative standard deviation of forces.

- **avg** – [in] The relative average of forces.
- **eps** – [in] The level parameter for computing the deviation.

## Class `DeepTensor`

- Defined in `file_source_api_c_include_deepmd.hpp`

## Class Documentation

class `DeepTensor`

Deep Tensor.

### Public Functions

inline `DeepTensor()`

Deep Tensor constructor without initialization.

inline `~DeepTensor()`

inline `DeepTensor(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

`DeepTensor` constructor with initialization.

Parameters

**model** – [in] The name of the frozen model file.

inline void `init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")`

Initialize the `DeepTensor`.

Parameters

**model** – [in] The name of the frozen model file.

template<typename `VALUETYPE`>

inline void `compute(std::vector<VALUETYPE> &tensor, const std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const std::vector<VALUETYPE> &box)`

Evaluate the tensor, force and virial by using this Deep Tensor.

Parameters

- **tensor** – [out] The atomic tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

template<typename `VALUETYPE`>



```
inline void compute(std::vector<VALUETYPE> &tensor, const std::vector<VALUETYPE> &coord,
                  const std::vector<int> &atype, const std::vector<VALUETYPE> &box, const
                  int nghost, const InputNlist &imp_list)
```

Evaluate the tensor, force and virial by using this Deep Tensor with the neighbor list.

Parameters

- **tensor** – [out] The tensor.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

```
template<typename VALUETYPE>
inline void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE>
                  &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE>
                  &coord, const std::vector<int> &atype, const std::vector<VALUETYPE>
                  &box)
```

Evaluate the global tensor, force and virial by using this Deep Tensor.

Parameters

- **global\_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

```
template<typename VALUETYPE>
inline void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE>
                  &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE>
                  &atom_tensor, std::vector<VALUETYPE> &atom_virial, const
                  std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const
                  std::vector<VALUETYPE> &box)
```

Evaluate the global tensor, force, virial, atomic tensor, and atomic virial by using this Deep Tensor.

Parameters

- **global\_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_tensor** – [out] The atomic tensor.
- **atom\_virial** – [out] The atomic virial.

- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).

```
template<typename VALUETYPE>
inline void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE>
                  &force, std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE>
                  &coord, const std::vector<int> &atype, const std::vector<VALUETYPE>
                  &box, const int nghost, const InputNlist &imp_list)
```

Evaluate the global tensor, force and virial by using this Deep Tensor with the neighbor list.

Parameters

- **global\_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

```
template<typename VALUETYPE>
inline void compute(std::vector<VALUETYPE> &global_tensor, std::vector<VALUETYPE>
                  &force, std::vector<VALUETYPE> &virial, std::vector<VALUETYPE>
                  &atom_tensor, std::vector<VALUETYPE> &atom_virial, const
                  std::vector<VALUETYPE> &coord, const std::vector<int> &atype, const
                  std::vector<VALUETYPE> &box, const int nghost, const InputNlist &imp_list)
```

Evaluate the global tensor, force, virial, atomic tensor, and atomic virial by using this Deep Tensor with the neighbor list.

Parameters

- **global\_tensor** – [out] The global tensor.
- **force** – [out] The force on each atom.
- **virial** – [out] The virial.
- **atom\_tensor** – [out] The atomic tensor.
- **atom\_virial** – [out] The atomic virial.
- **coord** – [in] The coordinates of atoms. The array should be of size nframes x natoms x 3.
- **atype** – [in] The atom types. The list should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size nframes x 9 (PBC) or empty (no PBC).
- **nghost** – [in] The number of ghost atoms.

- **nlist** – [in] The neighbor list.

```
inline double cutoff() const
```

Get the cutoff radius.

Returns

The cutoff radius.

```
inline int numb_types() const
```

Get the number of types.

Returns

The number of types.

```
inline int output_dim() const
```

Get the output dimension.

Returns

The output dimension.

```
inline std::vector<int> sel_types() const
```

```
inline void print_summary(const std::string &pre) const
```

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

**pre** – [in] The prefix to each line.

```
inline void get_type_map(std::string &type_map)
```

Get the type map (element name of the atom types) of this model.

Parameters

**type\_map** – [out] The type map of this model.

### Class DipoleChargeModifier

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

### Class Documentation

```
class DipoleChargeModifier
```

#### Public Functions

```
inline DipoleChargeModifier()
```

[DipoleChargeModifier](#) constructor without initialization.

```
inline ~DipoleChargeModifier()
```

```
inline DipoleChargeModifier(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope = "")
```

[DipoleChargeModifier](#) constructor with initialization.

Parameters

- **model** – [in] The name of the frozen model file.

- **gpu\_rank** – [in] The rank of the GPU to be used.
- **name\_scope** – [in] The name scope of the model.

```
inline void init(const std::string &model, const int &gpu_rank = 0, const std::string &name_scope =
    "")
```

Initialize the [DipoleChargeModifier](#).

Parameters

- **model** – [in] The name of the frozen model file.
- **gpu\_rank** – [in] The rank of the GPU to be used.
- **name\_scope** – [in] The name scope of the model.

```
template<typename VALUETYPE>
```

```
inline void compute(std::vector<VALUETYPE> &dfcorr_, std::vector<VALUETYPE> &dvcorr_,
    const std::vector<VALUETYPE> &dcoord_, const std::vector<int> &dtype_,
    const std::vector<VALUETYPE> &dbox, const std::vector<std::pair<int,
    int>> &pairs, const std::vector<VALUETYPE> &delef_, const int nghost, const
    InputNlist &lmp_list)
```

Evaluate the force and virial correction by using this dipole charge modifier.

Parameters

- **dfcorr\_** – [out] The force correction on each atom.
- **dvcorr\_** – [out] The virial correction.
- **dcoord\_** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **dtype\_** – [in] The atom types. The list should contain natoms ints.
- **dbox** – [in] The cell of the region. The array should be of size 9.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **delef\_** – [in] The electric field on each atom. The array should be of size natoms x 3.
- **nghost** – [in] The number of ghost atoms.
- **lmp\_list** – [in] The neighbor list.

```
inline double cutoff() const
```

Get the cutoff radius.

Returns

The cutoff radius.

```
inline int numb_types() const
```

Get the number of types.

Returns

The number of types.

```
inline std::vector<int> sel_types() const
```

```
inline void print_summary(const std::string &pre) const
```

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

**pre** – [in] The prefix to each line.

### 21.3.3 Functions

#### Template Function `_DP_DeepPotCompute`

- Defined in `file_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepPotCompute(DP_DeepPot *dp, const int nframes, const int natom, const FPTYPE
                               *coord, const int *atype, const FPTYPE *cell, const FPTYPE *fparam,
                               const FPTYPE *aparam, double *energy, FPTYPE *force, FPTYPE
                               *virial, FPTYPE *atomic_energy, FPTYPE *atomic_virial)
```

#### Specialized Template Function `_DP_DeepPotCompute< double >`

- Defined in `file_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<>
inline void _DP_DeepPotCompute<double>(DP_DeepPot *dp, const int nframes, const int natom, const
                                         double *coord, const int *atype, const double *cell, const double
                                         *fparam, const double *aparam, double *energy, double *force,
                                         double *virial, double *atomic_energy, double *atomic_virial)
```

#### Specialized Template Function `_DP_DeepPotCompute< float >`

- Defined in `file_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<>
inline void _DP_DeepPotCompute<float>(DP_DeepPot *dp, const int nframes, const int natom, const float
                                       *coord, const int *atype, const float *cell, const float *fparam,
                                       const float *aparam, double *energy, float *force, float *virial,
                                       float *atomic_energy, float *atomic_virial)
```

### Template Function `_DP_DeepPotComputeMixedType`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepPotComputeMixedType(DP_DeepPot *dp, const int nframes, const int natom, const
                                         FPTYPE *coord, const int *atype, const FPTYPE *cell, const
                                         FPTYPE *fparam, const FPTYPE *aparam, double *energy,
                                         FPTYPE *force, FPTYPE *virial, FPTYPE *atomic_energy,
                                         FPTYPE *atomic_virial)
```

### Specialized Template Function `_DP_DeepPotComputeMixedType< double >`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<>
inline void _DP_DeepPotComputeMixedType<double>(DP_DeepPot *dp, const int nframes, const int
                                                  natom, const double *coord, const int *atype, const
                                                  double *cell, const double *fparam, const double
                                                  *aparam, double *energy, double *force, double
                                                  *virial, double *atomic_energy, double
                                                  *atomic_virial)
```

### Specialized Template Function `_DP_DeepPotComputeMixedType< float >`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<>
inline void _DP_DeepPotComputeMixedType<float>(DP_DeepPot *dp, const int nframes, const int natom,
                                                const float *coord, const int *atype, const float *cell,
                                                const float *fparam, const float *aparam, double
                                                *energy, float *force, float *virial, float
                                                *atomic_energy, float *atomic_virial)
```

## Template Function `_DP_DeepPotComputeNList`

- Defined in file `_source_api_c_include_deepmd.hpp`

### Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepPotComputeNList(DP_DeepPot *dp, const int nframes, const int natom, const
    FPTYPE *coord, const int *atype, const FPTYPE *cell, const int
    nghost, const DP_Nlist *nlist, const int ago, const FPTYPE
    *fparam, const FPTYPE *aparam, double *energy, FPTYPE
    *force, FPTYPE *virial, FPTYPE *atomic_energy, FPTYPE
    *atomic_virial)
```

## Specialized Template Function `_DP_DeepPotComputeNList< double >`

- Defined in file `_source_api_c_include_deepmd.hpp`

### Function Documentation

```
template<>
inline void _DP_DeepPotComputeNList<double>(DP_DeepPot *dp, const int nframes, const int natom,
    const double *coord, const int *atype, const double *cell,
    const int nghost, const DP_Nlist *nlist, const int ago,
    const double *fparam, const double *aparam, double
    *energy, double *force, double *virial, double
    *atomic_energy, double *atomic_virial)
```

## Specialized Template Function `_DP_DeepPotComputeNList< float >`

- Defined in file `_source_api_c_include_deepmd.hpp`

### Function Documentation

```
template<>
inline void _DP_DeepPotComputeNList<float>(DP_DeepPot *dp, const int nframes, const int natom, const
    float *coord, const int *atype, const float *cell, const int
    nghost, const DP_Nlist *nlist, const int ago, const float
    *fparam, const float *aparam, double *energy, float *force,
    float *virial, float *atomic_energy, float *atomic_virial)
```

### Template Function `_DP_DeepPotModelDeviComputeNList`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepPotModelDeviComputeNList(DP_DeepPotModelDevi *dp, const int natom, const
                                             FPTYPE *coord, const int *atype, const FPTYPE *cell,
                                             const int nghost, const DP_Nlist *nlist, const int ago,
                                             const FPTYPE *fparam, const FPTYPE *aparam,
                                             double *energy, FPTYPE *force, FPTYPE *virial,
                                             FPTYPE *atomic_energy, FPTYPE *atomic_virial)
```

### Specialized Template Function `_DP_DeepPotModelDeviComputeNList< double >`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<>
inline void _DP_DeepPotModelDeviComputeNList<double>(DP_DeepPotModelDevi *dp, const int natom,
                                                       const double *coord, const int *atype, const
                                                       double *cell, const int nghost, const DP_Nlist
                                                       *nlist, const int ago, const double *fparam,
                                                       const double *aparam, double *energy, double
                                                       *force, double *virial, double *atomic_energy,
                                                       double *atomic_virial)
```

### Specialized Template Function `_DP_DeepPotModelDeviComputeNList< float >`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<>
inline void _DP_DeepPotModelDeviComputeNList<float>(DP_DeepPotModelDevi *dp, const int natom,
                                                       const float *coord, const int *atype, const float
                                                       *cell, const int nghost, const DP_Nlist *nlist,
                                                       const int ago, const float *fparam, const float
                                                       *aparam, double *energy, float *force, float
                                                       *virial, float *atomic_energy, float
                                                       *atomic_virial)
```



## Template Function `_DP_DeepTensorCompute`

- Defined in file `_source_api_c_include_deepmd.hpp`

### Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepTensorCompute(DP_DeepTensor *dt, const int natom, const FPTYPE *coord, const int
                                *atype, const FPTYPE *cell, FPTYPE *global_tensor, FPTYPE
                                *force, FPTYPE *virial, FPTYPE **atomic_energy, FPTYPE
                                *atomic_virial, int *size_at)
```

## Specialized Template Function `_DP_DeepTensorCompute< double >`

- Defined in file `_source_api_c_include_deepmd.hpp`

### Function Documentation

```
template<>
inline void _DP_DeepTensorCompute<double>(DP_DeepTensor *dt, const int natom, const double *coord,
                                           const int *atype, const double *cell, double *global_tensor,
                                           double *force, double *virial, double **atomic_tensor,
                                           double *atomic_virial, int *size_at)
```

## Specialized Template Function `_DP_DeepTensorCompute< float >`

- Defined in file `_source_api_c_include_deepmd.hpp`

### Function Documentation

```
template<>
inline void _DP_DeepTensorCompute<float>(DP_DeepTensor *dt, const int natom, const float *coord,
                                           const int *atype, const float *cell, float *global_tensor, float
                                           *force, float *virial, float **atomic_tensor, float
                                           *atomic_virial, int *size_at)
```

## Template Function `_DP_DeepTensorComputeNList`

- Defined in file `_source_api_c_include_deepmd.hpp`

## Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepTensorComputeNList(DP_DeepTensor *dt, const int natom, const FPTYPE *coord,
                                       const int *atype, const FPTYPE *cell, const int nghost, const
                                       DP_Nlist *nlist, FPTYPE *global_tensor, FPTYPE *force,
                                       FPTYPE *virial, FPTYPE **atomic_energy, FPTYPE
                                       *atomic_virial, int *size_at)
```

### Specialized Template Function `_DP_DeepTensorComputeNList< double >`

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

## Function Documentation

```
template<>
inline void _DP_DeepTensorComputeNList<double>(DP_DeepTensor *dt, const int natom, const double
                                                *coord, const int *atype, const double *cell, const int
                                                nghost, const DP_Nlist *nlist, double *global_tensor,
                                                double *force, double *virial, double
                                                **atomic_tensor, double *atomic_virial, int *size_at)
```

### Specialized Template Function `_DP_DeepTensorComputeNList< float >`

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

## Function Documentation

```
template<>
inline void _DP_DeepTensorComputeNList<float>(DP_DeepTensor *dt, const int natom, const float
                                                *coord, const int *atype, const float *cell, const int
                                                nghost, const DP_Nlist *nlist, float *global_tensor, float
                                                *force, float *virial, float **atomic_tensor, float
                                                *atomic_virial, int *size_at)
```

### Template Function `_DP_DeepTensorComputeTensor`

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

## Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepTensorComputeTensor(DP_DeepTensor *dt, const int natom, const FPTYPE *coord,
                                         const int *atype, const FPTYPE *cell, FPTYPE **tensor, int
                                         *size)
```

### Specialized Template Function \_DP\_DeepTensorComputeTensor< double >

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

## Function Documentation

```
template<>
inline void _DP_DeepTensorComputeTensor<double>(DP_DeepTensor *dt, const int natom, const double
                                                  *coord, const int *atype, const double *cell, double
                                                  **tensor, int *size)
```

### Specialized Template Function \_DP\_DeepTensorComputeTensor< float >

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

## Function Documentation

```
template<>
inline void _DP_DeepTensorComputeTensor<float>(DP_DeepTensor *dt, const int natom, const float
                                                  *coord, const int *atype, const float *cell, float
                                                  **tensor, int *size)
```

### Template Function \_DP\_DeepTensorComputeTensorNList

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

## Function Documentation

```
template<typename FPTYPE>
inline void _DP_DeepTensorComputeTensorNList(DP_DeepTensor *dt, const int natom, const FPTYPE
                                              *coord, const int *atype, const FPTYPE *cell, const int
                                              nghost, const DP_Nlist *nlist, FPTYPE **tensor, int
                                              *size)
```

**Specialized Template Function `_DP_DeepTensorComputeTensorNList< double >`**

- Defined in file `_source_api_c_include_deepmd.hpp`

**Function Documentation**

```
template<>
inline void _DP_DeepTensorComputeTensorNList<double>(DP_DeepTensor *dt, const int natom, const
double *coord, const int *atype, const double
*cell, const int nghost, const DP_Nlist *nlist,
double **tensor, int *size)
```

**Specialized Template Function `_DP_DeepTensorComputeTensorNList< float >`**

- Defined in file `_source_api_c_include_deepmd.hpp`

**Function Documentation**

```
template<>
inline void _DP_DeepTensorComputeTensorNList<float>(DP_DeepTensor *dt, const int natom, const float
*coord, const int *atype, const float *cell, const
int nghost, const DP_Nlist *nlist, float **tensor,
int *size)
```

**Template Function `_DP_DipoleChargeModifierComputeNList`**

- Defined in file `_source_api_c_include_deepmd.hpp`

**Function Documentation**

```
template<typename FPTYPE>
inline void _DP_DipoleChargeModifierComputeNList(DP_DipoleChargeModifier *dcm, const int natom,
const FPTYPE *coord, const int *atype, const
FPTYPE *cell, const int *pairs, const int npairs,
const FPTYPE *delef_, const int nghost, const
DP_Nlist *nlist, FPTYPE *dfcorr_, FPTYPE
*dvcorr_)
```

**Specialized Template Function `_DP_DipoleChargeModifierComputeNList< double >`**

- Defined in file `_source_api_c_include_deepmd.hpp`

**Function Documentation**

```
template<>
inline void _DP_DipoleChargeModifierComputeNList<double>(DP_DipoleChargeModifier *dcm, const
                                                         int natom, const double *coord, const int
                                                         *atype, const double *cell, const int
                                                         *pairs, const int npairs, const double
                                                         *delef_, const int nghost, const DP_Nlist
                                                         *nlist, double *dfcorr_, double *dvcorr_)
```

**Specialized Template Function `_DP_DipoleChargeModifierComputeNList< float >`**

- Defined in file `_source_api_c_include_deepmd.hpp`

**Function Documentation**

```
template<>
inline void _DP_DipoleChargeModifierComputeNList<float>(DP_DipoleChargeModifier *dcm, const int
                                                         natom, const float *coord, const int *atype,
                                                         const float *cell, const int *pairs, const int
                                                         npairs, const float *delef_, const int nghost,
                                                         const DP_Nlist *nlist, float *dfcorr_, float
                                                         *dvcorr_)
```

**Function `_DP_Get_Energy_Pointer(std::vector<double>&, const int)`**

- Defined in file `_source_api_c_include_deepmd.hpp`

**Function Documentation**

```
inline double *_DP_Get_Energy_Pointer(std::vector<double> &vec, const int nframes)
```

**Function `_DP_Get_Energy_Pointer(double&, const int)`**

- Defined in file `_source_api_c_include_deepmd.hpp`

## Function Documentation

inline double \*\_DP\_Get\_Energy\_Pointer(double &vec, const int nframes)

## Function `deepmd::hpp::convert_nlist`

- Defined in `file_source_api_c_include_deepmd.hpp`

## Function Documentation

inline void deepmd::hpp::convert\_nlist([InputNlist](#) &to\_nlist, std::vector<std::vector<int>>> &from\_nlist)

Convert int vector to [InputNlist](#).

Parameters

- `to_nlist` – [out] [InputNlist](#).
- `from_nlist` – [in] 2D int vector. The first axis represents the central atoms and the second axis represents the neighbor atoms.

## Function `deepmd::hpp::convert_pbtxt_to_pb`

- Defined in `file_source_api_c_include_deepmd.hpp`

## Function Documentation

inline void deepmd::hpp::convert\_pbtxt\_to\_pb(std::string fn\_pb\_txt, std::string fn\_pb)

Convert pbtxt to pb.

Parameters

- `fn_pb_txt` – [in] Filename of the pb txt file.
- `fn_pb` – [in] Filename of the pb file.

## Function `deepmd::hpp::read_file_to_string`

- Defined in `file_source_api_c_include_deepmd.hpp`

## Function Documentation

inline void deepmd::hpp::read\_file\_to\_string(std::string model, std::string &file\_content)

Read model file to a string.

Parameters

- `model` – [in] Path to the model.
- `file_content` – [out] Content of the model file.

### Template Function `deepmd::hpp::select_by_type`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<typename VALUETYPE>
void deepmd::hpp::select_by_type(std::vector<int> &fwd_map, std::vector<int> &bkw_map, int
                                &nghost_real, const std::vector<VALUETYPE> &dcoord_, const
                                std::vector<int> &dtype_, const int &nghost, const
                                std::vector<int> &sel_type_)
```

Get forward and backward map of selected atoms by atom types.

Parameters

- `fwd_map` – [out] The forward map with size `natoms`.
- `bkw_map` – [out] The backward map with size `nreal`.
- `nghost_real` – [out] The number of selected ghost atoms.
- `dcoord_` – [in] The coordinates of all atoms. Reserved for compatibility.
- `dtype_` – [in] The atom types of all atoms.
- `nghost` – [in] The number of ghost atoms.
- `sel_type_` – [in] The selected atom types.

### Template Function `deepmd::hpp::select_map`

- Defined in file `_source_api_c_include_deepmd.hpp`

#### Function Documentation

```
template<typename VT>
void deepmd::hpp::select_map(std::vector<VT> &out, const std::vector<VT> &in, const
                             std::vector<int> &fwd_map, const int &stride)
```

Apply the given map to a vector. Assume `nframes` is 1.

Template Parameters

`VT` – The value type of the vector. Only support `int`.

Parameters

- `out` – [out] The output vector.
- `in` – [in] The input vector.
- `fwd_map` – [in] The map.
- `stride` – [in] The stride of the input vector.

### Function DP\_ConvertPbtxtToPb

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

void DP\_ConvertPbtxtToPb(const char \*c\_pbtxt, const char \*c\_pb)

Convert Pbtxt to PB.

Parameters

- **c\_pbtxt** – [in] The name of the Pbtxt file.
- **c\_pb** – [in] The name of the PB file.

### Function DP\_DeepPotCheckOK

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

const char \*DP\_DeepPotCheckOK(DP\_DeepPot \*dp)

Check if there is any exceptions throw.

Parameters

**dp** – The DP to use.

Returns

const char\* error message.

### Function DP\_DeepPotCompute

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

void DP\_DeepPotCompute(DP\_DeepPot \*dp, const int natom, const double \*coord, const int \*atype, const double \*cell, double \*energy, double \*force, double \*virial, double \*atomic\_energy, double \*atomic\_virial)

Evaluate the energy, force and virial by using a DP. (double version)

Attention

The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters



- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

### Function DP\_DeepPotCompute2

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepPotCompute2(DP_DeepPot *dp, const int nframes, const int natom, const double *coord, const
                        int *atype, const double *cell, const double *fparam, const double *aparam,
                        double *energy, double *force, double *virial, double *atomic_energy, double
                        *atomic_virial)
```

Evaluate the energy, force and virial by using a DP. (double version)

Version  
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim\_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim\_aparam.

- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

### Function DP\_DeepPotComputeF

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepPotComputeF(DP_DeepPot *dp, const int natom, const float *coord, const int *atype, const
                        float *cell, double *energy, float *force, float *virial, float *atomic_energy, float
                        *atomic_virial)
```

Evaluate the energy, force and virial by using a DP. (float version)

Attention

The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

## Function DP\_DeepPotComputeF2

- Defined in file `_source_api_c_include_c_api.h`

### Function Documentation

```
void DP_DeepPotComputeF2(DP_DeepPot *dp, const int nframes, const int natom, const float *coord, const
                        int *atype, const float *cell, const float *fparam, const float *aparam, double
                        *energy, float *force, float *virial, float *atomic_energy, float *atomic_virial)
```

Evaluate the energy, force and virial by using a DP. (float version)

Version  
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The array should contain `natoms` ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size `nframes x dim_fparam`.
- **aparam** – [in] The atom parameters. The array can be of size `nframes x dim_aparam`.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size `natoms x 3`.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size `natoms`.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size `natoms x 9`.

## Function DP\_DeepPotComputeMixedType

- Defined in file `_source_api_c_include_c_api.h`

### Function Documentation

```
void DP_DeepPotComputeMixedType(DP_DeepPot *dp, const int nframes, const int natoms, const double
                                *coord, const int *atype, const double *cell, const double *fparam,
                                const double *aparam, double *energy, double *force, double *virial,
                                double *atomic_energy, double *atomic_virial)
```

Evaluate the energy, force and virial by using a DP with the mixed type. (double version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The array should contain `nframes x natoms` ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size `nframes x dim_fparam`.
- **aparam** – [in] The atom parameters. The array can be of size `nframes x dim_aparam`.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size `natoms x 3`.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size `natoms`.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size `natoms x 9`.

## Function DP\_DeepPotComputeMixedTypef

- Defined in file `_source_api_c_include_c_api.h`

## Function Documentation

```
void DP_DeepPotComputeMixedTypef (DP_DeepPot *dp, const int nframes, const int natoms, const float
                                *coord, const int *atype, const float *cell, const float *fparam, const
                                float *aparam, double *energy, float *force, float *virial, float
                                *atomic_energy, float *atomic_virial)
```

Evaluate the energy, force and virial by using a DP with the mixed type. (float version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

### Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain nframes x natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim\_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim\_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

## Function DP\_DeepPotComputeNList

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

```
void DP_DeepPotComputeNList (DP_DeepPot *dp, const int natom, const double *coord, const int *atype,
                             const double *cell, const int nghost, const DP_Nlist *nlist, const int ago,
                             double *energy, double *force, double *virial, double *atomic_energy,
                             double *atomic_virial)
```

Evaluate the energy, force and virial by using a DP with the neighbor list. (double version)

### Attention

The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

#### Function DP\_DeepPotComputeNList2

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

```
void DP_DeepPotComputeNList2(DP_DeepPot *dp, const int nframes, const int natom, const double
                             *coord, const int *atype, const double *cell, const int nghost, const
                             DP_Nlist *nlist, const int ago, const double *fparam, const double
                             *aparam, double *energy, double *force, double *virial, double
                             *atomic_energy, double *atomic_virial)
```

Evaluate the energy, force and virial by using a DP with the neighbor list. (double version)

Version  
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP to use.

- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim\_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim\_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

### Function DP\_DeepPotComputeNListf

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepPotComputeNListf(DP_DeepPot *dp, const int natom, const float *coord, const int *atype,
                             const float *cell, const int nghost, const DP_Nlist *nlist, const int ago,
                             double *energy, float *force, float *virial, float *atomic_energy, float
                             *atomic_virial)
```

Evaluate the energy, force and virial by using a DP with the neighbor list. (float version)

#### Attention

The number of frames is assumed to be 1.

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.

- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

### Function DP\_DeepPotComputeNListf2

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepPotComputeNListf2(DP_DeepPot *dp, const int nframes, const int natom, const float *coord,
                             const int *atype, const float *cell, const int nghost, const DP_Nlist *nlist,
                             const int ago, const float *fparam, const float *aparam, double *energy,
                             float *force, float *virial, float *atomic_energy, float *atomic_virial)
```

Evaluate the energy, force and virial by using a DP with the neighbor list. (float version)

Version  
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP to use.
- **nframes** – [in] The number of frames.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.



- **fparam** – [in] The frame parameters. The array can be of size nframes x dim\_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x dim\_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

### Function DP\_DeepPotGetCutoff

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

double DP\_DeepPotGetCutoff(DP\_DeepPot \*dp)

Get the type map of a DP.

Parameters

**dp** – [in] The DP to use.

Returns

The cutoff radius.

### Function DP\_DeepPotGetDimAParam

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

int DP\_DeepPotGetDimAParam(DP\_DeepPot \*dp)

Get the dimension of atomic parameters of a DP.

Parameters

**dp** – [in] The DP to use.

Returns

The dimension of atomic parameters of the DP.

### Function DP\_DeepPotGetDimFParam

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

`int DP_DeepPotGetDimFParam(DP_DeepPot *dp)`

Get the dimension of frame parameters of a DP.

Parameters

**dp** – [in] The DP to use.

Returns

The dimension of frame parameters of the DP.

### Function DP\_DeepPotGetNumbTypes

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

`int DP_DeepPotGetNumbTypes(DP_DeepPot *dp)`

Get the number of types of a DP.

Parameters

**dp** – [in] The DP to use.

Returns

The number of types of the DP.

### Function DP\_DeepPotGetNumbTypesSpin

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

`int DP_DeepPotGetNumbTypesSpin(DP_DeepPot *dp)`

Get the number of types with spin of a DP.

Parameters

**dp** – [in] The DP to use.

Returns

The number of types with spin of the DP.

### Function DP\_DeepPotGetTypeMap

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

const char \*DP\_DeepPotGetTypeMap(DP\_DeepPot \*dp)

Get the type map of a DP.

Parameters

**dp** – [in] The DP to use.

Returns

The type map of the DP.

## Function DP\_DeepPotIsAParamNall

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

bool DP\_DeepPotIsAParamNall(DP\_DeepPot \*dp)

Check whether the atomic dimension of atomic parameters is nall instead of nloc.

Parameters

**dp** – [in] The DP to use.

Returns

true the atomic dimension of atomic parameters is nall

Returns

false the atomic dimension of atomic parameters is nloc

## Function DP\_DeepPotModelDeviCheckOK

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

const char \*DP\_DeepPotModelDeviCheckOK(DP\_DeepPotModelDevi \*dp)

Check if there is any exceptions throw.

Parameters

**dp** – The DP model deviation to use.

Returns

const char\* error message.

## Function DP\_DeepPotModelDeviComputeNList

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepPotModelDeviComputeNList(DP_DeepPotModelDevi *dp, const int natom, const double
                                     *coord, const int *atype, const double *cell, const int nghost,
                                     const DP_Nlist *nlist, const int ago, double *energy, double
                                     *force, double *virial, double *atomic_energy, double
                                     *atomic_virial)
```

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP model deviation to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

## Function DP\_DeepPotModelDeviComputeNList2

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

```
void DP_DeepPotModelDeviComputeNList2(DP_DeepPotModelDevi *dp, const int nframes, const int
                                     natoms, const double *coord, const int *atype, const double
                                     *cell, const int nghost, const DP_Nlist *nlist, const int ago,
                                     const double *fparam, const double *aparam, double *energy,
                                     double *force, double *virial, double *atomic_energy, double
                                     *atomic_virial)
```

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (double version)

Version  
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

### Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim\_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim\_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

## Function DP\_DeepPotModelDeviComputeNListf

- Defined in file `_source_api_c_include_c_api.h`

### Function Documentation

```
void DP_DeepPotModelDeviComputeNListf(DP_DeepPotModelDevi *dp, const int natom, const float
                                       *coord, const int *atype, const float *cell, const int nghost,
                                       const DP_Nlist *nlist, const int ago, double *energy, float
                                       *force, float *virial, float *atomic_energy, float *atomic_virial)
```

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dp** – [in] The DP model deviation to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The array should contain `natoms` ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size `natoms x 3`.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size `natoms`.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size `natoms x 9`.

## Function DP\_DeepPotModelDeviComputeNListf2

- Defined in file `_source_api_c_include_c_api.h`

## Function Documentation

```
void DP_DeepPotModelDeviComputeNListf2(DP_DeepPotModelDevi *dp, const int nframes, const int
    natoms, const float *coord, const int *atype, const float *cell,
    const int nghost, const DP_Nlist *nlist, const int ago, const
    float *fparam, const float *aparam, double *energy, float
    *force, float *virial, float *atomic_energy, float
    *atomic_virial)
```

Evaluate the energy, force and virial by using a DP model deviation with neighbor list. (float version)

Version  
2

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

### Parameters

- **dp** – [in] The DP model deviation to use.
- **nframes** – [in] The number of frames. Only support 1 for now.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **ago** – [in] Update the internal neighbour list if ago is 0.
- **fparam** – [in] The frame parameters. The array can be of size nframes x dim\_fparam.
- **aparam** – [in] The atom parameters. The array can be of size nframes x natoms x dim\_aparam.
- **energy** – [out] Output energy.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_energy** – [out] Output atomic energy. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.

**Function DP\_DeepPotModelDeviGetCutoff**

- Defined in file `_source_api_c_include_c_api.h`

**Function Documentation**

double DP\_DeepPotModelDeviGetCutoff(DP\_DeepPotModelDevi \*dp)

Get the type map of a DP model deviation.

Parameters

**dp** – [in] The DP model deviation to use.

Returns

The cutoff radius.

**Function DP\_DeepPotModelDeviGetDimAParam**

- Defined in file `_source_api_c_include_c_api.h`

**Function Documentation**

int DP\_DeepPotModelDeviGetDimAParam(DP\_DeepPotModelDevi \*dp)

Get the dimension of atomic parameters of a DP Model Deviation.

Parameters

**dp** – [in] The DP Model Deviation to use.

Returns

The dimension of atomic parameters of the DP Model Deviation.

**Function DP\_DeepPotModelDeviGetDimFParam**

- Defined in file `_source_api_c_include_c_api.h`

**Function Documentation**

int DP\_DeepPotModelDeviGetDimFParam(DP\_DeepPotModelDevi \*dp)

Get the dimension of frame parameters of a DP Model Deviation.

Parameters

**dp** – [in] The DP Model Deviation to use.

Returns

The dimension of frame parameters of the DP Model Deviation.



### Function DP\_DeepPotModelDeviGetNumbTypes

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

int DP\_DeepPotModelDeviGetNumbTypes(DP\_DeepPotModelDevi \*dp)

Get the number of types of a DP model deviation.

Parameters

**dp** – [in] The DP model deviation to use.

Returns

The number of types of the DP model deviation.

### Function DP\_DeepPotModelDeviGetNumbTypesSpin

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

int DP\_DeepPotModelDeviGetNumbTypesSpin(DP\_DeepPotModelDevi \*dp)

Get the number of types with spin of a DP model deviation.

Parameters

**dp** – [in] The DP model deviation to use.

Returns

The number of types with spin of the DP model deviation.

### Function DP\_DeepPotModelDeviIsAParamNall

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

bool DP\_DeepPotModelDeviIsAParamNall(DP\_DeepPotModelDevi \*dp)

Check whether the atomic dimension of atomic parameters is nall instead of nloc.

Parameters

**dp** – [in] The DP Model Deviation to use.

Returns

true the atomic dimension of atomic parameters is nall

Returns

false the atomic dimension of atomic parameters is nloc

## Function DP\_DeepTensorCheckOK

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
const char *DP_DeepTensorCheckOK(DP_DeepTensor *dt)
```

Check if there is any exceptions throw.

Parameters

**dt** – The Deep Tensor to use.

Returns

const char\* error message.

## Function DP\_DeepTensorCompute

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepTensorCompute(DP_DeepTensor *dt, const int natom, const double *coord, const int *atype,
                           const double *cell, double *global_tensor, double *force, double *virial,
                           double **atomic_tensor, double *atomic_virial, int *size_at)
```

Evaluate the global tensor, force and virial by using a DP. (double version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **global\_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size\_at** – [out] Output size of atomic tensor.

## Function DP\_DeepTensorComputeF

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepTensorComputeF(DP_DeepTensor *dt, const int natom, const float *coord, const int *atype,
                           const float *cell, float *global_tensor, float *force, float *virial, float
                           **atomic_tensor, float *atomic_virial, int *size_at)
```

Evaluate the global tensor, force and virial by using a DP. (float version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **global\_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size\_at** – [out] Output size of atomic tensor.

## Function DP\_DeepTensorComputeNList

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepTensorComputeNList(DP_DeepTensor *dt, const int natom, const double *coord, const int
                               *atype, const double *cell, const int nghost, const DP_Nlist *nlist,
                               double *global_tensor, double *force, double *virial, double
                               **atomic_tensor, double *atomic_virial, int *size_at)
```

Evaluate the global tensor, force and virial by using a DP with the neighbor list. (double version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

## Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **global\_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size\_at** – [out] Output size of atomic tensor.

**Function DP\_DeepTensorComputeNListf**

- Defined in file\_source\_api\_c\_include\_c\_api.h

**Function Documentation**

```
void DP_DeepTensorComputeNListf(DP_DeepTensor *dt, const int natom, const float *coord, const int
                                *atype, const float *cell, const int nghost, const DP_Nlist *nlist, float
                                *global_tensor, float *force, float *virial, float **atomic_tensor, float
                                *atomic_virial, int *size_at)
```

Evaluate the global tensor, force and virial by using a DP with the neighbor list. (float version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

## Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.

- **global\_tensor** – [out] Output global tensor.
- **force** – [out] Output force. The array should be of size natoms x 3.
- **virial** – [out] Output virial. The array should be of size 9.
- **atomic\_tensor** – [out] Output atomic tensor. The array should be of size natoms.
- **atomic\_virial** – [out] Output atomic virial. The array should be of size natoms x 9.
- **size\_at** – [out] Output size of atomic tensor.

### Function DP\_DeepTensorComputeTensor

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepTensorComputeTensor(DP_DeepTensor *dt, const int natom, const double *coord, const int
                                *atype, const double *cell, double **tensor, int *size)
```

Evaluate the tensor by using a DP. (double version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **tensor** – [out] Output tensor.

### Function DP\_DeepTensorComputeTensorf

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

```
void DP_DeepTensorComputeTensorf(DP_DeepTensor *dt, const int natom, const float *coord, const int
                                  *atype, const float *cell, float **tensor, int *size)
```

Evaluate the tensor by using a DP. (float version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.

- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

### Function `DP_DeepTensorComputeTensorNList`

- Defined in `file_source_api_c_include_c_api.h`

#### Function Documentation

```
void DP_DeepTensorComputeTensorNList(DP_DeepTensor *dt, const int natom, const double *coord,
                                     const int *atype, const double *cell, const int nghost, const
                                     DP_Nlist *nlist, double **tensor, int *size)
```

Evaluate the tensor by using a DP with the neighbor list. (double version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The array should contain `natoms ints`.
- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

### Function `DP_DeepTensorComputeTensorNListf`

- Defined in `file_source_api_c_include_c_api.h`

#### Function Documentation

```
void DP_DeepTensorComputeTensorNListf(DP_DeepTensor *dt, const int natom, const float *coord, const
                                       int *atype, const float *cell, const int nghost, const DP_Nlist
                                       *nlist, float **tensor, int *size)
```

Evaluate the tensor by using a DP with the neighbor list. (float version)

Parameters

- **dt** – [in] The Deep Tensor to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The array should contain `natoms ints`.

- **box** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **tensor** – [out] Output tensor.
- **size** – [out] Output size of the tensor.

### Function DP\_DeepTensorGetCutoff

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

double DP\_DeepTensorGetCutoff(DP\_DeepTensor \*dt)

Get the type map of a Deep Tensor.

Parameters

**dt** – [in] The Deep Tensor to use.

Returns

The cutoff radius.

### Function DP\_DeepTensorGetNumbSelTypes

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

int DP\_DeepTensorGetNumbSelTypes(DP\_DeepTensor \*dt)

Get the number of sel types of a Deep Tensor.

Parameters

**dt** – [in] The Deep Tensor to use.

Returns

The number of sel types

### Function DP\_DeepTensorGetNumbTypes

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

`int DP_DeepTensorGetNumbTypes(DP_DeepTensor *dt)`

Get the type map of a Deep Tensor.

Parameters

`dt` – [in] The Deep Tensor to use.

Returns

The number of types of the Deep Tensor.

### Function DP\_DeepTensorGetOutputDim

- Defined in `file_source_api_c_include_c_api.h`

### Function Documentation

`int DP_DeepTensorGetOutputDim(DP_DeepTensor *dt)`

Get the output dimension of a Deep Tensor.

Parameters

`dt` – [in] The Deep Tensor to use.

Returns

The output dimension of the Deep Tensor.

### Function DP\_DeepTensorGetSelTypes

- Defined in `file_source_api_c_include_c_api.h`

### Function Documentation

`int *DP_DeepTensorGetSelTypes(DP_DeepTensor *dt)`

Get sel types of a Deep Tensor.

Parameters

`dt` – [in] The Deep Tensor to use.

Returns

The sel types

### Function DP\_DeepTensorGetTypeMap

- Defined in `file_source_api_c_include_c_api.h`



## Function Documentation

`const char *DP_DeepTensorGetTypeMap(DP_DeepTensor *dt)`

Get the type map of a Deep Tensor.

Parameters

**dt** – [in] The Deep Tensor to use.

Returns

The type map of the Deep Tensor.

## Function DP\_DeleteChar

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

`void DP_DeleteChar(const char *c_str)`

Destroy a char array.

Parameters

**c\_str** – The char array.

## Function DP\_DeleteDeepPot

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

`void DP_DeleteDeepPot(DP_DeepPot *dp)`

Delete a Deep Potential.

Parameters

**dp** – Deep Potential to delete.

## Function DP\_DeleteDeepPotModelDevi

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

`void DP_DeleteDeepPotModelDevi(DP_DeepPotModelDevi *dp)`

Delete a Deep Potential Model Deviation.

Parameters

**dp** – Deep Potential to delete.

**Function DP\_DeleteDeepTensor**

- Defined in file\_source\_api\_c\_include\_c\_api.h

**Function Documentation**

void **DP\_DeleteDeepTensor**(DP\_DeepTensor \*dt)

Delete a Deep Tensor.

Parameters

**dp** – Deep Tensor to delete.

**Function DP\_DeleteDipoleChargeModifier**

- Defined in file\_source\_api\_c\_include\_c\_api.h

**Function Documentation**

void **DP\_DeleteDipoleChargeModifier**(DP\_DipoleChargeModifier \*dcm)

Delete a Dipole Charge Modifier.

Parameters

**dp** – Dipole Charge Modifier to delete.

**Function DP\_DeleteNlist**

- Defined in file\_source\_api\_c\_include\_c\_api.h

**Function Documentation**

void **DP\_DeleteNlist**(DP\_Nlist \*nl)

Delete a neighbor list.

Parameters

**nl** – Neighbor list to delete.

**Function DP\_DipoleChargeModifierCheckOK**

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

`const char *DP_DipoleChargeModifierCheckOK(DP_DipoleChargeModifier *dcm)`

Check if there is any exceptions throw.

Parameters

**dcm** – The DipoleChargeModifier to use.

Returns

const char\* error message.

## Function DP\_DipoleChargeModifierComputeNList

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

`void DP_DipoleChargeModifierComputeNList(DP_DipoleChargeModifier *dcm, const int natom, const double *coord, const int *atype, const double *cell, const int *pairs, const int npairs, const double *delef_, const int nghost, const DP_Nlist *nlist, double *dfcorr_, double *dvcorr_)`

Evaluate the force and virial correction by using a dipole charge modifier with the neighbor list. (double version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

Parameters

- **dcm** – [in] The dipole charge modifier to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size natoms x 3.
- **atype** – [in] The atom types. The array should contain natoms ints.
- **cell** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **pairs** – [in] The pairs of atoms. The list should contain npairs pairs of ints.
- **npairs** – [in] The number of pairs.
- **delef\_** – [in] The electric field on each atom. The array should be of size nframes x natoms x 3.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **dfcorr\_** – [out] Output force correction. The array should be of size natoms x 3.
- **dvcorr\_** – [out] Output virial correction. The array should be of size 9.

## Function DP\_DipoleChargeModifierComputeNListf

- Defined in file `_source_api_c_include_c_api.h`

### Function Documentation

```
void DP_DipoleChargeModifierComputeNListf(DP_DipoleChargeModifier *dcm, const int natom, const
                                           float *coord, const int *atype, const float *cell, const int
                                           *pairs, const int npairs, const float *delef_, const int
                                           nghost, const DP_Nlist *nlist, float *dfcorr_, float
                                           *dvcorr_)
```

Evaluate the force and virial correction by using a dipole charge modifier with the neighbor list. (float version)

Warning: The output arrays should be allocated before calling this function. Pass NULL if not required.

#### Parameters

- **dcm** – [in] The dipole charge modifier to use.
- **natoms** – [in] The number of atoms.
- **coord** – [in] The coordinates of atoms. The array should be of size `natoms x 3`.
- **atype** – [in] The atom types. The array should contain `natoms` ints.
- **cell** – [in] The cell of the region. The array should be of size 9. Pass NULL if pbc is not used.
- **pairs** – [in] The pairs of atoms. The list should contain `npairs` pairs of ints.
- **npairs** – [in] The number of pairs.
- **delef\_** – [in] The electric field on each atom. The array should be of size `nframes x natoms x 3`.
- **nghost** – [in] The number of ghost atoms.
- **nlist** – [in] The neighbor list.
- **dfcorr\_** – [out] Output force correction. The array should be of size `natoms x 3`.
- **dvcorr\_** – [out] Output virial correction. The array should be of size 9.

## Function DP\_DipoleChargeModifierGetCutoff

- Defined in file `_source_api_c_include_c_api.h`

## Function Documentation

`double DP_DipoleChargeModifierGetCutoff(DP_DipoleChargeModifier *dt)`

Get the type map of a DipoleChargeModifier.

Parameters

`dcm` – [in] The DipoleChargeModifier to use.

Returns

The cutoff radius.

## Function DP\_DipoleChargeModifierGetNumbSelTypes

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`int DP_DipoleChargeModifierGetNumbSelTypes(DP_DipoleChargeModifier *dt)`

Get the number of sel types of a DipoleChargeModifier.

Parameters

`dcm` – [in] The DipoleChargeModifier to use.

Returns

The number of sel types

## Function DP\_DipoleChargeModifierGetNumbTypes

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`int DP_DipoleChargeModifierGetNumbTypes(DP_DipoleChargeModifier *dt)`

Get the type map of a DipoleChargeModifier.

Parameters

`dcm` – [in] The DipoleChargeModifier to use.

Returns

The number of types of the DipoleChargeModifier.

## Function DP\_DipoleChargeModifierGetSelTypes

- Defined in `file_source_api_c_include_c_api.h`

### Function Documentation

`int *DP_DipoleChargeModifierGetSelTypes(DP_DipoleChargeModifier *dt)`

Get sel types of a DipoleChargeModifier.

Parameters

`dcm` – [in] The DipoleChargeModifier to use.

Returns

The sel types

### Function DP\_NewDeepPot

- Defined in `file_source_api_c_include_c_api.h`

### Function Documentation

`DP_DeepPot *DP_NewDeepPot(const char *c_model)`

DP constructor with initialization.

Parameters

`c_model` – [in] The name of the frozen model file.

Returns

A pointer to the deep potential.

### Function DP\_NewDeepPotModelDevi

- Defined in `file_source_api_c_include_c_api.h`

### Function Documentation

`DP_DeepPotModelDevi *DP_NewDeepPotModelDevi(const char **c_models, int n_models)`

DP model deviation constructor with initialization.

Parameters

- `c_models` – [in] The array of the name of the frozen model file.
- `nmodels` – [in] The number of models.

### Function DP\_NewDeepPotModelDeviWithParam

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`DP_DeepPotModelDevi *DP_NewDeepPotModelDeviWithParam`(const char \*\*c\_model, const int n\_models, const int gpu\_rank, const char \*\*c\_file\_contents, const int n\_file\_contents, const int \*size\_file\_contents)

DP model deviation constructor with initialization.

Parameters

- **c\_models** – [in] The array of the name of the frozen model file.
- **nmodels** – [in] The number of models.
- **gpu\_rank** – [in] The rank of the GPU.
- **c\_file\_contents** – [in] The contents of the model file.
- **n\_file\_contents** – [in] The number of the contents of the model file.
- **size\_file\_contents** – [in] The sizes of the contents of the model file.

Returns

`DP_DeepPotModelDevi*` A pointer to the deep potential model deviation.

## Function `DP_NewDeepPotWithParam`

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`DP_DeepPot *DP_NewDeepPotWithParam`(const char \*c\_model, const int gpu\_rank, const char \*c\_file\_content)

DP constructor with initialization.

Parameters

- **c\_model** – The name of the frozen model file.
- **gpu\_rank** – The rank of the GPU.
- **c\_file\_content** – Broken implementation. Use `DP_NewDeepPotWithParam2` instead.

Returns

`DP_DeepPot*` A pointer to the deep potential.

## Function `DP_NewDeepPotWithParam2`

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`DP_DeepPot` \*`DP_NewDeepPotWithParam2`(const char \*`c_model`, const int `gpu_rank`, const char \*`c_file_content`, const int `size_file_content`)

DP constructor with initialization.

Version  
2

Parameters

- `c_model` – The name of the frozen model file.
- `gpu_rank` – The rank of the GPU.
- `c_file_content` – The content of the model file.
- `size_file_content` – The size of the model file.

Returns

`DP_DeepPot*` A pointer to the deep potential.

## Function `DP_NewDeepTensor`

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`DP_DeepTensor` \*`DP_NewDeepTensor`(const char \*`c_model`)

Deep Tensor constructor with initialization.

Parameters

`c_model` – [in] The name of the frozen model file.

Returns

A pointer to the deep tensor.

## Function `DP_NewDeepTensorWithParam`

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`DP_DeepTensor` \*`DP_NewDeepTensorWithParam`(const char \*`c_model`, const int `gpu_rank`, const char \*`c_name_scope`)

Deep Tensor constructor with initialization.

Parameters

- `c_model` – The name of the frozen model file.
- `gpu_rank` – The rank of the GPU.
- `c_name_scope` – The name scope.



Returns

DP\_DeepTensor\* A pointer to the deep tensor.

### Function DP\_NewDipoleChargeModifier

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

DP\_DipoleChargeModifier \*DP\_NewDipoleChargeModifier(const char \*c\_model)

Dipole charge modifier constructor with initialization.

Parameters

**c\_model** – [in] The name of the frozen model file.

Returns

A pointer to the dipole charge modifier.

### Function DP\_NewDipoleChargeModifierWithParam

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

DP\_DipoleChargeModifier \*DP\_NewDipoleChargeModifierWithParam(const char \*c\_model, const int  
gpu\_rank, const char  
\*c\_name\_scope)

Dipole charge modifier constructor with initialization.

Parameters

- **c\_model** – The name of the frozen model file.
- **gpu\_rank** – The rank of the GPU.
- **c\_name\_scope** – The name scope.

Returns

DP\_DipoleChargeModifier\* A pointer to the dipole charge modifier.

### Function DP\_NewNlist

- Defined in file\_source\_api\_c\_include\_c\_api.h

## Function Documentation

`DP_Nlist *DP_NewNlist(int inum_, int *ilist_, int *numneigh_, int **firstneigh_)`

Create a new neighbor list.

Parameters

- **inum\_** – [in] Number of core region atoms
- **Array** – [in] stores the core region atom's index
- **Array** – [in] stores the core region atom's neighbor atom number
- **Array** – [in] stores the core region atom's neighbor index

Returns

A pointer to the neighbor list.

## Function DP\_NlistCheckOK

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`const char *DP_NlistCheckOK(DP_Nlist *dp)`

Check if there is any exceptions throw.

Parameters

**dp** – The neighbor list to use.

Returns

`const char*` error message.

## Function DP\_PrintSummary

- Defined in `file_source_api_c_include_c_api.h`

## Function Documentation

`void DP_PrintSummary(const char *c_pre)`

Print the summary of DeePMD-kit, including the version and the build information.

Parameters

**c\_pre** – [in] The prefix to each line.

### Function DP\_ReadFileToChar

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

const char \*DP\_ReadFileToChar(const char \*c\_model)

Read a file to a char array.

Parameters

**c\_model** – [in] The name of the file.

Returns

const char\* The char array.

### Function DP\_ReadFileToChar2

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

const char \*DP\_ReadFileToChar2(const char \*c\_model, int \*size)

Read a file to a char array. This version can handle string with '\0'.

Version

2

Parameters

- **c\_model** – [in] The name of the file.
- **size** – [out] The size of the char array.

Returns

const char\* The char array.

### Function DP\_SelectByType

- Defined in file\_source\_api\_c\_include\_c\_api.h

#### Function Documentation

void DP\_SelectByType(const int natoms, const int \*atype, const int nghost, const int nsel\_type, const int \*sel\_type, int \*fwd\_map, int \*nreal, int \*bkw\_map, int \*nghost\_real)

Get forward and backward map of selected atoms by atom types.

Parameters

- **natoms** – [in] The number of atoms.
- **atype** – [in] The atom types of all atoms.

- **nghost** – [in] The number of ghost atoms.
- **nselect\_type** – [in] The number of selected atom types.
- **select\_type** – [in] The selected atom types.
- **fwd\_map** – [out] The forward map with size natoms.
- **nreal** – [out] The number of selected real atoms.
- **bkw\_map** – [out] The backward map with size nreal.
- **nghost\_real** – [out] The number of selected ghost atoms.

### Function DP\_SelectMapInt

- Defined in file\_source\_api\_c\_include\_c\_api.h

### Function Documentation

void **DP\_SelectMapInt**(const int \*in, const int \*fwd\_map, const int stride, const int nall1, const int nall2, int \*out)

Apply the given map to a vector. Assume nframes is 1.

Parameters

- **in** – [in] The input vector.
- **fwd\_map** – [in] The map.
- **stride** – [in] The stride of the input vector.
- **nall1** – [in] The number of atoms in the input vector.
- **nall2** – [out] The number of atoms in the output vector.
- **out** – [out] The output vector.

## 21.3.4 Defines

### Define DP\_CHECK\_OK

- Defined in file\_source\_api\_c\_include\_deepmd.hpp

### Define Documentation

**DP\_CHECK\_OK**(check\_func, dp)

Check if any exceptions throw in the C++ API. Throw if possible.

**Define DP\_NEW\_OK**

- Defined in file `_source_api_c_include_c_api_internal.h`

**Define Documentation**

`DP_NEW_OK(dpcls, xx)`

**Define DP\_REQUIRES\_OK**

- Defined in file `_source_api_c_include_c_api_internal.h`

**Define Documentation**

`DP_REQUIRES_OK(dp, xx)`

**21.3.5 Typedefs****Typedef DP\_DeepPot**

- Defined in file `_source_api_c_include_c_api.h`

**Typedef Documentation**

typedef struct `DP_DeepPot` `DP_DeepPot`

The deep potential.

**Typedef DP\_DeepPotModelDevi**

- Defined in file `_source_api_c_include_c_api.h`

**Typedef Documentation**

typedef struct `DP_DeepPotModelDevi` `DP_DeepPotModelDevi`

The deep potential model deviation.

**Typedef DP\_DeepTensor**

- Defined in file `_source_api_c_include_c_api.h`

**Typedef Documentation**

typedef struct [DP\\_DeepTensor](#) **DP\_DeepTensor**

The deep tensor.

**Typedef DP\_DipoleChargeModifier**

- Defined in file `_source_api_c_include_c_api.h`

**Typedef Documentation**

typedef struct [DP\\_DipoleChargeModifier](#) **DP\_DipoleChargeModifier**

The dipole charge modifier.

**Typedef DP\_Nlist**

- Defined in file `_source_api_c_include_c_api.h`

**Typedef Documentation**

typedef struct [DP\\_Nlist](#) **DP\_Nlist**

Neighbor list.

## 22.1 Class Hierarchy

- Namespace deepmd
  - Struct deepmd\_exception
  - Struct deepmd\_exception\_oom
  - Template Struct EwaldParameters
  - Struct InputNlist
  - Template Struct Region
- Template Struct DescrptSeRGPUExecuteFunctor
- Template Struct GeluGPUExecuteFunctor
- Template Struct GeluGradGPUExecuteFunctor
- Template Struct GeluGradGradGPUExecuteFunctor
- Template Struct ProdForceSeAGPUExecuteFunctor
- Template Struct ProdForceSeRGPUExecuteFunctor
- Template Struct ProdVirialSeAGPUExecuteFunctor
- Template Struct ProdVirialSeRGPUExecuteFunctor
- Template Struct TabulateCheckerGPUExecuteFunctor
- Template Struct TabulateFusionGPUExecuteFunctor
- Template Struct TabulateFusionGradGPUExecuteFunctor
- Template Class SimulationRegion
- Union U\_Flt64\_Int64

## 22.2 File Hierarchy

- dir\_source
  - dir\_source\_lib
    - \*dir\_source\_lib\_include
      - file\_source\_lib\_include\_ComputeDescriptor.h
      - file\_source\_lib\_include\_coord.h
      - file\_source\_lib\_include\_device.h
      - file\_source\_lib\_include\_DeviceFunctor.h
      - file\_source\_lib\_include\_env\_mat.h
      - file\_source\_lib\_include\_env\_mat\_nvnmd.h
      - file\_source\_lib\_include\_errors.h
      - file\_source\_lib\_include\_ewald.h
      - file\_source\_lib\_include\_fmt\_nlist.h
      - file\_source\_lib\_include\_gelu.h
      - file\_source\_lib\_include\_gpu\_cuda.h
      - file\_source\_lib\_include\_gpu\_rocm.h
      - file\_source\_lib\_include\_map\_aparam.h
      - file\_source\_lib\_include\_neighbor\_list.h
      - file\_source\_lib\_include\_neighbor\_stat.h
      - file\_source\_lib\_include\_pair\_tab.h
      - file\_source\_lib\_include\_pairwise.h
      - file\_source\_lib\_include\_prod\_env\_mat.h
      - file\_source\_lib\_include\_prod\_env\_mat\_nvnmd.h
      - file\_source\_lib\_include\_prod\_force.h
      - file\_source\_lib\_include\_prod\_force\_grad.h
      - file\_source\_lib\_include\_prod\_virial.h
      - file\_source\_lib\_include\_prod\_virial\_grad.h
      - file\_source\_lib\_include\_region.h
      - file\_source\_lib\_include\_SimulationRegion.h
      - file\_source\_lib\_include\_SimulationRegion\_Impl.h
      - file\_source\_lib\_include\_soft\_min\_switch.h
      - file\_source\_lib\_include\_soft\_min\_switch\_force.h
      - file\_source\_lib\_include\_soft\_min\_switch\_force\_grad.h
      - file\_source\_lib\_include\_soft\_min\_switch\_virial.h
      - file\_source\_lib\_include\_soft\_min\_switch\_virial\_grad.h



- file\_source\_lib\_include\_switcher.h
- file\_source\_lib\_include\_tabulate.h
- file\_source\_lib\_include\_utilities.h

## 22.3 Full API

### 22.3.1 Namespaces

#### Namespace deepmd

##### Contents

- Classes
- Functions
- Variables

#### Classes

- Struct deepmd\_exception
- Struct deepmd\_exception\_oom
- Template Struct EwaldParameters
- Struct InputNlist
- Template Struct Region

#### Functions

- Template Function deepmd::build\_nlist\_cpu
- Template Function deepmd::build\_nlist\_gpu
- Template Function deepmd::compute\_cell\_info
- Function deepmd::convert\_nlist
- Function deepmd::convert\_nlist\_gpu\_device
- Template Function deepmd::convert\_to\_inter\_cpu
- Template Function deepmd::convert\_to\_inter\_gpu
- Template Function deepmd::convert\_to\_phys\_cpu
- Template Function deepmd::convert\_to\_phys\_gpu
- Template Function deepmd::copy\_coord\_cpu
- Template Function deepmd::copy\_coord\_gpu
- Function deepmd::cos\_switch(const double&, const double&, const double&)

- Function `deepmd::cos_switch(double&, double&, const double&, const double&, const double&)`
- Template Function `deepmd::cprod`
- Function `deepmd::cum_sum`
- Template Function `deepmd::delete_device_memory`
- Template Function `deepmd::dot1`
- Template Function `deepmd::dot2`
- Template Function `deepmd::dot3`
- Template Function `deepmd::dot4`
- Template Function `deepmd::dotmv3`
- Function `deepmd::DPGetDeviceCount`
- Function `deepmd::dprc_pairwise_map_cpu`
- Function `deepmd::DPSetDevice`
- Template Function `deepmd::env_mat_a_cpu`
- Template Function `deepmd::env_mat_a_nvnmnd_quantize_cpu`
- Function `deepmd::env_mat_nbor_update`
- Template Function `deepmd::env_mat_r_cpu`
- Template Function `deepmd::ewald_recip`
- Function `deepmd::filter_ftype_gpu`
- Template Function `deepmd::format_nbor_list_gpu`
- Template Function `deepmd::format_nlist_cpu`
- Function `deepmd::free_nlist_gpu_device`
- Template Function `deepmd::gelu_cpu`
- Template Function `deepmd::gelu_gpu`
- Template Function `deepmd::gelu_grad_cpu`
- Template Function `deepmd::gelu_grad_gpu`
- Template Function `deepmd::gelu_grad_grad_cpu`
- Template Function `deepmd::gelu_grad_grad_gpu`
- Function `deepmd::group_atoms_cpu`
- Template Function `deepmd::init_region_cpu`
- Template Function `deepmd::invsqrt`
- Specialized Template Function `deepmd::invsqrt< double >`
- Specialized Template Function `deepmd::invsqrt< float >`
- Template Function `deepmd::malloc_device_memory(FPTYPE *&, std::vector<FPTYPE>&)`
- Template Function `deepmd::malloc_device_memory(FPTYPE *&, const std::vector<FPTYPE>&)`
- Template Function `deepmd::malloc_device_memory(FPTYPE *&, const int)`
- Template Function `deepmd::malloc_device_memory_sync(FPTYPE *&, std::vector<FPTYPE>&)`

- Template Function `deepmd::malloc_device_memory_sync(FPTYPE *&, const FPTYPE *, const int)`
- Template Function `deepmd::malloc_device_memory_sync(FPTYPE *&, const std::vector<FPTYPE>&)`
- Template Function `deepmd::map_aparam_cpu`
- Function `deepmd::max_numneigh`
- Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, FPTYPE *, const int)`
- Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, std::vector<FPTYPE>&)`
- Template Function `deepmd::memcpy_host_to_device(FPTYPE *, std::vector<FPTYPE>&)`
- Template Function `deepmd::memcpy_host_to_device(FPTYPE *, const std::vector<FPTYPE>&)`
- Template Function `deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)`
- Template Function `deepmd::memset_device_memory`
- Template Function `deepmd::neighbor_stat_gpu`
- Template Function `deepmd::normalize_coord_cpu`
- Template Function `deepmd::normalize_coord_gpu`
- Template Function `deepmd::pair_tab_cpu`
- Template Function `deepmd::prod_env_mat_a_cpu`
- Template Function `deepmd::prod_env_mat_a_gpu`
- Template Function `deepmd::prod_env_mat_a_nvnmmd_quantize_cpu`
- Template Function `deepmd::prod_env_mat_r_cpu`
- Template Function `deepmd::prod_env_mat_r_gpu`
- Template Function `deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int)`
- Template Function `deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int, const int, const int)`
- Template Function `deepmd::prod_force_a_gpu`
- Template Function `deepmd::prod_force_grad_a_cpu`
- Template Function `deepmd::prod_force_grad_a_gpu`
- Template Function `deepmd::prod_force_grad_r_cpu`
- Template Function `deepmd::prod_force_grad_r_gpu`
- Template Function `deepmd::prod_force_r_cpu`
- Template Function `deepmd::prod_force_r_gpu`
- Template Function `deepmd::prod_virial_a_cpu`
- Template Function `deepmd::prod_virial_a_gpu`
- Template Function `deepmd::prod_virial_grad_a_cpu`
- Template Function `deepmd::prod_virial_grad_a_gpu`
- Template Function `deepmd::prod_virial_grad_r_cpu`
- Template Function `deepmd::prod_virial_grad_r_gpu`

- Template Function `deepmd::prod_virial_r_cpu`
- Template Function `deepmd::prod_virial_r_gpu`
- Template Function `deepmd::soft_min_switch_cpu`
- Template Function `deepmd::soft_min_switch_force_cpu`
- Template Function `deepmd::soft_min_switch_force_grad_cpu`
- Template Function `deepmd::soft_min_switch_virial_cpu`
- Template Function `deepmd::soft_min_switch_virial_grad_cpu`
- Function `deepmd::spline3_switch`
- Template Function `deepmd::spline5_switch`
- Template Function `deepmd::tabulate_fusion_se_a_cpu`
- Template Function `deepmd::tabulate_fusion_se_a_gpu`
- Template Function `deepmd::tabulate_fusion_se_a_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_a_grad_gpu`
- Template Function `deepmd::tabulate_fusion_se_a_grad_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_a_grad_grad_gpu`
- Template Function `deepmd::tabulate_fusion_se_r_cpu`
- Template Function `deepmd::tabulate_fusion_se_r_gpu`
- Template Function `deepmd::tabulate_fusion_se_r_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_r_grad_gpu`
- Template Function `deepmd::tabulate_fusion_se_r_grad_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_r_grad_grad_gpu`
- Template Function `deepmd::tabulate_fusion_se_t_cpu`
- Template Function `deepmd::tabulate_fusion_se_t_gpu`
- Template Function `deepmd::tabulate_fusion_se_t_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_t_grad_gpu`
- Template Function `deepmd::tabulate_fusion_se_t_grad_grad_cpu`
- Template Function `deepmd::tabulate_fusion_se_t_grad_grad_gpu`
- Template Function `deepmd::test_encoding_decoding_nbor_info_gpu`
- Function `deepmd::use_nei_info_cpu`
- Function `deepmd::use_nei_info_gpu`
- Function `deepmd::use_nlist_map`
- Template Function `deepmd::volume_cpu`
- Template Function `deepmd::volume_gpu`

## Variables

- Variable `deepmd::ElectrostaticConversion`

## Namespace `std`

### 22.3.2 Classes and Structs

#### Struct `deepmd_exception`

- Defined in `file_source_lib_include_errors.h`

#### Inheritance Relationships

##### Base Type

- `public std::runtime_error`

##### Derived Type

- `public deepmd::deepmd_exception_oom (Struct deepmd_exception_oom)`

#### Struct Documentation

struct `deepmd_exception` : `public std::runtime_error`  
 General DeePMD-kit exception. Throw if anything doesn't work.  
 Subclassed by `deepmd::deepmd_exception_oom`

##### Public Functions

```
inline deepmd_exception()
inline deepmd_exception(const std::string &msg)
```

#### Struct `deepmd_exception_oom`

- Defined in `file_source_lib_include_errors.h`

## Inheritance Relationships

### Base Type

- `public deepmd::deepmd_exception (Struct deepmd_exception)`

### Struct Documentation

struct `deepmd_exception_oom` : public `deepmd::deepmd_exception`

#### Public Functions

`inline deepmd_exception_oom()`

`inline deepmd_exception_oom(const std::string &msg)`

### Template Struct `EwaldParameters`

- Defined in `file_source_lib_include_ewald.h`

### Struct Documentation

template<typename `VALUETYPE`>

struct `EwaldParameters`

#### Public Members

`VALUETYPE rcut = 6.0`

`VALUETYPE beta = 2`

`VALUETYPE spacing = 4`

### Struct `InputNlist`

- Defined in `file_source_lib_include_neighbor_list.h`

## Struct Documentation

struct **InputNlist**

Construct **InputNlist** with the input LAMMPS nbor list info.

### Public Functions

inline **InputNlist**()

inline **InputNlist**(int inum\_, int \*ilist\_, int \*numneigh\_, int \*\*firstneigh\_)

inline ~**InputNlist**()

### Public Members

int **inum**

Number of core region atoms.

int \***ilist**

Array stores the core region atom's index.

int \***numneigh**

Array stores the core region atom's neighbor atom number.

int \*\***firstneigh**

Array stores the core region atom's neighbor index.

## Template Struct Region

- Defined in file\_source\_lib\_include\_region.h

## Struct Documentation

template<typename **FPTYPE**>

struct **Region**

### Public Functions

`Region()`

`Region(FPTYPE *extern_box, FPTYPE *extern_rec_box)`

`~Region()`

### Public Members

`FPTYPE *box`

`FPTYPE *rec_box`

### Template Struct `DescriptSeRGPUExecuteFunctor`

- Defined in `file_source_lib_include_DeviceFunctor.h`

### Struct Documentation

`template<typename FPTYPE>`

`struct DescriptSeRGPUExecuteFunctor`

### Public Functions

`void operator() (const FPTYPE *coord, const int *type, const int *ilist, const int *jrange, const int *jlist, int *array_int, unsigned long long *array_longlong, const FPTYPE *avg, const FPTYPE *std, FPTYPE *descript, FPTYPE *descript_deriv, FPTYPE *rij, int *nlist, const int nloc, const int nall, const int nnei, const int ndescript, const float rcut_r, const float rcut_r_smth, const std::vector<int> sec_a, const bool fill_nei_a, const int MAGIC_NUMBER)`

### Template Struct `GeluGPUExecuteFunctor`

- Defined in `file_source_lib_include_DeviceFunctor.h`

### Struct Documentation

`template<typename FPTYPE>`

`struct GeluGPUExecuteFunctor`



### Public Functions

```
void operator() (const FTYPE *in, FTYPE *out, const int size)
```

### Template Struct GeluGradGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

### Struct Documentation

```
template<typename FTYPE>
struct GeluGradGPUExecuteFunctor
```

### Public Functions

```
void operator() (const FTYPE *dy, const FTYPE *in, FTYPE *out, const int size)
```

### Template Struct GeluGradGradGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

### Struct Documentation

```
template<typename FTYPE>
struct GeluGradGradGPUExecuteFunctor
```

### Public Functions

```
void operator() (const FTYPE *dy, const FTYPE *dy_, const FTYPE *in, FTYPE *out, const
                int size)
```

### Template Struct ProdForceSeAGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

### Struct Documentation

```
template<typename FTYPE>
struct ProdForceSeAGPUExecuteFunctor
```

### Public Functions

```
void operator() (FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int
                *nlist, const int nloc, const int nall, const int nnei, const int ndescript, const int
                n_a_sel, const int n_a_shift)
```

### Template Struct ProdForceSeRGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

### Struct Documentation

```
template<typename FPTYPE>
```

```
struct ProdForceSeRGPUExecuteFunctor
```

### Public Functions

```
void operator() (FPTYPE *force, const FPTYPE *net_derive, const FPTYPE *in_deriv, const int
                *nlist, const int nloc, const int nall, const int nnei, const int ndescript)
```

### Template Struct ProdVirialSeAGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

### Struct Documentation

```
template<typename FPTYPE>
```

```
struct ProdVirialSeAGPUExecuteFunctor
```

### Public Functions

```
void operator() (FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE
                *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int
                nnei, const int ndescript, const int n_a_sel, const int n_a_shift)
```

### Template Struct ProdVirialSeRGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

## Struct Documentation

```
template<typename FPTYPE>
```

```
struct ProdVirialSeRGPUExecuteFunctor
```

### Public Functions

```
void operator() (FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv, const FPTYPE
                 *in_deriv, const FPTYPE *rij, const int *nlist, const int nloc, const int nall, const int
                 nnei, const int ndescript)
```

## Template Struct TabulateCheckerGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

## Struct Documentation

```
template<typename FPTYPE>
```

```
struct TabulateCheckerGPUExecuteFunctor
```

### Public Functions

```
void operator() (const FPTYPE *table_info, const FPTYPE *in, int *out, const int nloc, const int
                 nnei)
```

## Template Struct TabulateFusionGPUExecuteFunctor

- Defined in file\_source\_lib\_include\_DeviceFunctor.h

## Struct Documentation

```
template<typename FPTYPE>
```

```
struct TabulateFusionGPUExecuteFunctor
```

### Public Functions

```
void operator() (const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const
                 FPTYPE *ff, const int nloc, const int nnei, const int last_layer_size, FPTYPE *out)
```

### Template Struct `TabulateFusionGradGPUExecuteFunctor`

- Defined in file `_source_lib_include_DeviceFunctor.h`

#### Struct Documentation

```
template<typename FPTYPE>
```

```
struct TabulateFusionGradGPUExecuteFunctor
```

#### Public Functions

```
void operator() (const FPTYPE *table, const FPTYPE *table_info, const FPTYPE *in, const  
FPTYPE *ff, const FPTYPE *dy, const int nloc, const int nnei, const int  
last_layer_size, FPTYPE *dy_dx, FPTYPE *dy_df)
```

### Template Class `SimulationRegion`

- Defined in file `_source_lib_include_SimulationRegion.h`

#### Class Documentation

```
template<typename VALUETYPE>
```

```
class SimulationRegion
```

#### Public Functions

```
inline void reinitBox(const double *boxv)  
inline void affineTransform(const double *affine_map)  
inline void reinitOrigin(const double *orig)  
inline void reinitOrigin(const std::vector<double> &orig)  
void backup()  
void recover()  
SimulationRegion()  
~SimulationRegion()  
inline double *getBoxTensor()  
inline const double *getBoxTensor() const  
inline double *getRecBoxTensor()  
inline const double *getRecBoxTensor() const
```

```

inline double *getBoxOrigin()

inline const double *getBoxOrigin() const

inline double getVolume() const

inline void toFaceDistance(double *dd) const

inline void phys2Inter(double *i_v, const VALUETYPE *p_v) const

inline void inter2Phys(VALUETYPE *p_v, const double *i_v) const

inline bool isPeriodic(const int dim) const

inline double *getShiftVec(const int index = 0)

inline const double *getShiftVec(const int index = 0) const

inline int getShiftIndex(const int *idx) const

inline int getNullShiftIndex() const

inline void shiftCoord(const int *idx, VALUETYPE &x, VALUETYPE &y, VALUETYPE &z) const

inline void diffNearestNeighbor(const VALUETYPE *r0, const VALUETYPE *r1, VALUETYPE
                                *phys) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const
                                         VALUETYPE z0, const VALUETYPE x1, const
                                         VALUETYPE y1, const VALUETYPE z1, VALUETYPE
                                         &dx, VALUETYPE &dy, VALUETYPE &dz) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const
                                         VALUETYPE z0, const VALUETYPE x1, const
                                         VALUETYPE y1, const VALUETYPE z1, VALUETYPE
                                         &dx, VALUETYPE &dy, VALUETYPE &dz, int &shift_x,
                                         int &shift_y, int &shift_z) const

inline virtual void diffNearestNeighbor(const VALUETYPE x0, const VALUETYPE y0, const
                                         VALUETYPE z0, const VALUETYPE x1, const
                                         VALUETYPE y1, const VALUETYPE z1, VALUETYPE
                                         &dx, VALUETYPE &dy, VALUETYPE &dz,
                                         VALUETYPE &shift_x, VALUETYPE &shift_y,
                                         VALUETYPE &shift_z) const

```

### Public Static Functions

```

static inline int compactIndex(const int *idx)

static inline int getNumbShiftVec()

static inline int getShiftVecTotalSize()

```

### Protected Functions

```
void computeShiftVec()
```

```
inline double *getInterShiftVec(const int index = 0)
```

```
inline const double *getInterShiftVec(const int index = 0) const
```

### Protected Attributes

```
double shift_vec[shift_vec_size]
```

```
double inter_shift_vec[shift_vec_size]
```

### Protected Static Functions

```
static inline int index3to1(const int tx, const int ty, const int tz)
```

### Protected Static Attributes

```
static const int SPACENDIM = 3
```

```
static const int DBOX_XX = 1
```

```
static const int DBOX_YY = 1
```

```
static const int DBOX_ZZ = 1
```

```
static const int NBOX_XX = DBOX_XX * 2 + 1
```

```
static const int NBOX_YY = DBOX_YY * 2 + 1
```

```
static const int NBOX_ZZ = DBOX_ZZ * 2 + 1
```

```
static const int shift_info_size = NBOX_XX * NBOX_YY * NBOX_ZZ
```

```
static const int shift_vec_size = SPACENDIM * shift_info_size
```

### 22.3.3 Unions

#### Union `U_Flt64_Int64`

- Defined in `file_source_lib_include_env_mat_nvnmd.h`

#### Union Documentation

`union U_Flt64_Int64`

#### Public Members

`double nflt`

`int64_t nint`

### 22.3.4 Functions

#### Template Function `add_flt_nvnmd`

- Defined in `file_source_lib_include_env_mat_nvnmd.h`

#### Function Documentation

```
template<class T>
void add_flt_nvnmd(T &y, T x1, T x2)
```

**Function** `build_nlist`(`std::vector<std::vector<int>>&`, `std::vector<std::vector<int>>&`, `const std::vector<double>&`, `const int&`, `const double&`, `const double&`, `const std::vector<int>&`, `const std::vector<int>&`, `const std::vector<int>&`, `const SimulationRegion<double>&`, `const std::vector<int>&`)

- Defined in `file_source_lib_include_neighbor_list.h`

#### Function Documentation

```
void build_nlist(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const
std::vector<double> &coord, const int &nloc, const double &rc0, const double &rc1,
const std::vector<int> &nat_stt_, const std::vector<int> &nat_end_, const
std::vector<int> &ext_stt_, const std::vector<int> &ext_end_, const
SimulationRegion<double> &region, const std::vector<int> &global_grid)
```

**Function** `build_nlist(std::vector<std::vector<int>>&, std::vector<std::vector<int>>&, const std::vector<double>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)`

- Defined in file\_source\_lib\_include\_neighbor\_list.h

### Function Documentation

void **build\_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const double &rc0, const double &rc1, const std::vector<int> &grid, const SimulationRegion<double> &region)

**Function** `build_nlist(std::vector<std::vector<int>>&, std::vector<std::vector<int>>&, const std::vector<double>&, const std::vector<int>&, const std::vector<int>&, const double&, const double&, const std::vector<int>&, const SimulationRegion<double>&)`

- Defined in file\_source\_lib\_include\_neighbor\_list.h

### Function Documentation

void **build\_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const std::vector<int> &sel0, const std::vector<int> &sel1, const double &rc0, const double &rc1, const std::vector<int> &grid, const SimulationRegion<double> &region)

**Function** `build_nlist(std::vector<std::vector<int>>&, std::vector<std::vector<int>>&, const std::vector<double>&, const double&, const double&, const SimulationRegion<double> *)`

- Defined in file\_source\_lib\_include\_neighbor\_list.h

### Function Documentation

void **build\_nlist**(std::vector<std::vector<int>> &nlist0, std::vector<std::vector<int>> &nlist1, const std::vector<double> &coord, const double &rc0\_, const double &rc1\_, const SimulationRegion<double> \*region = NULL)

**Function** `compute_descriptor(std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const SimulationRegion<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)`

- Defined in file\_source\_lib\_include\_ComputeDescriptor.h



## Function Documentation

```
inline void compute_descriptor(std::vector<double> &descript_a, std::vector<double> &descript_r,
                                std::vector<double> &rot_mat, const std::vector<double> &posi, const
                                int &ntypes, const std::vector<int> &type, const
                                SimulationRegion<double> &region, const bool &b_pbc, const int
                                &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int>
                                &fmt_nlist_r, const std::vector<int> &sec_a, const std::vector<int>
                                &sec_r, const int axis0_type, const int axis0_idx, const int axis1_type,
                                const int axis1_idx)
```

**Function `compute_descriptor`**(std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, std::vector<double>&, const std::vector<double>&, const int&, const std::vector<int>&, const [SimulationRegion](#)<double>&, const bool&, const int&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const std::vector<int>&, const int, const int, const int, const int)

- Defined in file\_source\_lib\_include\_ComputeDescriptor.h

## Function Documentation

```
inline void compute_descriptor(std::vector<double> &descript_a, std::vector<double>
                                &descript_a_deriv, std::vector<double> &descript_r,
                                std::vector<double> &descript_r_deriv, std::vector<double> &rrij_a,
                                std::vector<double> &rrij_r, std::vector<double> &rot_mat, const
                                std::vector<double> &posi, const int &ntypes, const std::vector<int>
                                &type, const SimulationRegion<double> &region, const bool &b_pbc,
                                const int &i_idx, const std::vector<int> &fmt_nlist_a, const
                                std::vector<int> &fmt_nlist_r, const std::vector<int> &sec_a, const
                                std::vector<int> &sec_r, const int axis0_type, const int axis0_idx, const
                                int axis1_type, const int axis1_idx)
```

## Function `compute_descriptor_se_a_ef_para`

- Defined in file\_source\_lib\_include\_ComputeDescriptor.h

## Function Documentation

```
inline void compute_descriptor_se_a_ef_para(std::vector<double> &descript_a, std::vector<double>
                                &descript_a_deriv, std::vector<double> &rrij_a, const
                                std::vector<double> &posi, const int &ntypes, const
                                std::vector<int> &type, const
                                SimulationRegion<double> &region, const bool &b_pbc,
                                const std::vector<double> &efield, const int &i_idx, const
                                std::vector<int> &fmt_nlist_a, const std::vector<int>
                                &sec_a, const double &rmin, const double &rmax)
```

### Function `compute_descriptor_se_a_ef_vert`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

#### Function Documentation

```
inline void compute_descriptor_se_a_ef_vert(std::vector<double> &descript_a, std::vector<double>
&descript_a_deriv, std::vector<double> &rij_a, const
std::vector<double> &posi, const int &ntypes, const
std::vector<int> &type, const
SimulationRegion<double> &region, const bool &b_pbc,
const std::vector<double> &efield, const int &i_idx, const
std::vector<int> &fmt_nlist_a, const std::vector<int>
&sec_a, const double &rmin, const double &rmax)
```

### Function `compute_descriptor_se_a_extf`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

#### Function Documentation

```
inline void compute_descriptor_se_a_extf(std::vector<double> &descript_a, std::vector<double>
&descript_a_deriv, std::vector<double> &rij_a, const
std::vector<double> &posi, const int &ntypes, const
std::vector<int> &type, const SimulationRegion<double>
&region, const bool &b_pbc, const std::vector<double>
&efield, const int &i_idx, const std::vector<int>
&fmt_nlist_a, const std::vector<int> &sec_a, const double
&rmin, const double &rmax)
```

### Function `compute_dRdT`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

#### Function Documentation

Warning: doxygenfunction: Unable to resolve function “compute\_dRdT” with arguments (double (\*), const double\*, const double\*, const double\*) in doxygen xml output for project “core” from directory: `_build/core/xml/`. Potential matches:

```
- void compute_dRdT(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

## Function `compute_dRdT_1`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

### Function Documentation

Warning: doxygenfunction: Unable to resolve function “compute\_dRdT\_1” with arguments (double (\*), const double\*, const double\*, const double\*) in doxygen xml output for project “core” from directory: `_build/core/xml/`. Potential matches:

```
- void compute_dRdT_1(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

## Function `compute_dRdT_2`

- Defined in file `_source_lib_include_ComputeDescriptor.h`

### Function Documentation

Warning: doxygenfunction: Unable to resolve function “compute\_dRdT\_2” with arguments (double (\*), const double\*, const double\*, const double\*) in doxygen xml output for project “core” from directory: `_build/core/xml/`. Potential matches:

```
- void compute_dRdT_2(double (*dRdT)[9], const double *r1, const double *r2, const double *rot)
```

## Function `copy_coord`

- Defined in file `_source_lib_include_neighbor_list.h`

### Function Documentation

```
void copy_coord(std::vector<double> &out_c, std::vector<int> &out_t, std::vector<int> &mapping,
               std::vector<int> &ncell, std::vector<int> &ngcell, const std::vector<double> &in_c,
               const std::vector<int> &in_t, const double &rc, const SimulationRegion<double>
               &region)
```

### Template Function `deepmd::build_nlist_cpu`

- Defined in file `_source_lib_include_neighbor_list.h`

#### Function Documentation

```
template<typename FPTYPE>
int deepmd::build_nlist_cpu(InputNlist &nlist, int *max_list_size, const FPTYPE *c_cpy, const int
                           &nloc, const int &nall, const int &mem_size, const float &rcut)
```

### Template Function `deepmd::build_nlist_gpu`

- Defined in file `_source_lib_include_neighbor_list.h`

#### Function Documentation

```
template<typename FPTYPE>
int deepmd::build_nlist_gpu(InputNlist &nlist, int *max_list_size, int *nlist_data, const FPTYPE
                           *c_cpy, const int &nloc, const int &nall, const int &mem_size, const float
                           &rcut)
```

### Template Function `deepmd::compute_cell_info`

- Defined in file `_source_lib_include_coord.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::compute_cell_info(int *cell_info, const float &rcut, const deepmd::Region<FPTYPE>
                              &region)
```

### Function `deepmd::convert_nlist`

- Defined in file `_source_lib_include_neighbor_list.h`

#### Function Documentation

```
void deepmd::convert_nlist(InputNlist &to_nlist, std::vector<std::vector<int>> &from_nlist)
```

Construct the `InputNlist` with a two-dimensional vector.

Parameters

- `to_nlist` – `InputNlist` struct which stores the neighbor information of the core region atoms.
- `from_nlist` – Vector which stores the neighbor information of the core region atoms.

### Function `deepmd::convert_nlist_gpu_device`

- Defined in file `_source_lib_include_neighbor_list.h`

### Function Documentation

```
void deepmd::convert_nlist_gpu_device(InputNlist &gpu_nlist, InputNlist &cpu_nlist, int
                                     *&gpu_memory, const int &max_nbor_size)
```

Convert the a host memory `InputNlist` to a device memory `InputNlist`.

Parameters

- `cpu_nlist` – Host memory `InputNlist` struct which stores the neighbor information of the core region atoms
- `gpu_nlist` – Device memory `InputNlist` struct which stores the neighbor information of the core region atoms
- `gpu_memory` – Device array which stores the elements of `gpu_nlist`
- `max_nbor_size` –

### Template Function `deepmd::convert_to_inter_cpu`

- Defined in file `_source_lib_include_region.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::convert_to_inter_cpu(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE
                                  *rp)
```

### Template Function `deepmd::convert_to_inter_gpu`

- Defined in file `_source_lib_include_region.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::convert_to_inter_gpu(FPTYPE *ri, const Region<FPTYPE> &region, const FPTYPE
                                  *rp)
```

**Template Function `deepmd::convert_to_phys_cpu`**

- Defined in `file_source_lib_include_region.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::convert_to_phys_cpu(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)
```

**Template Function `deepmd::convert_to_phys_gpu`**

- Defined in `file_source_lib_include_region.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::convert_to_phys_gpu(FPTYPE *rp, const Region<FPTYPE> &region, const FPTYPE *ri)
```

**Template Function `deepmd::copy_coord_cpu`**

- Defined in `file_source_lib_include_coord.h`

**Function Documentation**

```
template<typename FPTYPE>
int deepmd::copy_coord_cpu(FPTYPE *out_c, int *out_t, int *mapping, int *nall, const FPTYPE *in_c,
                           const int *in_t, const int &nloc, const int &mem_nall, const float &rcut,
                           const deepmd::Region<FPTYPE> &region)
```

**Template Function `deepmd::copy_coord_gpu`**

- Defined in `file_source_lib_include_coord.h`

**Function Documentation**

```
template<typename FPTYPE>
int deepmd::copy_coord_gpu(FPTYPE *out_c, int *out_t, int *mapping, int *nall, int *int_data, const
                           FPTYPE *in_c, const int *in_t, const int &nloc, const int &mem_nall, const
                           int &loc_cellnum, const int &total_cellnum, const int *cell_info, const
                           deepmd::Region<FPTYPE> &region)
```

**Function `deepmd::cos_switch(const double&, const double&, const double&)`**

- Defined in `file_source_lib_include_switcher.h`

**Function Documentation**

```
inline double deepmd::cos_switch(const double &xxx, const double &rmin, const double &rmax)
```

**Function `deepmd::cos_switch(double&, double&, const double&, const double&, const double&)`**

- Defined in `file_source_lib_include_switcher.h`

**Function Documentation**

```
inline void deepmd::cos_switch(double &vv, double &dd, const double &xxx, const double &rmin, const
                                double &rmax)
```

**Template Function `deepmd::cprod`**

- Defined in `file_source_lib_include_utilities.h`

**Function Documentation**

```
template<typename TYPE>
inline void deepmd::cprod(const TYPE *r0, const TYPE *r1, TYPE *r2)
```

**Function `deepmd::cum_sum`**

- Defined in `file_source_lib_include_utilities.h`

**Function Documentation**

```
void deepmd::cum_sum(std::vector<int> &zsec, const std::vector<int> &n_sel)
```

**Template Function `deepmd::delete_device_memory`**

- Defined in `file_source_lib_include_gpu_cuda.h`

## Function Documentation

```
template<typename FPTYPE>
void deepmd::delete_device_memory(FPTYPE *&device)
```

### Template Function deepmd::dot1

- Defined in file\_source\_lib\_include\_utilities.h

## Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot1(const TYPE *r0, const TYPE *r1)
```

### Template Function deepmd::dot2

- Defined in file\_source\_lib\_include\_utilities.h

## Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot2(const TYPE *r0, const TYPE *r1)
```

### Template Function deepmd::dot3

- Defined in file\_source\_lib\_include\_utilities.h

## Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot3(const TYPE *r0, const TYPE *r1)
```

### Template Function deepmd::dot4

- Defined in file\_source\_lib\_include\_utilities.h

## Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::dot4(const TYPE *r0, const TYPE *r1)
```



### Template Function `deepmd::dotmv3`

- Defined in file `_source_lib_include_utilities.h`

### Function Documentation

```
template<typename TYPE>
inline void deepmd::dotmv3(TYPE *vec_o, const TYPE *tensor, const TYPE *vec_i)
```

### Function `deepmd::DPGetDeviceCount`

- Defined in file `_source_lib_include_gpu_cuda.h`

### Function Documentation

```
inline void deepmd::DPGetDeviceCount(int &gpu_num)
```

### Function `deepmd::dprc_pairwise_map_cpu`

- Defined in file `_source_lib_include_pairwise.h`

### Function Documentation

```
void deepmd::dprc_pairwise_map_cpu(std::vector<int> &forward_qm_map, std::vector<int>
                                   &backward_qm_map, std::vector<int> &forward_qmmm_map,
                                   std::vector<int> &backward_qmmm_map, int &nloc_qm, int
                                   &nloc_qmmm, int &nall_qm, int &nall_qmmm, const
                                   std::vector<std::vector<int>> &fragments, const int nloc, const
                                   int nall)
```

DPRc pairwise map.

Parameters

- **forward\_qm\_map** – [out] Forward map for QM atoms.
- **backward\_qm\_map** – [out] Backward map for QM atoms.
- **forward\_qmmm\_map** – [out] Forward map for QM/MM atoms.
- **backward\_qmmm\_map** – [out] Backward map for QM/MM atoms.
- **nloc\_qm** – [out] The number of local QM atoms.
- **nloc\_qmmm** – [out] The number of local QM/MM atoms.
- **nall\_qm** – [out] The number of all QM atoms, including local and ghost atoms.
- **nall\_qmmm** – [out] The number of all QM/MM atoms, including local and ghost atoms.
- **fragments** – [in] The indexes of atoms that each fragment contains. Assume that only the first fragment consists of QM atoms.
- **nloc** – [in] The number of local atoms.
- **nall** – [in] The number of all atoms, including local and ghost atoms.

**Function `deepmd::DPSetDevice`**

- Defined in `file_source_lib_include_gpu_cuda.h`

**Function Documentation**

```
inline cudaError_t deepmd::DPSetDevice(int rank)
```

**Template Function `deepmd::env_mat_a_cpu`**

- Defined in `file_source_lib_include_env_mat.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::env_mat_a_cpu(std::vector<FPTYPE> &descript_a, std::vector<FPTYPE>
    &descript_a_deriv, std::vector<FPTYPE> &rij_a, const
    std::vector<FPTYPE> &posi, const std::vector<int> &type, const int
    &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec,
    const float &rmin, const float &rmax)
```

**Template Function `deepmd::env_mat_a_nvnmnd_quantize_cpu`**

- Defined in `file_source_lib_include_env_mat_nvnmnd.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::env_mat_a_nvnmnd_quantize_cpu(std::vector<FPTYPE> &descript_a,
    std::vector<FPTYPE> &descript_a_deriv,
    std::vector<FPTYPE> &rij_a, const
    std::vector<FPTYPE> &posi, const std::vector<int>
    &type, const int &i_idx, const std::vector<int>
    &fmt_nlist, const std::vector<int> &sec, const float
    &rmin, const float &rmax)
```

**Function `deepmd::env_mat_nbor_update`**

- Defined in `file_source_lib_include_prod_env_mat.h`

## Function Documentation

void deepmd::env\_mat\_nbor\_update([InputNlist](#) &inlist, [InputNlist](#) &gpu\_inlist, int &max\_nbor\_size, int \*&nbor\_list\_dev, const int \*mesh, const int size)

### Template Function deepmd::env\_mat\_r\_cpu

- Defined in file\_source\_lib\_include\_env\_mat.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::env_mat_r_cpu(std::vector<FPTYPE> &descript_a, std::vector<FPTYPE>
    &descript_a_deriv, std::vector<FPTYPE> &rij_a, const
    std::vector<FPTYPE> &posi, const std::vector<int> &type, const int
    &i_idx, const std::vector<int> &fmt_nlist_a, const std::vector<int>
    &sec_a, const float &rmin, const float &rmax)
```

### Template Function deepmd::ewald\_recp

- Defined in file\_source\_lib\_include\_ewald.h

## Function Documentation

```
template<typename VALUETYPE>
void deepmd::ewald_recp(VALUETYPE &ener, std::vector<VALUETYPE> &force,
    std::vector<VALUETYPE> &virial, const std::vector<VALUETYPE>
    &coord, const std::vector<VALUETYPE> &charge, const
    deepmd::Region<VALUETYPE> &region, const
    EwaldParameters<VALUETYPE> &param)
```

### Function deepmd::filter\_ftype\_gpu

- Defined in file\_source\_lib\_include\_neighbor\_list.h

## Function Documentation

void deepmd::filter\_ftype\_gpu(int \*ftype\_out, const int \*ftype\_in, const int nloc)

Filter the fake atom type.

If  $\geq 0$ , set to 0; if  $< 0$ , set to -1.

Parameters

- **ftype\_out** – The output filtered atom type.
- **ftype\_in** – The input atom type.
- **nloc** – The number of atoms.

### Template Function `deepmd::format_nbor_list_gpu`

- Defined in `file_source_lib_include_fmt_nlist.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::format_nbor_list_gpu(int *nlist, const FPTYPE *coord, const int *type, const
                                deepmd::InputNlist &gpu_inlist, int *array_int, uint_64
                                *array_longlong, const int max_nbor_size, const int nloc, const int
                                nall, const float rcut, const std::vector<int> sec)
```

### Template Function `deepmd::format_nlist_cpu`

- Defined in `file_source_lib_include_fmt_nlist.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::format_nlist_cpu(int *nlist, const InputNlist &in_nlist, const FPTYPE *coord, const int
                             *type, const int nloc, const int nall, const float rcut, const
                             std::vector<int> sec)
```

### Function `deepmd::free_nlist_gpu_device`

- Defined in `file_source_lib_include_neighbor_list.h`

#### Function Documentation

```
void deepmd::free_nlist_gpu_device(InputNlist &gpu_nlist)
```

Reclaim the allocated device memory of struct `InputNlist`.

Parameters

**gpu\_nlist** – Device memory `InputNlist` struct which stores the neighbor information of the core region atoms

### Template Function `deepmd::gelu_cpu`

- Defined in `file_source_lib_include_gelu.h`

## Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_cpu(FPTYPE *out, const FPTYPE *xx, const int_64 size)
```

### Template Function deepmd::gelu\_gpu

- Defined in file\_source\_lib\_include\_gelu.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_gpu(FPTYPE *out, const FPTYPE *xx, const int_64 size)
```

### Template Function deepmd::gelu\_grad\_cpu

- Defined in file\_source\_lib\_include\_gelu.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_cpu(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64 size)
```

### Template Function deepmd::gelu\_grad\_gpu

- Defined in file\_source\_lib\_include\_gelu.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_gpu(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const int_64 size)
```

### Template Function deepmd::gelu\_grad\_grad\_cpu

- Defined in file\_source\_lib\_include\_gelu.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_grad_cpu(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const
                                FPTYPE *dy_2, const int_64 size)
```

### Template Function `deepmd::gelu_grad_grad_gpu`

- Defined in `file_source_lib_include_gelu.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::gelu_grad_grad_gpu(FPTYPE *out, const FPTYPE *xx, const FPTYPE *dy, const
                                FPTYPE *dy_2, const int_64 size)
```

### Function `deepmd::group_atoms_cpu`

- Defined in `file_source_lib_include_pairwise.h`

### Function Documentation

```
void deepmd::group_atoms_cpu(std::vector<std::vector<int>> &fragments, const std::vector<int>
                              &idxs)
```

Group atoms into different fragments according to indexes.

Parameters

- **fragments** – [out] The indexes of atoms that each fragment contains. Fragment has been sorted.
- **idxs** – [in] The indexes of the fragment that each atom belongs to. -1 will be ignored.

### Template Function `deepmd::init_region_cpu`

- Defined in `file_source_lib_include_region.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::init_region_cpu(Region<FPTYPE> &region, const FPTYPE *boxt)
```

### Template Function `deepmd::invsqrt`

- Defined in `file_source_lib_include_utilities.h`

### Function Documentation

```
template<typename TYPE>
inline TYPE deepmd::invsqrt(const TYPE x)
```

#### Specialized Template Function deepmd::invsqrt< double >

- Defined in file\_source\_lib\_include\_utilities.h

### Function Documentation

```
template<>
inline double deepmd::invsqrt<double>(const double x)
```

#### Specialized Template Function deepmd::invsqrt< float >

- Defined in file\_source\_lib\_include\_utilities.h

### Function Documentation

```
template<>
inline float deepmd::invsqrt<float>(const float x)
```

#### Template Function deepmd::malloc\_device\_memory(FPTYPE \*&, const std::vector<FPTYPE>&)

- Defined in file\_source\_lib\_include\_gpu\_cuda.h

### Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory(FPTYPE *&device, const std::vector<FPTYPE> &host)
```

#### Template Function deepmd::malloc\_device\_memory(FPTYPE \*&, const int)

- Defined in file\_source\_lib\_include\_gpu\_cuda.h

### Function Documentation

```
template<typename FPTYPE>
void deepmd::malloc_device_memory(FPTYPE *&device, const int size)
```

**Template Function** `deepmd::malloc_device_memory(FPTYPE *&, std::vector<FPTYPE>&)`

- Defined in file `_source_lib_include_gpu_rocm.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::malloc_device_memory(FPTYPE *&device, std::vector<FPTYPE> &host)
```

<b>Template</b>	<b>Function</b>	<b>deepmd::malloc_device_memory_sync(FPTYPE</b>	<b>*&amp;,&amp;</b>	<b>const</b>
<b>std::vector&lt;FPTYPE&gt;&amp;)</b>				

- Defined in file `_source_lib_include_gpu_cuda.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::malloc_device_memory_sync(FPTYPE *&device, const std::vector<FPTYPE> &host)
```

**Template Function** `deepmd::malloc_device_memory_sync(FPTYPE *&, const FPTYPE *, const int)`

- Defined in file `_source_lib_include_gpu_cuda.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::malloc_device_memory_sync(FPTYPE *&device, const FPTYPE *host, const int size)
```

**Template Function** `deepmd::malloc_device_memory_sync(FPTYPE *&, std::vector<FPTYPE>&)`

- Defined in file `_source_lib_include_gpu_rocm.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::malloc_device_memory_sync(FPTYPE *&device, std::vector<FPTYPE> &host)
```



**Template Function `deepmd::map_aparam_cpu`**

- Defined in file `_source_lib_include_map_aparam.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::map_aparam_cpu(FPTYPE *output, const FPTYPE *aparam, const int *nlist, const int
                           &nloc, const int &nnei, const int &numb_aparam)
```

**Function `deepmd::max_numneigh`**

- Defined in file `_source_lib_include_neighbor_list.h`

**Function Documentation**

```
int deepmd::max_numneigh(const InputNlist &to_nlist)
```

Compute the max number of neighbors within the core region atoms.

Parameters

**to\_nlist** – `InputNlist` struct which stores the neighbor information of the core region atoms.

Return values

**max** – number of neighbors

Returns

integer

**Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, std::vector<FPTYPE>&)`**

- Defined in file `_source_lib_include_gpu_cuda.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::memcpy_device_to_host(const FPTYPE *device, std::vector<FPTYPE> &host)
```

**Template Function `deepmd::memcpy_device_to_host(const FPTYPE *, FPTYPE *, const int)`**

- Defined in file `_source_lib_include_gpu_cuda.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_device_to_host(const FPTYPE *device, FPTYPE *host, const int size)
```

**Template Function** `deepmd::memcpy_host_to_device(FPTYPE *, const std::vector<FPTYPE>&)`

- Defined in `file_source_lib_include_gpu_cuda.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_host_to_device(FPTYPE *device, const std::vector<FPTYPE> &host)
```

**Template Function** `deepmd::memcpy_host_to_device(FPTYPE *, const FPTYPE *, const int)`

- Defined in `file_source_lib_include_gpu_cuda.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_host_to_device(FPTYPE *device, const FPTYPE *host, const int size)
```

**Template Function** `deepmd::memcpy_host_to_device(FPTYPE *, std::vector<FPTYPE>&)`

- Defined in `file_source_lib_include_gpu_rocm.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::memcpy_host_to_device(FPTYPE *device, std::vector<FPTYPE> &host)
```

**Template Function** `deepmd::memset_device_memory`

- Defined in `file_source_lib_include_gpu_cuda.h`

### Function Documentation

```
template<typename FPTYPE>
void deepmd::memset_device_memory(FPTYPE *device, const int var, const int size)
```

**Template Function `deepmd::neighbor_stat_gpu`**

- Defined in file `_source_lib_include_neighbor_stat.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::neighbor_stat_gpu(const FPTYPE *coord, const int *type, const int nloc, const
                             deepmd::InputNlist &gpu_nlist, int *max_nbor_size, FPTYPE
                             *min_nbor_dist, const int ntypes, const int MAX_NNEI)
```

**Template Function `deepmd::normalize_coord_cpu`**

- Defined in file `_source_lib_include_coord.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::normalize_coord_cpu(FPTYPE *coord, const int natom, const
                                deepmd::Region<FPTYPE> &region)
```

**Template Function `deepmd::normalize_coord_gpu`**

- Defined in file `_source_lib_include_coord.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::normalize_coord_gpu(FPTYPE *coord, const int natom, const
                                deepmd::Region<FPTYPE> &region)
```

**Template Function `deepmd::pair_tab_cpu`**

- Defined in file `_source_lib_include_pair_tab.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::pair_tab_cpu(FPTYPE *energy, FPTYPE *force, FPTYPE *virial, const double
                          *table_info, const double *table_data, const FPTYPE *rij, const FPTYPE
                          *scale, const int *type, const int *nlist, const int *natoms, const
                          std::vector<int> &sel_a, const std::vector<int> &sel_r)
```

### Template Function `deepmd::prod_env_mat_a_cpu`

- Defined in `file_source_lib_include_prod_env_mat.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_a_cpu(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const
                                FPTYPE *coord, const int *type, const InputNlist &inlist, const int
                                max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int
                                nloc, const int nall, const float rcut, const float rcut_smth, const
                                std::vector<int> sec, const int *f_type = NULL)
```

### Template Function `deepmd::prod_env_mat_a_gpu`

- Defined in `file_source_lib_include_prod_env_mat.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_a_gpu(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const
                                FPTYPE *coord, const int *type, const InputNlist &gpu_inlist, int
                                *array_int, unsigned long long *array_longlong, const int
                                max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int
                                nloc, const int nall, const float rcut, const float rcut_smth, const
                                std::vector<int> sec, const int *f_type = NULL)
```

### Template Function `deepmd::prod_env_mat_a_nvnmmd_quantize_cpu`

- Defined in `file_source_lib_include_prod_env_mat_nvnmmd.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_a_nvnmmd_quantize_cpu(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij,
                                                  int *nlist, const FPTYPE *coord, const int *type,
                                                  const InputNlist &inlist, const int max_nbor_size,
                                                  const FPTYPE *avg, const FPTYPE *std, const int
                                                  nloc, const int nall, const float rcut, const float
                                                  rcut_smth, const std::vector<int> sec, const int
                                                  *f_type = NULL)
```

### Template Function `deepmd::prod_env_mat_r_cpu`

- Defined in `file_source_lib_include_prod_env_mat.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_r_cpu(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const
                                FPTYPE *coord, const int *type, const InputNlist &inlist, const int
                                max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int
                                nloc, const int nall, const float rcut, const float rcut_smth, const
                                std::vector<int> sec)
```

### Template Function `deepmd::prod_env_mat_r_gpu`

- Defined in `file_source_lib_include_prod_env_mat.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_env_mat_r_gpu(FPTYPE *em, FPTYPE *em_deriv, FPTYPE *rij, int *nlist, const
                                FPTYPE *coord, const int *type, const InputNlist &gpu_inlist, int
                                *array_int, unsigned long long *array_longlong, const int
                                max_nbor_size, const FPTYPE *avg, const FPTYPE *std, const int
                                nloc, const int nall, const float rcut, const float rcut_smth, const
                                std::vector<int> sec)
```

### Template Function `deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int)`

- Defined in `file_source_lib_include_prod_force.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_a_cpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv,
                              const int *nlist, const int nloc, const int nall, const int nnei, const int
                              nframes)
```

Produce force from `net_deriv` and `in_deriv`.

Template Parameters

**FPTYPE** – float or double

Parameters

- **force** – [out] Atomic forces.
- **net\_deriv** – [in] Net derivative.
- **in\_deriv** – [in] Environmental derivative.

- **nlist** – [in] Neighbor list.
- **nloc** – [in] The number of local atoms.
- **nall** – [in] The number of all atoms, including ghost atoms.
- **nnei** – [in] The number of neighbors.
- **nframes** – [in] The number of frames.

**Template Function `deepmd::prod_force_a_cpu(FPTYPE *, const FPTYPE *, const FPTYPE *, const int *, const int, const int, const int, const int, const int, const int)`**

- Defined in file `_source_lib_include_prod_force.h`

## Function Documentation

```
template<typename FPTYPE>
void deepmd::prod_force_a_cpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv,
                             const int *nlist, const int nloc, const int nall, const int nnei, const int
                             nframes, const int thread_nloc, const int thread_start_index)
```

Produce force from `net_deriv` and `in_deriv`.

This function is used for multi-threading. Only part of atoms are computed in this thread. They will be computed in parallel.

Template Parameters

**FPTYPE** – float or double

Parameters

- **force** – [out] Atomic forces.
- **net\_deriv** – [in] Net derivative.
- **in\_deriv** – [in] Environmental derivative.
- **nlist** – [in] Neighbor list.
- **nloc** – [in] The number of local atoms.
- **nall** – [in] The number of all atoms, including ghost atoms.
- **nnei** – [in] The number of neighbors.
- **nframes** – [in] The number of frames.
- **thread\_nloc** – [in] The number of local atoms to be computed in this thread.
- **thread\_start\_index** – [in] The start index of local atoms to be computed in this thread. The index should be in `[0, nloc)`.

**Template Function `deepmd::prod_force_a_gpu`**

- Defined in file `_source_lib_include_prod_force.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_force_a_gpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv,
                             const int *nlist, const int nloc, const int nall, const int nnei, const int
                             nframes)
```

**Template Function `deepmd::prod_force_grad_a_cpu`**

- Defined in file `_source_lib_include_prod_force_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_force_grad_a_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const int *nlist, const int nloc, const int nnei, const int
                                   nframes)
```

**Template Function `deepmd::prod_force_grad_a_gpu`**

- Defined in file `_source_lib_include_prod_force_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_force_grad_a_gpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const int *nlist, const int nloc, const int nnei, const int
                                   nframes)
```

**Template Function `deepmd::prod_force_grad_r_cpu`**

- Defined in file `_source_lib_include_prod_force_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_force_grad_r_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const int *nlist, const int nloc, const int nnei, const int
                                   nframes)
```

**Template Function `deepmd::prod_force_grad_r_gpu`**

- Defined in file `_source_lib_include_prod_force_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_force_grad_r_gpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const int *nlist, const int nloc, const int nnei, const int
                                   nframes)
```

**Template Function `deepmd::prod_force_r_cpu`**

- Defined in file `_source_lib_include_prod_force.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_force_r_cpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv,
                              const int *nlist, const int nloc, const int nall, const int nnei, const int
                              nframes)
```

**Template Function `deepmd::prod_force_r_gpu`**

- Defined in file `_source_lib_include_prod_force.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_force_r_gpu(FPTYPE *force, const FPTYPE *net_deriv, const FPTYPE *in_deriv,
                              const int *nlist, const int nloc, const int nall, const int nnei, const int
                              nframes)
```

**Template Function `deepmd::prod_virial_a_cpu`**

- Defined in file `_source_lib_include_prod_virial.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_a_cpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv,
                              const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const
                              int nloc, const int nall, const int nnei)
```



**Template Function `deepmd::prod_virial_a_gpu`**

- Defined in file `_source_lib_include_prod_virial.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_a_gpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv,
                              const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const
                              int nloc, const int nall, const int nnei)
```

**Template Function `deepmd::prod_virial_grad_a_cpu`**

- Defined in file `_source_lib_include_prod_virial_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_a_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                    *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc,
                                    const int nnei)
```

**Template Function `deepmd::prod_virial_grad_a_gpu`**

- Defined in file `_source_lib_include_prod_virial_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_a_gpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                    *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc,
                                    const int nnei)
```

**Template Function `deepmd::prod_virial_grad_r_cpu`**

- Defined in file `_source_lib_include_prod_virial_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_r_cpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                    *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc,
                                    const int nnei)
```

**Template Function `deepmd::prod_virial_grad_r_gpu`**

- Defined in file `_source_lib_include_prod_virial_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_grad_r_gpu(FPTYPE *grad_net, const FPTYPE *grad, const FPTYPE
                                   *env_deriv, const FPTYPE *rij, const int *nlist, const int nloc,
                                   const int nnei)
```

**Template Function `deepmd::prod_virial_r_cpu`**

- Defined in file `_source_lib_include_prod_virial.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_r_cpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv,
                              const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const
                              int nloc, const int nall, const int nnei)
```

**Template Function `deepmd::prod_virial_r_gpu`**

- Defined in file `_source_lib_include_prod_virial.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::prod_virial_r_gpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE *net_deriv,
                              const FPTYPE *env_deriv, const FPTYPE *rij, const int *nlist, const
                              int nloc, const int nall, const int nnei)
```

**Template Function `deepmd::soft_min_switch_cpu`**

- Defined in file `_source_lib_include_soft_min_switch.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::soft_min_switch_cpu(FPTYPE *sw_value, FPTYPE *sw_deriv, const FPTYPE *rij, const
                                 int *nlist, const int &nloc, const int &nnei, const FPTYPE &alpha,
                                 const FPTYPE &rmin, const FPTYPE &rmax)
```

**Template Function `deepmd::soft_min_switch_force_cpu`**

- Defined in file `_source_lib_include_soft_min_switch_force.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::soft_min_switch_force_cpu(FPTYPE *force, const FPTYPE *du, const FPTYPE
                                     *sw_deriv, const int *nlist, const int nloc, const int nall, const
                                     int nnei)
```

**Template Function `deepmd::soft_min_switch_force_grad_cpu`**

- Defined in file `_source_lib_include_soft_min_switch_force_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::soft_min_switch_force_grad_cpu(FPTYPE *grad_net, const FPTYPE *grad, const
                                           FPTYPE *sw_deriv, const int *nlist, const int nloc,
                                           const int nnei)
```

**Template Function `deepmd::soft_min_switch_virial_cpu`**

- Defined in file `_source_lib_include_soft_min_switch_virial.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::soft_min_switch_virial_cpu(FPTYPE *virial, FPTYPE *atom_virial, const FPTYPE
                                       *du, const FPTYPE *sw_deriv, const FPTYPE *rij, const int
                                       *nlist, const int nloc, const int nall, const int nnei)
```

**Template Function `deepmd::soft_min_switch_virial_grad_cpu`**

- Defined in file `_source_lib_include_soft_min_switch_virial_grad.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::soft_min_switch_virial_grad_cpu(FPTYPE *grad_net, const FPTYPE *grad, const
                                             FPTYPE *sw_deriv, const FPTYPE *rij, const int
                                             *nlist, const int nloc, const int nnei)
```

### Function `deepmd::spline3_switch`

- Defined in `file_source_lib_include_switcher.h`

#### Function Documentation

```
inline void deepmd::spline3_switch(double &vv, double &dd, const double &xx, const double &rmin,
                                   const double &rmax)
```

### Template Function `deepmd::spline5_switch`

- Defined in `file_source_lib_include_switcher.h`

#### Function Documentation

```
template<typename FPTYPE>
inline void deepmd::spline5_switch(FPTYPE &vv, FPTYPE &dd, const FPTYPE &xx, const float
                                   &rmin, const float &rmax)
```

### Template Function `deepmd::tabulate_fusion_se_a_cpu`

- Defined in `file_source_lib_include_tabulate.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_cpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                       *table_info, const FPTYPE *em_x, const FPTYPE *em, const
                                       FPTYPE *two_embed, const int nloc, const int nnei, const int
                                       last_layer_size, const bool is_sorted = true)
```

### Template Function `deepmd::tabulate_fusion_se_a_gpu`

- Defined in `file_source_lib_include_tabulate.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_gpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                       *table_info, const FPTYPE *em_x, const FPTYPE *em, const
                                       FPTYPE *two_embed, const int nloc, const int nnei, const int
                                       last_layer_size, const bool is_sorted = true)
```

**Template Function `deepmd::tabulate_fusion_se_a_grad_cpu`**

- Defined in file `_source_lib_include_tabulate.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_cpu(FPTYPE *dy_dem_x, FPTYPE *dy_dem, FPTYPE
                                           *dy_dtwo, const FPTYPE *table, const FPTYPE
                                           *table_info, const FPTYPE *em_x, const FPTYPE *em,
                                           const FPTYPE *two_embed, const FPTYPE *dy, const
                                           int nloc, const int nnei, const int last_layer_size, const
                                           bool is_sorted = true)
```

**Template Function `deepmd::tabulate_fusion_se_a_grad_gpu`**

- Defined in file `_source_lib_include_tabulate.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_gpu(FPTYPE *dy_dem_x, FPTYPE *dy_dem, FPTYPE
                                           *dy_dtwo, const FPTYPE *table, const FPTYPE
                                           *table_info, const FPTYPE *em_x, const FPTYPE *em,
                                           const FPTYPE *two_embed, const FPTYPE *dy, const
                                           int nloc, const int nnei, const int last_layer_size, const
                                           bool is_sorted = true)
```

**Template Function `deepmd::tabulate_fusion_se_a_grad_grad_cpu`**

- Defined in file `_source_lib_include_tabulate.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const
                                                  FPTYPE *table_info, const FPTYPE *em_x, const
                                                  FPTYPE *em, const FPTYPE *two_embed, const
                                                  FPTYPE *dz_dy_dem_x, const FPTYPE
                                                  *dz_dy_dem, const FPTYPE *dz_dy_dtwo, const
                                                  int nloc, const int nnei, const int last_layer_size,
                                                  const bool is_sorted = true)
```

**Template Function `deepmd::tabulate_fusion_se_a_grad_grad_gpu`**

- Defined in `file_source_lib_include_tabulate.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_a_grad_grad_gpu(FPTYPE *dz_dy, const FPTYPE *table, const
FPTYPE *table_info, const FPTYPE *em_x, const
FPTYPE *em, const FPTYPE *two_embed, const
FPTYPE *dz_dy_dem_x, const FPTYPE
*dz_dy_dem, const FPTYPE *dz_dy_dtwo, const
int nloc, const int nnei, const int last_layer_size,
const bool is_sorted = true)
```

**Template Function `deepmd::tabulate_fusion_se_r_cpu`**

- Defined in `file_source_lib_include_tabulate.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_cpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
*table_info, const FPTYPE *em, const int nloc, const int nnei,
const int last_layer_size)
```

**Template Function `deepmd::tabulate_fusion_se_r_gpu`**

- Defined in `file_source_lib_include_tabulate.h`

**Function Documentation**

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_gpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
*table_info, const FPTYPE *em, const int nloc, const int nnei,
const int last_layer_size)
```

**Template Function `deepmd::tabulate_fusion_se_r_grad_cpu`**

- Defined in `file_source_lib_include_tabulate.h`

## Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_cpu(FPTYPE *dy_dem, const FPTYPE *table, const
                                           FPTYPE *table_info, const FPTYPE *em, const
                                           FPTYPE *dy, const int nloc, const int nnei, const int
                                           last_layer_size)
```

### Template Function deepmd::tabulate\_fusion\_se\_r\_grad\_gpu

- Defined in file\_source\_lib\_include\_tabulate.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_gpu(FPTYPE *dy_dem, const FPTYPE *table, const
                                           FPTYPE *table_info, const FPTYPE *em, const
                                           FPTYPE *dy, const int nloc, const int nnei, const int
                                           last_layer_size)
```

### Template Function deepmd::tabulate\_fusion\_se\_r\_grad\_grad\_cpu

- Defined in file\_source\_lib\_include\_tabulate.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const
                                                FPTYPE *table_info, const FPTYPE *em, const
                                                FPTYPE *dz_dy_dem, const int nloc, const int
                                                nnei, const int last_layer_size)
```

### Template Function deepmd::tabulate\_fusion\_se\_r\_grad\_grad\_gpu

- Defined in file\_source\_lib\_include\_tabulate.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_r_grad_grad_gpu(FPTYPE *dz_dy, const FPTYPE *table, const
                                                FPTYPE *table_info, const FPTYPE *em, const
                                                FPTYPE *dz_dy_dem, const int nloc, const int
                                                nnei, const int last_layer_size)
```

### Template Function `deepmd::tabulate_fusion_se_t_cpu`

- Defined in `file_source_lib_include_tabulate.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_cpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                     *table_info, const FPTYPE *em_x, const FPTYPE *em, const
                                     int nloc, const int nnei_i, const int nnei_j, const int
                                     last_layer_size)
```

### Template Function `deepmd::tabulate_fusion_se_t_gpu`

- Defined in `file_source_lib_include_tabulate.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_gpu(FPTYPE *out, const FPTYPE *table, const FPTYPE
                                     *table_info, const FPTYPE *em_x, const FPTYPE *em, const
                                     int nloc, const int nnei_i, const int nnei_j, const int
                                     last_layer_size)
```

### Template Function `deepmd::tabulate_fusion_se_t_grad_cpu`

- Defined in `file_source_lib_include_tabulate.h`

#### Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_cpu(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const
                                           FPTYPE *table, const FPTYPE *table_info, const
                                           FPTYPE *em_x, const FPTYPE *em, const FPTYPE
                                           *dy, const int nloc, const int nnei_i, const int nnei_j,
                                           const int last_layer_size)
```

### Template Function `deepmd::tabulate_fusion_se_t_grad_gpu`

- Defined in `file_source_lib_include_tabulate.h`



## Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_gpu(FPTYPE *dy_dem_x, FPTYPE *dy_dem, const
FPTYPE *table, const FPTYPE *table_info, const
FPTYPE *em_x, const FPTYPE *em, const FPTYPE
*dy, const int nloc, const int nnei_i, const int nnei_j,
const int last_layer_size)
```

### Template Function deepmd::tabulate\_fusion\_se\_t\_grad\_grad\_cpu

- Defined in file\_source\_lib\_include\_tabulate.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_grad_cpu(FPTYPE *dz_dy, const FPTYPE *table, const
FPTYPE *table_info, const FPTYPE *em_x, const
FPTYPE *em, const FPTYPE *dz_dy_dem_x,
const FPTYPE *dz_dy_dem, const int nloc, const
int nnei_i, const int nnei_j, const int
last_layer_size)
```

### Template Function deepmd::tabulate\_fusion\_se\_t\_grad\_grad\_gpu

- Defined in file\_source\_lib\_include\_tabulate.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::tabulate_fusion_se_t_grad_grad_gpu(FPTYPE *dz_dy, const FPTYPE *table, const
FPTYPE *table_info, const FPTYPE *em_x, const
FPTYPE *em, const FPTYPE *dz_dy_dem_x,
const FPTYPE *dz_dy_dem, const int nloc, const
int nnei_i, const int nnei_j, const int
last_layer_size)
```

### Template Function deepmd::test\_encoding\_decoding\_nbor\_info\_gpu

- Defined in file\_source\_lib\_include\_fmt\_nlist.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::test_encoding_decoding_nbor_info_gpu(uint_64 *key, int *out_type, int *out_index,
                                                    const int *in_type, const FPTYPE *in_dist,
                                                    const int *in_index, const int size_of_array)
```

## Function `deepmd::use_nei_info_cpu`

- Defined in `file_source_lib_include_neighbor_list.h`

## Function Documentation

```
void deepmd::use_nei_info_cpu(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map,
                              const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)
```

## Function `deepmd::use_nei_info_gpu`

- Defined in `file_source_lib_include_neighbor_list.h`

## Function Documentation

```
void deepmd::use_nei_info_gpu(int *nlist, int *ntype, bool *nmask, const int *type, const int *nlist_map,
                              const int nloc, const int nnei, const int ntypes, const bool b_nlist_map)
```

## Function `deepmd::use_nlist_map`

- Defined in `file_source_lib_include_neighbor_list.h`

## Function Documentation

```
void deepmd::use_nlist_map(int *nlist, const int *nlist_map, const int nloc, const int nnei)
```

## Template Function `deepmd::volume_cpu`

- Defined in `file_source_lib_include_region.h`

## Function Documentation

```
template<typename FPTYPE>
FPTYPE deepmd::volume_cpu(const Region<FPTYPE> &region)
```

## Template Function deepmd::volume\_gpu

- Defined in file\_source\_lib\_include\_region.h

## Function Documentation

```
template<typename FPTYPE>
void deepmd::volume_gpu(FPTYPE *volume, const Region<FPTYPE> &region)
```

## Template Function dotmul\_flt\_nvnmmd

- Defined in file\_source\_lib\_include\_env\_mat\_nvnmmd.h

## Function Documentation

```
template<class T>
void dotmul_flt_nvnmmd(T &y, T *x1, T *x2, int64_t M)
```

## Function DPAssert(cudaError\_t, const char \*, int, bool)

- Defined in file\_source\_lib\_include\_gpu\_cuda.h

## Function Documentation

```
inline void DPAssert(cudaError_t code, const char *file, int line, bool abort = true)
```

## Function DPAssert(hipError\_t, const char \*, int, bool)

- Defined in file\_source\_lib\_include\_gpu\_rocm.h

## Function Documentation

```
inline void DPAssert(hipError_t code, const char *file, int line, bool abort = true)
```

### Function `env_mat_a`

- Defined in file `_source_lib_include_env_mat.h`

### Function Documentation

```
void env_mat_a(std::vector<double> &descript_a, std::vector<double> &descript_a_deriv,  
               std::vector<double> &rij_a, const std::vector<double> &posi, const int &ntypes, const  
               std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc,  
               const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const  
               double &rmin, const double &rmax)
```

### Function `env_mat_r`

- Defined in file `_source_lib_include_env_mat.h`

### Function Documentation

```
void env_mat_r(std::vector<double> &descript_r, std::vector<double> &descript_r_deriv,  
               std::vector<double> &rij_r, const std::vector<double> &posi, const int &ntypes, const  
               std::vector<int> &type, const SimulationRegion<double> &region, const bool &b_pbc,  
               const int &i_idx, const std::vector<int> &fmt_nlist, const std::vector<int> &sec, const  
               double &rmin, const double &rmax)
```

### Template Function `find_max_expo(int64_t&, T *, int64_t)`

- Defined in file `_source_lib_include_env_mat_nvnmd.h`

### Function Documentation

```
template<class T>  
void find_max_expo(int64_t &max_expo, T *x, int64_t M)
```

### Template Function `find_max_expo(int64_t&, T *, int64_t, int64_t)`

- Defined in file `_source_lib_include_env_mat_nvnmd.h`

### Function Documentation

```
template<class T>  
void find_max_expo(int64_t &max_expo, T *x, int64_t N, int64_t M)
```

### Template Function `format_nlist_i_cpu`

- Defined in `file_source_lib_include_fmt_nlist.h`

#### Function Documentation

```
template<typename FPTYPE>
int format_nlist_i_cpu(std::vector<int> &fmt_nei_idx_a, const std::vector<FPTYPE> &posi, const
    std::vector<int> &type, const int &i_idx, const std::vector<int> &nei_idx_a,
    const float &rcut, const std::vector<int> &sec_a)
```

### Function `format_nlist_i_fill_a`

- Defined in `file_source_lib_include_fmt_nlist.h`

#### Function Documentation

```
int format_nlist_i_fill_a(std::vector<int> &fmt_nei_idx_a, std::vector<int> &fmt_nei_idx_r, const
    std::vector<double> &posi, const int &ntypes, const std::vector<int> &type,
    const SimulationRegion<double> &region, const bool &b_pbc, const int
    &i_idx, const std::vector<int> &nei_idx_a, const std::vector<int>
    &nei_idx_r, const double &rcut, const std::vector<int> &sec_a, const
    std::vector<int> &sec_r)
```

### Template Function `mul_flt_nvnmmd`

- Defined in `file_source_lib_include_env_mat_nvnmmd.h`

#### Function Documentation

```
template<class T>
void mul_flt_nvnmmd(T &y, T x1, T x2)
```

### Function `nborAssert(cudaError_t, const char *, int, bool)`

- Defined in `file_source_lib_include_gpu_cuda.h`

#### Function Documentation

```
inline void nborAssert(cudaError_t code, const char *file, int line, bool abort = true)
```

**Function `nborAssert(hipError_t, const char *, int, bool)`**

- Defined in `file_source_lib_include_gpu_rocm.h`

**Function Documentation**

```
inline void nborAssert(hipError_t code, const char *file, int line, bool abort = true)
```

**Function `omp_get_num_threads`**

- Defined in `file_source_lib_include_ewald.h`

**Function Documentation**

```
int omp_get_num_threads()
```

**Function `omp_get_thread_num`**

- Defined in `file_source_lib_include_ewald.h`

**Function Documentation**

```
int omp_get_thread_num()
```

**Template Function `splitflt`**

- Defined in `file_source_lib_include_env_mat_nvnmd.h`

**Function Documentation**

```
template<class T>  
void splitflt(T x, int64_t &sign, int64_t &expo, int64_t &mant)
```

## 22.3.5 Variables

**Variable `deepmd::ElectrostaticConversion`**

- Defined in `file_source_lib_include_ewald.h`

## Variable Documentation

`const double deepmd::ElectrostaticConversion = 14.39964535475696995031`

## 22.3.6 Defines

### Define DPErrcheck

- Defined in file `_source_lib_include_gpu_cuda.h`

### Define Documentation

`DPErrcheck(res)`

### Define DPErrcheck

- Defined in file `_source_lib_include_gpu_rocm.h`

### Define Documentation

`DPErrcheck(res)`

### Define FLT\_MASK

- Defined in file `_source_lib_include_env_mat_nvnmd.h`

### Define Documentation

`FLT_MASK`

### Define GPU\_MAX\_NBOR\_SIZE

- Defined in file `_source_lib_include_gpu_cuda.h`

### Define Documentation

`GPU_MAX_NBOR_SIZE`

### Define GPU\_MAX\_NBOR\_SIZE

- Defined in file\_source\_lib\_include\_gpu\_rocm.h

### Define Documentation

GPU\_MAX\_NBOR\_SIZE

### Define gpuDeviceSynchronize

- Defined in file\_source\_lib\_include\_gpu\_cuda.h

### Define Documentation

gpuDeviceSynchronize

### Define gpuDeviceSynchronize

- Defined in file\_source\_lib\_include\_gpu\_rocm.h

### Define Documentation

gpuDeviceSynchronize

### Define gpuGetLastError

- Defined in file\_source\_lib\_include\_gpu\_cuda.h

### Define Documentation

gpuGetLastError

### Define gpuGetLastError

- Defined in file\_source\_lib\_include\_gpu\_rocm.h



### Define Documentation

`gpuGetLastError`

### Define `gpuMemcpy`

- Defined in file `_source_lib_include_gpu_cuda.h`

### Define Documentation

`gpuMemcpy`

### Define `gpuMemcpy`

- Defined in file `_source_lib_include_gpu_rocm.h`

### Define Documentation

`gpuMemcpy`

### Define `gpuMemcpyDeviceToDevice`

- Defined in file `_source_lib_include_gpu_cuda.h`

### Define Documentation

`gpuMemcpyDeviceToDevice`

### Define `gpuMemcpyDeviceToDevice`

- Defined in file `_source_lib_include_gpu_rocm.h`

### Define Documentation

`gpuMemcpyDeviceToDevice`

### Define `gpuMemcpyDeviceToHost`

- Defined in file `_source_lib_include_gpu_cuda.h`

### Define Documentation

`gpuMemcpyDeviceToHost`

### Define `gpuMemcpyDeviceToHost`

- Defined in file `_source_lib_include_gpu_rocm.h`

### Define Documentation

`gpuMemcpyDeviceToHost`

### Define `gpuMemcpyHostToDevice`

- Defined in file `_source_lib_include_gpu_cuda.h`

### Define Documentation

`gpuMemcpyHostToDevice`

### Define `gpuMemcpyHostToDevice`

- Defined in file `_source_lib_include_gpu_rocm.h`

### Define Documentation

`gpuMemcpyHostToDevice`

### Define `gpuMemset`

- Defined in file `_source_lib_include_gpu_cuda.h`

### Define Documentation

`gpuMemset`

### Define `gpuMemset`

- Defined in `file_source_lib_include_gpu_rocm.h`

### Define Documentation

`gpuMemset`

### Define `MOASPNDIM`

- Defined in `file_source_lib_include_SimulationRegion.h`

### Define Documentation

`MOASPNDIM`

### Define `NBIT_CUTF`

- Defined in `file_source_lib_include_env_mat_nvnmd.h`

### Define Documentation

`NBIT_CUTF`

### Define `NBIT_FLTF`

- Defined in `file_source_lib_include_env_mat_nvnmd.h`

### Define Documentation

`NBIT_FLTF`

**Define nborErrcheck**

- Defined in file\_source\_lib\_include\_gpu\_cuda.h

**Define Documentation**

**nborErrcheck**(res)

**Define nborErrcheck**

- Defined in file\_source\_lib\_include\_gpu\_rocm.h

**Define Documentation**

**nborErrcheck**(res)

**Define SQRT\_2\_PI**

- Defined in file\_source\_lib\_include\_device.h

**Define Documentation**

**SQRT\_2\_PI**

**Define TPB**

- Defined in file\_source\_lib\_include\_device.h

**Define Documentation**

**TPB**

## 22.3.7 Typedefs

**Typedef int\_64**

- Defined in file\_source\_lib\_include\_device.h

### Typedef Documentation

typedef long long **int\_64**

### Typedef **uint\_64**

- Defined in file\_source\_lib\_include\_device.h

### Typedef Documentation

typedef unsigned long long **uint\_64**



## LICENSE

The project DeePMD-kit is licensed under [GNU LGPLv3.0](#).





## AUTHORS AND CREDITS

### 24.1 Cite DeePMD-kit and methods

- For general purpose,
- If GPU version is used,
- If local frame (`loc_frame`) is used,
- If DeepPot-SE (`se_e2_a`, `se_e2_r`, `se_e3`, `se_atten`) is used,
- If three-body embedding DeepPot-SE (`se_e3`) is used,
- If attention-based descriptor (`se_atten`, `se_atten_v2`) is used,
- If frame-specific parameters (`fparam`, e.g. electronic temperature) is used,
- If atom-specific parameters (`aparam`, e.g. electronic temperature) is used,
- If fitting dipole,
- If fitting polarizability,
- If fitting density of states,
- If fitting relative energies,
- If DPLR is used, or `se_e2_r` and `hybrid` are used,
- If DPRc is used,
- If interpolation with a pair-wise potential is used,
- If the model is compressed (`dp_compress`),
- If model deviation is computed,
- If relative or atomic model deviation is computed,
- If NVNMD is used,

## 24.2 Package Contributors

- AngelJia
- AnguseZhang
- Anurag Kumar Singh
- Chenqqian Zhang
- Chenxing Luo
- Chun Cai
- Davide Tisi
- Denghui Lu
- Duo
- Eisuke Kawashima
- Futaki Haduki
- GeiduanLiu
- Han Wang
- Hananeh Oliaei
- Harvey Que
- HuangJiameng
- HydrogenSulfate
- Jia-Xin Zhu
- Jiequn Han
- Jingchao Zhang
- Jinzhe Zeng
- Koki MURAOKA
- LiangWenshuo1118
- Linfeng Zhang
- LiuGroupHNU
- Lu
- Lysithea
- Marián Rynik
- Nick Lin
- Rhys Goodall
- Shaochen Shi
- TrellixVulnTeam
- Wanrun Jiang
- Xia, Yu

- YWolfeee
- Ye Ding
- Yifan Li
- Yingze Wang
- Yixiao Chen
- Zeyu Li
- ZhengdQin
- ZiyaoLi
- baohan
- bwang-ecnu
- deepmodeling
- denghuilu
- dependabot[bot]
- haidi
- hlyang
- hsulab
- hztttt
- iProzd
- imgbot[bot]
- jxxiaoshaoye
- liangadam
- likefallwind
- link89
- marian-code
- mingzhong15
- nahso
- njzjz
- pkulzy
- pre-commit-ci[bot]
- readthedocs-assistant
- sigbjobo
- tuoping
- wsyxbcl
- ziyao

## 24.3 Other Credits

- Zhang ZiXuan for designing the Deepmodeling logo.
- Everyone on the Deepmodeling mailing list for contributing to many discussions and decisions!

## LOGO

The logo of DeePMD-kit is a beaver. Beavers were widely distributed in Europe and Asia but became nearly extinct due to hunting. Listed as a first-class state-protected animal in China, the population of beavers in China is less than the giant pandas. We hope that users of DeePMD-kit can enhance the awareness to protect beavers.

- [genindex](#)
- [modindex](#)
- [search](#)



## BIBLIOGRAPHY

- [1] Jinzhe Zeng, Timothy J. Giese, Sölen Ekesan, Darrin M. York, Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution, *J. Chem. Theory Comput.*, 2021, 17 (11), 6993-7009.
- [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4441-4451.
- [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4441-4451.
- [1] Jinzhe Zeng, Timothy J. Giese, Sölen Ekesan, Darrin M. York, Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution, *J. Chem. Theory Comput.*, 2021, 17 (11), 6993-7009.
- [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4441-4451.
- [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4441-4451.
- [1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision - ECCV 2016*, pages 630-645. Springer International Publishing, 2016.
- [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4441-4451.
- [1] Linfeng Zhang, Jiequn Han, Han Wang, Wissam A. Saidi, Roberto Car, and E. Weinan. 2018. End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4441-4451.
- [WZHE18] Han Wang, Linfeng Zhang, Jiequn Han, and Weinan E. DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics. *Comput. Phys. Comm.*, 228:178–184, jul 2018. doi:10.1016/j.cpc.2018.03.016.

- [ZZL+23] Jinzhe Zeng, Duo Zhang, Denghui Lu, Pinghui Mo, Zeyu Li, Yixiao Chen, Marián Rynik, Li'ang Huang, Ziyao Li, Shaochen Shi, Yingze Wang, Haotian Ye, Ping Tuo, Jiabin Yang, Ye Ding, Yifan Li, Davide Tisi, Qiyu Zeng, Han Bao, Yu Xia, Jiameng Huang, Koki Muraoka, Yibo Wang, Junhan Chang, Fengbo Yuan, Sigbjørn Løland Bore, Chun Cai, Yinnian Lin, Bo Wang, Jiayan Xu, Jia-Xin Zhu, Chenxing Luo, Yuzhi Zhang, Rhys E A Goodall, Wenshuo Liang, Anurag Kumar Singh, Sikai Yao, Jingchao Zhang, Renata Wentzcovitch, Jiequn Han, Jie Liu, Weile Jia, Darrin M York, Weinan E, Roberto Car, Linfeng Zhang, and Han Wang. DeePMD-kit v2: A software package for deep potential models. *J. Chem. Phys.*, 159:054801, 2023. doi:10.1063/5.0155600.
- [LWC+21] Denghui Lu, Han Wang, Mohan Chen, Lin Lin, Roberto Car, Weinan E, Weile Jia, and Linfeng Zhang. 86 PFLOPS Deep Potential Molecular Dynamics simulation of 100 million atoms with ab initio accuracy. *Comput. Phys. Comm.*, 259:107624, 2021. doi:10.1016/j.cpc.2020.107624.
- [ZHW+18] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. *Phys. Rev. Lett.*, 120(14):143001, 2018. doi:10.1103/PhysRevLett.120.143001.
- [ZHW+18] Linfeng Zhang, Jiequn Han, Han Wang, Wissam Saidi, Roberto Car, and Weinan E. End-to-end Symmetry Preserving Inter-atomic Potential Energy Model for Finite and Extended Systems. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 4436–4446. Curran Associates, Inc., 2018. URL: <https://dl.acm.org/doi/10.5555/3327345.3327356>.
- [WWZ+22] Xiaoyang Wang, Yinan Wang, Linfeng Zhang, Fuzhi Dai, and Han Wang. A tungsten deep neural-network potential for simulating mechanical property degradation under fusion service environment. *Nucl. Fusion*, 62:126013, 2022. doi:10.1088/1741-4326/ac888b.
- [ZBD+22] Duo Zhang, Hangrui Bi, Fu-Zhi Dai, Wanrun Jiang, Linfeng Zhang, and Han Wang. DPA-1: Pretraining of Attention-based Deep Potential Model for Molecular Simulation. 2022. doi:10.48550/arXiv.2208.08236.
- [ZGL+20] Yuzhi Zhang, Chang Gao, Qianrui Liu, Linfeng Zhang, Han Wang, and Mohan Chen. Warm dense matter simulation via electron temperature dependent deep potential molecular dynamics. *Phys. Plasmas*, 27(12):122704, 12 2020. doi:10.1063/5.0023265.
- [ZCZ+23] Qiyu Zeng, Bo Chen, Shen Zhang, Dongdong Kang, Han Wang, Xiaoxiang Yu, and Jiayu Dai. Full-scale ab initio simulations of laser-driven atomistic dynamics. 2023. doi:10.48550/arXiv.2308.13863.
- [ZCW+20] Linfeng Zhang, Mohan Chen, Xifan Wu, Han Wang, Weinan E, and Roberto Car. Deep neural network for the dielectric response of insulators. *Phys. Rev. B*, 102(4):041121, 2020. doi:10.1103/PhysRevB.102.041121.
- [SAZ+20] Grace M Sommers, Marcos F Calegari Andrade, Linfeng Zhang, Han Wang, and Roberto Car. Raman spectrum and polarizability of liquid water from deep neural networks. *Phys. Chem. Chem. Phys.*, 22(19):10592–10602, 2020. doi:10.1039/D0CP01893G.
- [ZCY+22] Qiyu Zeng, Bo Chen, Xiaoxiang Yu, Shen Zhang, Dongdong Kang, Han Wang, and Jiayu Dai. Towards large-scale and spatiotemporally resolved diagnosis of electronic density of states by deep learning. *Phys. Rev. B*, 105:174109, 2022. doi:10.1103/PhysRevB.105.174109.
- [ZTGY23] Jinzhe Zeng, Yujun Tao, Timothy J Giese, and Darrin M York. QD $\pi$ : A Quantum Deep Potential Interaction Model for Drug Discovery. *J. Chem. Theory Comput.*, 19:1261–1275, 2023. doi:10.1021/acs.jctc.2c01172.
- [ZWM+22] Linfeng Zhang, Han Wang, Maria Carolina Muniz, Athanassios Z Panagiotopoulos, Roberto Car, and Weinan E. A deep potential model with long-range electrostatic interactions. *J. Chem. Phys.*, 156:124107, 2022. doi:10.1063/5.0083669.



- [ZGEY21] Jinzhe Zeng, Timothy J Giese, Şölen Ekesan, and Darrin M York. Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/molecular Mechanical Simulations of Chemical Reactions in Solution. *J. Chem. Theory Comput.*, 17:6993–7009, 2021. doi:10.1021/acs.jctc.1c00201.
- [WGZ+19] Hao Wang, Xun Guo, Linfeng Zhang, Han Wang, and Jianming Xue. Deep learning inter-atomic potential model for accurate irradiation damage simulations. *Appl. Phys. Lett.*, 114(24):244101, 2019. doi:10.1063/1.5098061.
- [LJC+22] Denghui Lu, Wanrun Jiang, Yixiao Chen, Linfeng Zhang, Weile Jia, Han Wang, and Mohan Chen. DP Compress: A Model Compression Scheme for Generating Efficient Deep Potential Models. *J. Chem. Theory Comput.*, 18:5555–5567, 2022. doi:10.1021/acs.jctc.2c00102.
- [ZLW+19] Linfeng Zhang, De-Ye Lin, Han Wang, Roberto Car, and Weinan E. Active learning of uniformly accurate interatomic potentials for materials simulation. *Phys. Rev. Mater.*, 3:23804, 2019. doi:10.1103/PhysRevMaterials.3.023804.
- [ZZWZ21] Jinzhe Zeng, Linfeng Zhang, Han Wang, and Tong Zhu. Exploring the Chemical Space of Linear Alkane Pyrolysis via Deep Potential GENerator. *Energy & Fuels*, 35(1):762–769, 2021. doi:10.1021/acs.energyfuels.0c03211.
- [MLZ+22] Pinghui Mo, Chang Li, Dan Zhao, Yujia Zhang, Mengchao Shi, Junhua Li, and Jie Liu. Accurate and efficient molecular dynamics based on machine learning and non von Neumann architecture. *npj Comput. Mater.*, 8:107, 2022. doi:10.1038/s41524-022-00773-z.



## PYTHON MODULE INDEX

### b

backend, 217  
backend.dynamic\_metadata, 217  
backend.find\_tensorflow, 217  
backend.read\_env, 218

### d

deepmd, 219  
deepmd.calculator, 552  
deepmd.cluster, 225  
deepmd.cluster.local, 225  
deepmd.cluster.slurm, 226  
deepmd.common, 554  
deepmd.descriptor, 226  
deepmd.descriptor.descriptor, 278  
deepmd.descriptor.hybrid, 285  
deepmd.descriptor.loc\_frame, 291  
deepmd.descriptor.se, 295  
deepmd.descriptor.se\_a, 297  
deepmd.descriptor.se\_a\_ebd, 303  
deepmd.descriptor.se\_a\_ebd\_v2, 305  
deepmd.descriptor.se\_a\_ef, 306  
deepmd.descriptor.se\_a\_mask, 312  
deepmd.descriptor.se\_atten, 316  
deepmd.descriptor.se\_atten\_v2, 321  
deepmd.descriptor.se\_r, 323  
deepmd.descriptor.se\_t, 328  
deepmd.entrypoints, 332  
deepmd.entrypoints.compress, 337  
deepmd.entrypoints.convert, 338  
deepmd.entrypoints.doc, 338  
deepmd.entrypoints.freeze, 338  
deepmd.entrypoints.gui, 338  
deepmd.entrypoints.ipi, 339  
deepmd.entrypoints.main, 339  
deepmd.entrypoints.neighbor\_stat, 340  
deepmd.entrypoints.test, 340  
deepmd.entrypoints.train, 341  
deepmd.entrypoints.transfer, 342  
deepmd.env, 559  
deepmd.env.op\_grads\_module, 659  
deepmd.env.op\_module, 607

deepmd.fit, 343  
deepmd.fit.dipole, 357  
deepmd.fit.dos, 360  
deepmd.fit.ener, 363  
deepmd.fit.fitting, 366  
deepmd.fit.polar, 368  
deepmd.infer, 372  
deepmd.infer.data\_modifier, 392  
deepmd.infer.deep\_dipole, 394  
deepmd.infer.deep\_dos, 395  
deepmd.infer.deep\_eval, 398  
deepmd.infer.deep\_polar, 402  
deepmd.infer.deep\_pot, 405  
deepmd.infer.deep\_tensor, 409  
deepmd.infer.deep\_wfc, 412  
deepmd.infer.ewald\_recip, 413  
deepmd.infer.model\_devi, 414  
deepmd.lmp, 559  
deepmd.loggers, 418  
deepmd.loggers.loggers, 419  
deepmd.loss, 420  
deepmd.loss.dos, 426  
deepmd.loss.ener, 427  
deepmd.loss.loss, 432  
deepmd.loss.tensor, 433  
deepmd.model, 434  
deepmd.model.dos, 446  
deepmd.model.ener, 449  
deepmd.model.frozen, 452  
deepmd.model.linear, 455  
deepmd.model.model, 458  
deepmd.model.model\_stat, 465  
deepmd.model.multi, 466  
deepmd.model.pairtab, 469  
deepmd.model.pairwise\_dprc, 472  
deepmd.model.tensor, 475  
deepmd.nvnmd, 480  
deepmd.nvnmd.data, 480  
deepmd.nvnmd.data.data, 481  
deepmd.nvnmd.descriptor, 481  
deepmd.nvnmd.descriptor.se\_a, 482  
deepmd.nvnmd.descriptor.se\_atten, 482

- deepmd.nvnmd.entrypoints, 483
- deepmd.nvnmd.entrypoints.freeze, 487
- deepmd.nvnmd.entrypoints.mapt, 488
- deepmd.nvnmd.entrypoints.train, 490
- deepmd.nvnmd.entrypoints.wrap, 491
- deepmd.nvnmd.fit, 492
- deepmd.nvnmd.fit.ener, 492
- deepmd.nvnmd.utils, 493
- deepmd.nvnmd.utils.argcheck, 497
- deepmd.nvnmd.utils.config, 497
- deepmd.nvnmd.utils.encode, 500
- deepmd.nvnmd.utils.fio, 502
- deepmd.nvnmd.utils.network, 504
- deepmd.nvnmd.utils.op, 505
- deepmd.nvnmd.utils.weight, 505
- deepmd.op, 507
- deepmd.train, 507
- deepmd.train.run\_options, 507
- deepmd.train.trainer, 508
- deepmd.utils, 510
- deepmd.utils.argcheck, 519
- deepmd.utils.batch\_size, 519
- deepmd.utils.compat, 520
- deepmd.utils.compress, 521
- deepmd.utils.convert, 521
- deepmd.utils.data, 523
- deepmd.utils.data\_system, 526
- deepmd.utils.errors, 529
- deepmd.utils.finetune, 530
- deepmd.utils.graph, 530
- deepmd.utils.learning\_rate, 535
- deepmd.utils.multi\_init, 536
- deepmd.utils.neighbor\_stat, 537
- deepmd.utils.network, 538
- deepmd.utils.pair\_tab, 539
- deepmd.utils.parallel\_op, 540
- deepmd.utils.path, 541
- deepmd.utils.plugin, 545
- deepmd.utils.random, 546
- deepmd.utils.sess, 547
- deepmd.utils.spin, 547
- deepmd.utils.tabulate, 548
- deepmd.utils.type\_embed, 550
- deepmd.utils.weight\_avg, 552
- deepmd\_utils, 560
- deepmd\_utils.common, 601
- deepmd\_utils.entrypoints, 560
- deepmd\_utils.entrypoints.doc, 560
- deepmd\_utils.entrypoints.gui, 560
- deepmd\_utils.env, 603
- deepmd\_utils.loggers, 561
- deepmd\_utils.loggers.loggers, 562
- deepmd\_utils.main, 604
- deepmd\_utils.model\_format, 563
- deepmd\_utils.model\_format.common, 571
- deepmd\_utils.model\_format.env\_mat, 571
- deepmd\_utils.model\_format.network, 572
- deepmd\_utils.model\_format.output\_def, 576
- deepmd\_utils.model\_format.se\_e2\_a, 578
- deepmd\_utils.utils, 580
- deepmd\_utils.utils.argcheck, 580
- deepmd\_utils.utils.argcheck\_nvnmd, 583
- deepmd\_utils.utils.batch\_size, 584
- deepmd\_utils.utils.compat, 586
- deepmd\_utils.utils.data, 587
- deepmd\_utils.utils.data\_system, 590
- deepmd\_utils.utils.errors, 593
- deepmd\_utils.utils.model\_stat, 594
- deepmd\_utils.utils.pair\_tab, 594
- deepmd\_utils.utils.path, 595
- deepmd\_utils.utils.plugin, 598
- deepmd\_utils.utils.random, 600
- deepmd\_utils.utils.weight\_avg, 601

## Symbols

\_DP\_DeepPotCompute (C++ function), 739  
 \_DP\_DeepPotCompute<double> (C++ function), 739  
 \_DP\_DeepPotCompute<float> (C++ function), 739  
 \_DP\_DeepPotComputeMixedType (C++ function), 740  
 \_DP\_DeepPotComputeMixedType<double> (C++ function), 740  
 \_DP\_DeepPotComputeMixedType<float> (C++ function), 740  
 \_DP\_DeepPotComputeNList (C++ function), 741  
 \_DP\_DeepPotComputeNList<double> (C++ function), 741  
 \_DP\_DeepPotComputeNList<float> (C++ function), 741  
 \_DP\_DeepPotModelDeviComputeNList (C++ function), 742  
 \_DP\_DeepPotModelDeviComputeNList<double> (C++ function), 742  
 \_DP\_DeepPotModelDeviComputeNList<float> (C++ function), 742  
 \_DP\_DeepTensorCompute (C++ function), 743  
 \_DP\_DeepTensorCompute<double> (C++ function), 743  
 \_DP\_DeepTensorCompute<float> (C++ function), 743  
 \_DP\_DeepTensorComputeNList (C++ function), 744  
 \_DP\_DeepTensorComputeNList<double> (C++ function), 744  
 \_DP\_DeepTensorComputeNList<float> (C++ function), 744  
 \_DP\_DeepTensorComputeTensor (C++ function), 745  
 \_DP\_DeepTensorComputeTensor<double> (C++ function), 745  
 \_DP\_DeepTensorComputeTensor<float> (C++ function), 745  
 \_DP\_DeepTensorComputeTensorNList (C++ function), 745  
 \_DP\_DeepTensorComputeTensorNList<double> (C++ function), 746  
 \_DP\_DeepTensorComputeTensorNList<float>

(C++ function), 746

\_DP\_DipoleChargeModifierComputeNList (C++ function), 746  
 \_DP\_DipoleChargeModifierComputeNList<double> (C++ function), 747  
 \_DP\_DipoleChargeModifierComputeNList<float> (C++ function), 747  
 \_DP\_Get\_Energy\_Pointer (C++ function), 747, 748

## A

activation\_function:

model/type\_embedding/activation\_function (Argument), 90  
 model[standard]/descriptor[se\_a\_ebd\_v2]/activation\_function (Argument), 108  
 model[standard]/descriptor[se\_a\_mask]/activation\_function (Argument), 110  
 model[standard]/descriptor[se\_a\_tpe]/activation\_function (Argument), 98  
 model[standard]/descriptor[se\_atten\_v2]/activation\_function (Argument), 105  
 model[standard]/descriptor[se\_atten]/activation\_function (Argument), 103  
 model[standard]/descriptor[se\_e2\_a]/activation\_function (Argument), 95  
 model[standard]/descriptor[se\_e2\_r]/activation\_function (Argument), 100  
 model[standard]/descriptor[se\_e3]/activation\_function (Argument), 97  
 model[standard]/fitting\_net[dipole]/activation\_function (Argument), 116  
 model[standard]/fitting\_net[dos]/activation\_function (Argument), 113  
 model[standard]/fitting\_net[ener]/activation\_function (Argument), 112  
 model[standard]/fitting\_net[polar]/activation\_function (Argument), 115  
 add() (deepmd.utils.data.DeepmdData method), 524  
 add() (deepmd.utils.data\_system.DeepmdDataSystem method), 527  
 add() (deepmd.utils.DeepmdData method), 511  
 add() (deepmd.utils.DeepmdDataSystem method),

514  
 add() (deepmd\_utils.utils.data.DeepmdData method), 588  
 add() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 591  
 add\_data\_requirement() (in module deepmd.common), 554  
 add\_data\_requirement() (in module deepmd\_utils.common), 601  
 add\_dict() (deepmd\_utils.data\_system.DeepmdDataSystem method), 528  
 add\_dict() (deepmd\_utils.DeepmdDataSystem method), 514  
 add\_dict() (deepmd\_utils.data\_system.DeepmdDataSystem method), 592  
 add\_flt\_nvnmd(C++ function), 805  
 add\_flt\_nvnmd() (in module deepmd.env.op\_module), 633  
 AddFltNvnmd() (in module deepmd.env.op\_module), 607  
 Annotated (class in deepmd.env.op\_grads\_module), 659  
 Annotated (class in deepmd.env.op\_module), 607  
 ArgsPlugin (class in deepmd\_utils.utils.argcheck), 580  
 atom\_ener:  
   model[standard]/fitting\_net[ener]/atom\_ener (Argument), 113  
 attn:  
   model[standard]/descriptor[se\_atten\_v2]/attn (Argument), 106  
   model[standard]/descriptor[se\_atten]/attn (Argument), 103  
 attn\_dotr:  
   model[standard]/descriptor[se\_atten\_v2]/attn\_dotr (Argument), 107  
   model[standard]/descriptor[se\_atten]/attn\_dotr (Argument), 104  
 attn\_layer:  
   model[standard]/descriptor[se\_atten\_v2]/attn\_layer (Argument), 106  
   model[standard]/descriptor[se\_atten]/attn\_layer (Argument), 104  
 attn\_mask:  
   model[standard]/descriptor[se\_atten\_v2]/attn\_mask (Argument), 107  
   model[standard]/descriptor[se\_atten]/attn\_mask (Argument), 104  
 auto\_prob:  
   training/training\_data/auto\_prob (Argument), 126  
   training/validation\_data/auto\_prob (Argument), 127  
 AutoBatchSize (class in deepmd\_utils.batch\_size), 519  
 AutoBatchSize (class in deepmd\_utils.batch\_size), 584  
 avg() (deepmd\_utils.data.DeepmdData method), 525  
 avg() (deepmd\_utils.DeepmdData method), 512  
 avg() (deepmd\_utils.data.DeepmdData method), 589  
 axis\_neuron:  
   model[standard]/descriptor[se\_a\_ebd\_v2]/axis\_neuron (Argument), 108  
   model[standard]/descriptor[se\_a\_mask]/axis\_neuron (Argument), 110  
   model[standard]/descriptor[se\_a\_tpe]/axis\_neuron (Argument), 98  
   model[standard]/descriptor[se\_atten\_v2]/axis\_neuron (Argument), 105  
   model[standard]/descriptor[se\_atten]/axis\_neuron (Argument), 102  
   model[standard]/descriptor[se\_e2\_a]/axis\_neuron (Argument), 95  
 axis\_rule:  
   model[standard]/descriptor[loc\_frame]/axis\_rule (Argument), 93  
**B**  
 backend  
   module, 217  
   backend.dynamic\_metadata  
     module, 217  
   backend.find\_tensorflow  
     module, 217  
   backend.read\_env  
     module, 218  
   batch\_size:  
     training/training\_data/batch\_size (Argument), 125  
     training/validation\_data/batch\_size (Argument), 127  
   bin2hex() (deepmd.nvnmd\_utils.Encode method), 493  
   bin2hex() (deepmd.nvnmd\_utils.encode.Encode method), 500  
   bin2hex\_str() (deepmd.nvnmd\_utils.Encode method), 493  
   bin2hex\_str() (deepmd.nvnmd\_utils.encode.Encode method), 500  
   build() (deepmd.descriptor.Descriptor method), 227  
   build() (deepmd.descriptor.descriptor.Descriptor method), 279  
   build() (deepmd.descriptor.DescriptHybrid method), 234  
   build() (deepmd.descriptor.DescriptLocFrame method), 240

<code>build()</code> (deepmd.descriptor.DescriptSeA method), 245	<code>build()</code> (deepmd.loss.ener.EnerSpinLoss method), 428
<code>build()</code> (deepmd.descriptor.DescriptSeAEbd method), 250	<code>build()</code> (deepmd.loss.ener.EnerStdLoss method), 431
<code>build()</code> (deepmd.descriptor.DescriptSeAEf method), 254	<code>build()</code> (deepmd.loss.EnerDipoleLoss method), 421
<code>build()</code> (deepmd.descriptor.DescriptSeAEfLower method), 257	<code>build()</code> (deepmd.loss.EnerSpinLoss method), 422
<code>build()</code> (deepmd.descriptor.DescriptSeAMask method), 260	<code>build()</code> (deepmd.loss.EnerStdLoss method), 424
<code>build()</code> (deepmd.descriptor.DescriptSeAtten method), 264	<code>build()</code> (deepmd.loss.loss.Loss method), 432
<code>build()</code> (deepmd.descriptor.DescriptSeR method), 271	<code>build()</code> (deepmd.loss.tensor.TensorLoss method), 433
<code>build()</code> (deepmd.descriptor.DescriptSeT method), 275	<code>build()</code> (deepmd.loss.TensorLoss method), 425
<code>build()</code> (deepmd.descriptor.hybrid.DescriptHybrid method), 286	<code>build()</code> (deepmd.model.dos.DOSModel method), 447
<code>build()</code> (deepmd.descriptor.loc_frame.DescriptLocFrame method), 292	<code>build()</code> (deepmd.model.DOSModel method), 435
<code>build()</code> (deepmd.descriptor.se_a.DescriptSeA method), 299	<code>build()</code> (deepmd.model.ener.EnerModel method), 450
<code>build()</code> (deepmd.descriptor.se_a_ebd.DescriptSeAEbd method), 304	<code>build()</code> (deepmd.model.EnerModel method), 439
<code>build()</code> (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 308	<code>build()</code> (deepmd.model.frozen.FrozenModel method), 453
<code>build()</code> (deepmd.descriptor.se_a_ef.DescriptSeAEfLower method), 311	<code>build()</code> (deepmd.model.linear.LinearEnergyModel method), 455
<code>build()</code> (deepmd.descriptor.se_a_mask.DescriptSeAMask method), 314	<code>build()</code> (deepmd.model.model.Model method), 459
<code>build()</code> (deepmd.descriptor.se_atten.DescriptSeAtten method), 318	<code>build()</code> (deepmd.model.multi.MultiModel method), 467
<code>build()</code> (deepmd.descriptor.se_r.DescriptSeR method), 325	<code>build()</code> (deepmd.model.MultiModel method), 442
<code>build()</code> (deepmd.descriptor.se_t.DescriptSeT method), 329	<code>build()</code> (deepmd.model.pairtab.PairTabModel method), 470
<code>build()</code> (deepmd.fit.dipole.DipoleFittingSeA method), 358	<code>build()</code> (deepmd.model.pairwise_dprc.PairwiseDPRC method), 472
<code>build()</code> (deepmd.fit.DipoleFittingSeA method), 346	<code>build()</code> (deepmd.model.tensor.TensorModel method), 478
<code>build()</code> (deepmd.fit.dos.DOSFitting method), 361	<code>build()</code> (deepmd.train.trainer.DatasetLoader method), 509
<code>build()</code> (deepmd.fit.DOSFitting method), 344	<code>build()</code> (deepmd.train.trainer.DPTrainer method), 508
<code>build()</code> (deepmd.fit.ener.EnerFitting method), 364	<code>build()</code> (deepmd.utils.learning_rate.LearningRateExp method), 536
<code>build()</code> (deepmd.fit.EnerFitting method), 349	<code>build()</code> (deepmd.utils.LearningRateExp method), 516
<code>build()</code> (deepmd.fit.GlobalPolarFittingSeA method), 354	<code>build()</code> (deepmd.utils.spin.Spin method), 548
<code>build()</code> (deepmd.fit.polar.GlobalPolarFittingSeA method), 369	<code>build()</code> (deepmd.utils.tabulate.DPTabulate method), 549
<code>build()</code> (deepmd.fit.polar.PolarFittingSeA method), 371	<code>build()</code> (deepmd.utils.type_embed.TypeEmbedNet method), 551
<code>build()</code> (deepmd.fit.PolarFittingSeA method), 356	<code>build_davg_dstd()</code> (deepmd.nvnmd.entrypoints.maptable.MapTable method), 489
<code>build()</code> (deepmd.loss.dos.DOSLoss method), 426	<code>build_davg_dstd()</code> (deepmd.nvnmd.entrypoints.MapTable method), 484
<code>build()</code> (deepmd.loss.DOSLoss method), 420	<code>build_davg_dstd()</code> (in module deepmd.nvnmd.descriptor.se_a), 482
<code>build()</code> (deepmd.loss.ener.EnerDipoleLoss method), 427	<code>build_davg_dstd()</code> (in module deepmd.nvnmd.descriptor.se_atten), 482
	<code>build_descrpt()</code> (deepmd.model.model.Model method), 460
	<code>build_embedding_net()</code>



(deepmd.nvnmd.entrypoints.maptable.MapTable method), 489	(deepmd.descriptor.descriptor.Descriptor method), 280
build_embedding_net() (deepmd.nvnmd.entrypoints.MapTable method), 484	build_type_exclude_mask() (deepmd.descriptor.DescriptSeAtten method), 265
build_fv_graph() (deepmd.DipoleChargeModifier method), 224	build_type_exclude_mask() (deepmd.descriptor.se_atten.DescriptSeAtten method), 319
build_fv_graph() (deepmd.infer.data_modifier.DipoleChargeModifier method), 393	build_u2s() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 490
build_fv_graph() (deepmd.infer.DipoleChargeModifier method), 389	build_u2s() (deepmd.nvnmd.entrypoints.MapTable method), 485
build_grad() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 489	build_u2s_grad() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 490
build_grad() (deepmd.nvnmd.entrypoints.MapTable method), 484	build_u2s_grad() (deepmd.nvnmd.entrypoints.MapTable method), 485
build_map() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 489	byte2hex() (deepmd.nvnmd.utils.Encode method), 493
build_map() (deepmd.nvnmd.entrypoints.MapTable method), 484	byte2hex() (deepmd.nvnmd.utils.encode.Encode method), 500
build_map_coef() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 489	
build_map_coef() (deepmd.nvnmd.entrypoints.MapTable method), 484	
build_neighbor_list() (deepmd.DeepEval method), 219	cal_coef4() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 490
build_neighbor_list() (deepmd.infer.deep_eval.DeepEval method), 399	cal_coef4() (deepmd.nvnmd.entrypoints.MapTable method), 485
build_neighbor_list() (deepmd.infer.DeepEval method), 377	cal_g() (deepmd_utils.model_format.DescriptSeA method), 564
build_nlist (C++ function), 805, 806	cal_g() (deepmd_utils.model_format.se_e2_a.DescriptSeA method), 580
build_op_descriptor() (in module deepmd.nvnmd.descriptor.se_a), 482	calc_model_devi() (in module deepmd.infer), 391
build_op_descriptor() (in module deepmd.nvnmd.descriptor.se_atten), 482	calc_model_devi() (in module deepmd.infer.model_devi), 414
build_s2g() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 489	calc_model_devi_e() (in module deepmd.infer.model_devi), 415
build_s2g() (deepmd.nvnmd.entrypoints.MapTable method), 484	calc_model_devi_f() (in module deepmd.infer.model_devi), 416
build_s2g_grad() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 489	calc_model_devi_v() (in module deepmd.infer.model_devi), 416
build_s2g_grad() (deepmd.nvnmd.entrypoints.MapTable method), 484	calculate() (deepmd.calculator.DP method), 554
build_t2g() (deepmd.nvnmd.entrypoints.maptable.MapTable method), 489	call() (deepmd_utils.model_format.common.NativeOP method), 571
build_t2g() (deepmd.nvnmd.entrypoints.MapTable method), 484	call() (deepmd_utils.model_format.DescriptSeA method), 564
build_type_embedding() (deepmd.model.model.Model method), 461	call() (deepmd_utils.model_format.env_mat.EnvMat method), 571
build_type_exclude_mask() (deepmd.descriptor.Descriptor method), 228	call() (deepmd_utils.model_format.EnvMat method), 565
build_type_exclude_mask()	call() (deepmd_utils.model_format.NativeLayer method), 567
	call() (deepmd_utils.model_format.NativeOP method), 568
	call() (deepmd_utils.model_format.network.NativeLayer method), 573



`call()` (deepmd\_utils.model\_format.se\_e2\_a.DescriptSeA method), 568  
`call()` (deepmd\_utils.model\_format.se\_e2\_a.DescriptSeA method), 580  
`cast_precision()` (in module deepmd.common), 555  
`change_energy_bias()` (deepmd.fit.ener.EnerFitting method), 365  
`change_energy_bias()` (deepmd.fit.EnerFitting method), 350  
`change_energy_bias()` (deepmd.model.ener.EnerModel method), 451  
`change_energy_bias()` (deepmd.model.EnerModel method), 439  
`change_energy_bias()` (deepmd.model.model.Model method), 461  
`check_batch_size()` (deepmd\_utils.data.DeepmdData method), 525  
`check_batch_size()` (deepmd\_utils.DeepmdData method), 512  
`check_batch_size()` (deepmd\_utils.data.DeepmdData method), 589  
`check_completeness()` (deepmd\_utils.model\_format.network.NetworkCollection method), 575  
`check_completeness()` (deepmd\_utils.model\_format.NetworkCollection method), 569  
`check_dec()` (deepmd.nvnmd\_utils.Encode method), 494  
`check_dec()` (deepmd.nvnmd\_utils.encode.Encode method), 500  
`check_shape()` (in module deepmd\_utils.model\_format.output\_def), 578  
`check_shape_consistency()` (deepmd\_utils.model\_format.NativeLayer method), 567  
`check_shape_consistency()` (deepmd\_utils.model\_format.network.NativeLayer method), 574  
`check_switch_range()` (in module deepmd.nvnmd.descriptor.se\_a), 482  
`check_switch_range()` (in module deepmd.nvnmd.descriptor.se\_atten), 482  
`check_test_size()` (deepmd\_utils.data.DeepmdData method), 525  
`check_test_size()` (deepmd\_utils.DeepmdData method), 512  
`check_test_size()` (deepmd\_utils.data.DeepmdData method), 589  
`check_type_consistency()` (deepmd\_utils.model\_format.NativeLayer method), 574  
`check_type_consistency()` (deepmd\_utils.model\_format.network.NativeLayer method), 574  
`check_var()` (in module deepmd\_utils.model\_format.output\_def), 578  
`choice()` (in module deepmd\_utils.random), 546  
`choice()` (in module deepmd\_utils.random), 600  
`clear_session()` (in module deepmd.common), 555  
`compress()` (in module deepmd.entrypoints), 332  
`compress()` (in module deepmd.entrypoints.compress), 337  
`compress:`  
`model/compress` (Argument), 92  
`compute_descriptor` (C++ function), 807  
`compute_descriptor_se_a_ef_para` (C++ function), 807  
`compute_descriptor_se_a_ef_vert` (C++ function), 808  
`compute_descriptor_se_a_extf` (C++ function), 808  
`compute_energy_shift()` (deepmd\_utils.data\_system.DeepmdDataSystem method), 528  
`compute_energy_shift()` (deepmd\_utils.DeepmdDataSystem method), 514  
`compute_energy_shift()` (deepmd\_utils.data\_system.DeepmdDataSystem method), 592  
`compute_input_stats()` (deepmd.descriptor.Descriptor method), 229  
`compute_input_stats()` (deepmd.descriptor.descriptor.Descriptor method), 281  
`compute_input_stats()` (deepmd.descriptor.DescriptHybrid method), 235  
`compute_input_stats()` (deepmd.descriptor.DescriptLocFrame method), 241  
`compute_input_stats()` (deepmd.descriptor.DescriptSeA method), 246  
`compute_input_stats()` (deepmd.descriptor.DescriptSeAEf method), 255  
`compute_input_stats()` (deepmd.descriptor.DescriptSeAEfLower method), 258  
`compute_input_stats()`

(deepmd.descriptor.DescriptSeAMask method), 261	compute_output_stats()	(deepmd.fit.dos.DOSFitting method), 361
compute_input_stats()	(deepmd.descriptor.DescriptSeAtten method), 266	compute_output_stats() (deepmd.fit.DOSFitting method), 344
compute_input_stats()	(deepmd.descriptor.DescriptSeR method), 272	compute_output_stats()
compute_input_stats()	(deepmd.descriptor.DescriptSeT method), 276	(deepmd.fit.ener.EnerFitting method), 365
compute_input_stats()	(deepmd.descriptor.hybrid.DescriptHybrid method), 287	compute_output_stats() (deepmd.fit.EnerFitting method), 350
compute_input_stats()	(deepmd.descriptor.loc_frame.DescriptLocFrame method), 293	compute_prec:
compute_input_stats()	(deepmd.descriptor.se_a.DescriptSeA method), 300	training/mixed_precision/compute_prec (Argument), 128
compute_input_stats()	(deepmd.descriptor.se_a_ef.DescriptSeAEf method), 309	compute_smooth_weight() (in module deepmd_utils.model_format.env_mat), 572
compute_input_stats()	(deepmd.descriptor.se_a_ef.DescriptSeAEfLower method), 312	config_file:
compute_input_stats()	(deepmd.descriptor.se_a_mask.DescriptSeAMask method), 315	nvnmdd/config_file (Argument), 130
compute_input_stats()	(deepmd.descriptor.se_atten.DescriptSeAtten method), 320	convert() (in module deepmd.entrypoints), 333
compute_input_stats()	(deepmd.descriptor.se_r.DescriptSeR method), 326	convert() (in module deepmd.entrypoints.convert), 338
compute_input_stats()	(deepmd.descriptor.se_t.DescriptSeT method), 330	convert_012_to_21() (in module deepmd.utils.convert), 521
compute_input_stats()	(deepmd.fit.dos.DOSFitting method), 361	convert_10_to_21() (in module deepmd.utils.convert), 521
compute_input_stats() (deepmd.fit.DOSFitting method), 344		convert_12_to_21() (in module deepmd.utils.convert), 521
compute_input_stats()	(deepmd.fit.ener.EnerFitting method), 365	convert_13_to_21() (in module deepmd.utils.convert), 521
compute_input_stats() (deepmd.fit.EnerFitting method), 350		convert_20_to_21() (in module deepmd.utils.convert), 521
compute_input_stats()	(deepmd.fit.polar.PolarFittingSeA method), 371	convert_dp012_to_dp10() (in module deepmd.utils.convert), 522
compute_input_stats()	(deepmd.fit.PolarFittingSeA method), 356	convert_dp10_to_dp11() (in module deepmd.utils.convert), 522
		convert_dp12_to_dp13() (in module deepmd.utils.convert), 522
		convert_dp13_to_dp20() (in module deepmd.utils.convert), 522
		convert_dp20_to_dp21() (in module deepmd.utils.convert), 522
		convert_forward_map() (in module deepmd.env.op_module), 633
		convert_input_v0_v1() (in module deepmd.utils.compat), 520
		convert_input_v0_v1() (in module deepmd_utils.utils.compat), 586
		convert_input_v1_v2() (in module deepmd.utils.compat), 520
		convert_input_v1_v2() (in module deepmd_utils.utils.compat), 586
		convert_pb_to_pbtxt() (in module deepmd.utils.convert), 522
		convert_pbtxt_to_pb() (in module)

deepmd.utils.convert), 522  
 convert\_to\_21() (in module deepmd.utils.convert), 523  
 ConvertForwardMap() (in module deepmd.env.op\_module), 608  
 copy\_coord (C++ function), 809  
 copy\_flt\_nvnmnd() (in module deepmd.env.op\_module), 633  
 CopyFltNvnmd() (in module deepmd.env.op\_module), 608  
 Counter (class in deepmd\_utils.model\_format.network), 572  
 create\_file\_path() (deepmd.nvnmd.utils.fio.Fio method), 502  
**D**  
 data\_bias\_nsample:  
   model/data\_bias\_nsample (Argument), 89  
 data\_dict:  
   training/data\_dict (Argument), 129  
 data\_stat() (deepmd.model.dos.DOSModel method), 448  
 data\_stat() (deepmd.model.DOSModel method), 436  
 data\_stat() (deepmd.model.ener.EnerModel method), 451  
 data\_stat() (deepmd.model.EnerModel method), 440  
 data\_stat() (deepmd.model.frozen.FrozenModel method), 454  
 data\_stat() (deepmd.model.linear.LinearModel method), 457  
 data\_stat() (deepmd.model.model.Model method), 462  
 data\_stat() (deepmd.model.multi.MultiModel method), 468  
 data\_stat() (deepmd.model.MultiModel method), 443  
 data\_stat() (deepmd.model.pairtab.PairTabModel method), 471  
 data\_stat() (deepmd.model.pairwise\_dprc.PairwiseDprcModel method), 473  
 data\_stat() (deepmd.model.tensor.TensorModel method), 479  
 data\_stat\_nbatch:  
   model/data\_stat\_nbatch (Argument), 89  
 data\_stat\_protect:  
   model/data\_stat\_protect (Argument), 89  
 DatasetLoader (class in deepmd.train.trainer), 509  
 dec2bin() (deepmd.nvnmd.utils.Encode method), 494  
 dec2bin() (deepmd.nvnmd.utils.encode.Encode method), 500  
 decay\_steps:  
   learning\_rate[exp]/decay\_steps (Argument), 119  
 DeepDipole (class in deepmd.infer), 375  
 DeepDipole (class in deepmd.infer.deep\_dipole), 394  
 DeepDOS (class in deepmd.infer), 372  
 DeepDOS (class in deepmd.infer.deep\_dos), 395  
 DeepEval (class in deepmd), 219  
 DeepEval (class in deepmd.infer), 376  
 DeepEval (class in deepmd.infer.deep\_eval), 398  
 DeepGlobalPolar (class in deepmd.infer), 380  
 DeepGlobalPolar (class in deepmd.infer.deep\_polar), 402  
 deepmd  
   module, 219  
 deepmd.calculator  
   module, 552  
 deepmd.cluster  
   module, 225  
 deepmd.cluster.local  
   module, 225  
 deepmd.cluster.slurm  
   module, 226  
 deepmd.common  
   module, 554  
 deepmd.descriptor  
   module, 226  
 deepmd.descriptor.descriptor  
   module, 278  
 deepmd.descriptor.hybrid  
   module, 285  
 deepmd.descriptor.loc\_frame  
   module, 291  
 deepmd.descriptor.se  
   module, 295  
 deepmd.descriptor.se\_a  
   module, 297  
 deepmd.descriptor.se\_a\_ebd  
   module, 303  
 deepmd.descriptor.se\_a\_ebd\_v2  
   module, 305  
 deepmd.descriptor.se\_a\_ef  
   module, 306  
 deepmd.descriptor.se\_a\_mask  
   module, 312  
 deepmd.descriptor.se\_atten  
   module, 316  
 deepmd.descriptor.se\_atten\_v2  
   module, 321  
 deepmd.descriptor.se\_r  
   module, 323  
 deepmd.descriptor.se\_t  
   module, 328  
 deepmd.entrypoints  
   module, 332

deepmd.entrypoints.compress  
  module, 337

deepmd.entrypoints.convert  
  module, 338

deepmd.entrypoints.doc  
  module, 338

deepmd.entrypoints.freeze  
  module, 338

deepmd.entrypoints.gui  
  module, 338

deepmd.entrypoints.ipi  
  module, 339

deepmd.entrypoints.main  
  module, 339

deepmd.entrypoints.neighbor\_stat  
  module, 340

deepmd.entrypoints.test  
  module, 340

deepmd.entrypoints.train  
  module, 341

deepmd.entrypoints.transfer  
  module, 342

deepmd.env  
  module, 559

deepmd.env.op\_grads\_module  
  module, 659

deepmd.env.op\_module  
  module, 607

deepmd.fit  
  module, 343

deepmd.fit.dipole  
  module, 357

deepmd.fit.dos  
  module, 360

deepmd.fit.ener  
  module, 363

deepmd.fit.fitting  
  module, 366

deepmd.fit.polar  
  module, 368

deepmd.infer  
  module, 372

deepmd.infer.data\_modifier  
  module, 392

deepmd.infer.deep\_dipole  
  module, 394

deepmd.infer.deep\_dos  
  module, 395

deepmd.infer.deep\_eval  
  module, 398

deepmd.infer.deep\_polar  
  module, 402

deepmd.infer.deep\_pot  
  module, 405

deepmd.infer.deep\_tensor  
  module, 409

deepmd.infer.deep\_wfc  
  module, 412

deepmd.infer.ewald\_recip  
  module, 413

deepmd.infer.model\_dev  
  module, 414

deepmd.lmp  
  module, 559

deepmd.loggers  
  module, 418

deepmd.loggers.loggers  
  module, 419

deepmd.loss  
  module, 420

deepmd.loss.dos  
  module, 426

deepmd.loss.ener  
  module, 427

deepmd.loss.loss  
  module, 432

deepmd.loss.tensor  
  module, 433

deepmd.model  
  module, 434

deepmd.model.dos  
  module, 446

deepmd.model.ener  
  module, 449

deepmd.model.frozen  
  module, 452

deepmd.model.linear  
  module, 455

deepmd.model.model  
  module, 458

deepmd.model.model\_stat  
  module, 465

deepmd.model.multi  
  module, 466

deepmd.model.pairtab  
  module, 469

deepmd.model.pairwise\_dprc  
  module, 472

deepmd.model.tensor  
  module, 475

deepmd.nvnmd  
  module, 480

deepmd.nvnmd.data  
  module, 480

deepmd.nvnmd.data.data  
  module, 481

deepmd.nvnmd.descriptor  
  module, 481

---

deepmd.nvnmd.descriptor.se_a	deepmd.utils.data
module, 482	module, 523
deepmd.nvnmd.descriptor.se_atten	deepmd.utils.data_system
module, 482	module, 526
deepmd.nvnmd.entrpoints	deepmd.utils.errors
module, 483	module, 529
deepmd.nvnmd.entrpoints.freeze	deepmd.utils.finetune
module, 487	module, 530
deepmd.nvnmd.entrpoints.mapt	deepmd.utils.graph
module, 488	module, 530
deepmd.nvnmd.entrpoints.train	deepmd.utils.learning_rate
module, 490	module, 535
deepmd.nvnmd.entrpoints.wrap	deepmd.utils.multi_init
module, 491	module, 536
deepmd.nvnmd.fit	deepmd.utils.neighbor_stat
module, 492	module, 537
deepmd.nvnmd.fit.ener	deepmd.utils.network
module, 492	module, 538
deepmd.nvnmd.utils	deepmd.utils.pair_tab
module, 493	module, 539
deepmd.nvnmd.utils.argcheck	deepmd.utils.parallel_op
module, 497	module, 540
deepmd.nvnmd.utils.config	deepmd.utils.path
module, 497	module, 541
deepmd.nvnmd.utils.encode	deepmd.utils.plugin
module, 500	module, 545
deepmd.nvnmd.utils.fio	deepmd.utils.random
module, 502	module, 546
deepmd.nvnmd.utils.network	deepmd.utils.sess
module, 504	module, 547
deepmd.nvnmd.utils.op	deepmd.utils.spin
module, 505	module, 547
deepmd.nvnmd.utils.weight	deepmd.utils.tabulate
module, 505	module, 548
deepmd.op	deepmd.utils.type_embed
module, 507	module, 550
deepmd.train	deepmd.utils.weight_avg
module, 507	module, 552
deepmd.train.run_options	deepmd::AtomMap (C++ class), 674
module, 507	deepmd::AtomMap::AtomMap (C++ function), 674
deepmd.train.trainer	deepmd::AtomMap::backward (C++ function), 674
module, 508	deepmd::AtomMap::forward (C++ function), 674
deepmd.utils	deepmd::AtomMap::get_bkw_map (C++ function), 674
module, 510	
deepmd.utils.argcheck	deepmd::AtomMap::get_fwd_map (C++ function), 674
module, 519	
deepmd.utils.batch_size	deepmd::AtomMap::get_type (C++ function), 674
module, 519	deepmd::build_nlist_cpu (C++ function), 810
deepmd.utils.compat	deepmd::build_nlist_gpu (C++ function), 810
module, 520	deepmd::check_status (C++ function), 708
deepmd.utils.compress	deepmd::compute_cell_info (C++ function), 810
module, 521	deepmd::convert_nlist (C++ function), 810
deepmd.utils.convert	deepmd::convert_nlist_gpu_device (C++ function), 811
module, 521	

---

`deepmd::convert_pbtxt_to_pb` (C++ function), 708  
`deepmd::convert_to_inter_cpu` (C++ function), 811  
`deepmd::convert_to_inter_gpu` (C++ function), 811  
`deepmd::convert_to_phys_cpu` (C++ function), 812  
`deepmd::convert_to_phys_gpu` (C++ function), 812  
`deepmd::copy_coord_cpu` (C++ function), 812  
`deepmd::copy_coord_gpu` (C++ function), 812  
`deepmd::cos_switch` (C++ function), 813  
`deepmd::cprod` (C++ function), 813  
`deepmd::cum_sum` (C++ function), 813  
`deepmd::deepmd_exception` (C++ struct), 672, 795  
`deepmd::deepmd_exception::deepmd_exception` (C++ function), 795  
`deepmd::deepmd_exception_oom` (C++ struct), 796  
`deepmd::deepmd_exception_oom::deepmd_exception_oom` (C++ function), 796  
`deepmd::DeepPot` (C++ class), 674  
`deepmd::DeepPot::~~DeepPot` (C++ function), 679  
`deepmd::DeepPot::compute` (C++ function), 674--677  
`deepmd::DeepPot::compute_mixed_type` (C++ function), 678, 679  
`deepmd::DeepPot::cutoff` (C++ function), 680  
`deepmd::DeepPot::DeepPot` (C++ function), 679  
`deepmd::DeepPot::dim_aparam` (C++ function), 680  
`deepmd::DeepPot::dim_fparam` (C++ function), 680  
`deepmd::DeepPot::get_type_map` (C++ function), 680  
`deepmd::DeepPot::init` (C++ function), 679  
`deepmd::DeepPot::is_aparam_nall` (C++ function), 680  
`deepmd::DeepPot::numb_types` (C++ function), 680  
`deepmd::DeepPot::numb_types_spin` (C++ function), 680  
`deepmd::DeepPot::print_summary` (C++ function), 680  
`deepmd::DeepPotBase` (C++ class), 681  
`deepmd::DeepPotBase::~~DeepPotBase` (C++ function), 684  
`deepmd::DeepPotBase::computew` (C++ function), 681, 682  
`deepmd::DeepPotBase::computew_mixed_type` (C++ function), 683  
`deepmd::DeepPotBase::cutoff` (C++ function), 684  
`deepmd::DeepPotBase::DeepPotBase` (C++ function), 684  
`deepmd::DeepPotBase::dim_aparam` (C++ function), 684  
`deepmd::DeepPotBase::dim_fparam` (C++ function), 684  
`deepmd::DeepPotBase::get_type_map` (C++ function), 684  
`deepmd::DeepPotBase::init` (C++ function), 684  
`deepmd::DeepPotBase::is_aparam_nall` (C++ function), 685  
`deepmd::DeepPotBase::numb_types` (C++ function), 684  
`deepmd::DeepPotBase::numb_types_spin` (C++ function), 684  
`deepmd::DeepPotModelDevi` (C++ class), 685  
`deepmd::DeepPotModelDevi::~~DeepPotModelDevi` (C++ function), 685  
`deepmd::DeepPotModelDevi::compute` (C++ function), 685, 686  
`deepmd::DeepPotModelDevi::compute_avg` (C++ function), 687  
`deepmd::DeepPotModelDevi::compute_relative_std` (C++ function), 688  
`deepmd::DeepPotModelDevi::compute_relative_std_f` (C++ function), 688  
`deepmd::DeepPotModelDevi::compute_std` (C++ function), 688  
`deepmd::DeepPotModelDevi::compute_std_e` (C++ function), 688  
`deepmd::DeepPotModelDevi::compute_std_f` (C++ function), 688  
`deepmd::DeepPotModelDevi::cutoff` (C++ function), 687  
`deepmd::DeepPotModelDevi::DeepPotModelDevi` (C++ function), 685  
`deepmd::DeepPotModelDevi::dim_aparam` (C++ function), 687  
`deepmd::DeepPotModelDevi::dim_fparam` (C++ function), 687  
`deepmd::DeepPotModelDevi::init` (C++ function), 685  
`deepmd::DeepPotModelDevi::is_aparam_nall` (C++ function), 689  
`deepmd::DeepPotModelDevi::numb_types` (C++ function), 687  
`deepmd::DeepPotModelDevi::numb_types_spin` (C++ function), 687  
`deepmd::DeepPotTF` (C++ class), 689  
`deepmd::DeepPotTF::~~DeepPotTF` (C++ function), 689  
`deepmd::DeepPotTF::computew` (C++ function), 690--692  
`deepmd::DeepPotTF::computew_mixed_type` (C++ function), 692, 693



---

`deepmd::DeepPotTF::cutoff` (C++ function), 690  
`deepmd::DeepPotTF::DeepPotTF` (C++ function), 689  
`deepmd::DeepPotTF::dim_aparam` (C++ function), 690  
`deepmd::DeepPotTF::dim_fparam` (C++ function), 690  
`deepmd::DeepPotTF::get_type_map` (C++ function), 690  
`deepmd::DeepPotTF::init` (C++ function), 689  
`deepmd::DeepPotTF::is_aparam_nall` (C++ function), 690  
`deepmd::DeepPotTF::numb_types` (C++ function), 690  
`deepmd::DeepPotTF::numb_types_spin` (C++ function), 690  
`deepmd::DeepTensor` (C++ class), 693  
`deepmd::DeepTensor::~DeepTensor` (C++ function), 693  
`deepmd::DeepTensor::compute` (C++ function), 694, 695  
`deepmd::DeepTensor::cutoff` (C++ function), 696  
`deepmd::DeepTensor::DeepTensor` (C++ function), 693  
`deepmd::DeepTensor::get_type_map` (C++ function), 696  
`deepmd::DeepTensor::init` (C++ function), 693  
`deepmd::DeepTensor::numb_types` (C++ function), 696  
`deepmd::DeepTensor::output_dim` (C++ function), 696  
`deepmd::DeepTensor::print_summary` (C++ function), 693  
`deepmd::DeepTensor::sel_types` (C++ function), 696  
`deepmd::DeepTensorBase` (C++ class), 697  
`deepmd::DeepTensorBase::~DeepTensorBase` (C++ function), 698  
`deepmd::DeepTensorBase::computew` (C++ function), 697, 698  
`deepmd::DeepTensorBase::cutoff` (C++ function), 699  
`deepmd::DeepTensorBase::DeepTensorBase` (C++ function), 698  
`deepmd::DeepTensorBase::get_type_map` (C++ function), 699  
`deepmd::DeepTensorBase::init` (C++ function), 698  
`deepmd::DeepTensorBase::numb_types` (C++ function), 699  
`deepmd::DeepTensorBase::output_dim` (C++ function), 699  
`deepmd::DeepTensorBase::sel_types` (C++ function), 699  
`deepmd::DeepTensorTF` (C++ class), 700  
`deepmd::DeepTensorTF::~DeepTensorTF` (C++ function), 701  
`deepmd::DeepTensorTF::computew` (C++ function), 700, 701  
`deepmd::DeepTensorTF::cutoff` (C++ function), 701  
`deepmd::DeepTensorTF::DeepTensorTF` (C++ function), 701  
`deepmd::DeepTensorTF::get_type_map` (C++ function), 702  
`deepmd::DeepTensorTF::init` (C++ function), 701  
`deepmd::DeepTensorTF::numb_types` (C++ function), 702  
`deepmd::DeepTensorTF::output_dim` (C++ function), 702  
`deepmd::DeepTensorTF::sel_types` (C++ function), 702  
`deepmd::delete_device_memory` (C++ function), 814  
`deepmd::DipoleChargeModifier` (C++ class), 702  
`deepmd::DipoleChargeModifier::~DipoleChargeModifier` (C++ function), 702  
`deepmd::DipoleChargeModifier::compute` (C++ function), 703  
`deepmd::DipoleChargeModifier::cutoff` (C++ function), 703  
`deepmd::DipoleChargeModifier::DipoleChargeModifier` (C++ function), 702  
`deepmd::DipoleChargeModifier::init` (C++ function), 703  
`deepmd::DipoleChargeModifier::numb_types` (C++ function), 703  
`deepmd::DipoleChargeModifier::print_summary` (C++ function), 703  
`deepmd::DipoleChargeModifier::sel_types` (C++ function), 703  
`deepmd::DipoleChargeModifierBase` (C++ class), 704  
`deepmd::DipoleChargeModifierBase::~DipoleChargeModifierBase` (C++ function), 705  
`deepmd::DipoleChargeModifierBase::computew` (C++ function), 704  
`deepmd::DipoleChargeModifierBase::cutoff` (C++ function), 705  
`deepmd::DipoleChargeModifierBase::DipoleChargeModifierBase` (C++ function), 705  
`deepmd::DipoleChargeModifierBase::init` (C++ function), 705  
`deepmd::DipoleChargeModifierBase::numb_types` (C++ function), 705  
`deepmd::DipoleChargeModifierBase::sel_types` (C++ function), 705  
`deepmd::DipoleChargeModifierTF` (C++ class), 706

---

```

deepmd::DipoleChargeModifierTF::~DipoleChargeModifierTF (
    (C++ function), 706
deepmd::DipoleChargeModifierTF::computew
    (C++ function), 707
deepmd::DipoleChargeModifierTF::cutoff (C++
    function), 706
deepmd::DipoleChargeModifierTF::DipoleChargeModifierTF
    (C++ function), 706
deepmd::DipoleChargeModifierTF::init (C++
    function), 706
deepmd::DipoleChargeModifierTF::numb_types
    (C++ function), 706
deepmd::DipoleChargeModifierTF::sel_types
    (C++ function), 707
deepmd::dot1 (C++ function), 814
deepmd::dot2 (C++ function), 814
deepmd::dot3 (C++ function), 814
deepmd::dot4 (C++ function), 814
deepmd::dotmv3 (C++ function), 815
deepmd::DPBackend (C++ enum), 708
deepmd::DPBackend::Paddle (C++ enumerator),
    708
deepmd::DPBackend::PyTorch (C++ enumerator),
    708
deepmd::DPBackend::TensorFlow (C++ enumera-
    tor), 708
deepmd::DPBackend::Unknown (C++ enumerator),
    708
deepmd::DPGetDeviceCount (C++ function), 815
deepmd::dprc_pairwise_map_cpu (C++ function),
    815
deepmd::DPSetDevice (C++ function), 816
deepmd::ElectrostaticConversion (C++ mem-
    ber), 845
deepmd::ENERGYTYPE (C++ type), 717
deepmd::env_mat_a_cpu (C++ function), 816
deepmd::env_mat_a_nvnmnd_quantize_cpu (C++
    function), 816
deepmd::env_mat_nbor_update (C++ function),
    817
deepmd::env_mat_r_cpu (C++ function), 817
deepmd::ewald_recip (C++ function), 817
deepmd::EwaldParameters (C++ struct), 796
deepmd::EwaldParameters::beta (C++ member),
    796
deepmd::EwaldParameters::rcut (C++ member),
    796
deepmd::EwaldParameters::spacing (C++ mem-
    ber), 796
deepmd::filter_ftype_gpu (C++ function), 817
deepmd::format_nbor_list_gpu (C++ function),
    818
deepmd::format_nlist_cpu (C++ function), 818
deepmd::free_nlist_gpu_device (C++ function),
    818
deepmd::gelu_cpu (C++ function), 819
deepmd::gelu_gpu (C++ function), 819
deepmd::gelu_grad_cpu (C++ function), 819
deepmd::gelu_grad_gpu (C++ function), 819
deepmd::gelu_grad_grad_cpu (C++ function), 819
deepmd::gelu_grad_grad_gpu (C++ function), 820
deepmd::get_env_nthreads (C++ function), 709
deepmd::group_atoms_cpu (C++ function), 820
deepmd::hpp::convert_nlist (C++ function), 748
deepmd::hpp::convert_pbtxt_to_pb (C++ func-
    tion), 748
deepmd::hpp::deepmd_exception (C++ struct), 721
deepmd::hpp::deepmd_exception::deepmd_exception
    (C++ function), 721
deepmd::hpp::DeepPot (C++ class), 725
deepmd::hpp::DeepPot::~DeepPot (C++ function),
    725
deepmd::hpp::DeepPot::compute (C++ function),
    725--727
deepmd::hpp::DeepPot::compute_mixed_type
    (C++ function), 728, 729
deepmd::hpp::DeepPot::cutoff (C++ function),
    730
deepmd::hpp::DeepPot::DeepPot (C++ function),
    725
deepmd::hpp::DeepPot::dim_aparam (C++ func-
    tion), 730
deepmd::hpp::DeepPot::dim_fparam (C++ func-
    tion), 730
deepmd::hpp::DeepPot::get_type_map (C++
    function), 730
deepmd::hpp::DeepPot::init (C++ function), 725
deepmd::hpp::DeepPot::numb_types (C++ func-
    tion), 730
deepmd::hpp::DeepPot::numb_types_spin (C++
    function), 730
deepmd::hpp::DeepPot::print_summary (C++
    function), 730
deepmd::hpp::DeepPotModelDevi (C++ class), 731
deepmd::hpp::DeepPotModelDevi::~DeepPotModelDevi
    (C++ function), 731
deepmd::hpp::DeepPotModelDevi::compute (C++
    function), 731
deepmd::hpp::DeepPotModelDevi::compute_avg
    (C++ function), 732
deepmd::hpp::DeepPotModelDevi::compute_relative_std
    (C++ function), 733
deepmd::hpp::DeepPotModelDevi::compute_relative_std_f
    (C++ function), 733
deepmd::hpp::DeepPotModelDevi::compute_std
    (C++ function), 733
deepmd::hpp::DeepPotModelDevi::compute_std_f
    (C++ function), 733

```



---

deepmd::hpp::DeepPotModelDevi::cutoff (C++ function), 732	deepmd::hpp::InputNlist::~InputNlist (C++ function), 721
deepmd::hpp::DeepPotModelDevi::DeepPotModelDevi (C++ function), 731	deepmd::hpp::InputNlist::firstneigh (C++ member), 722
deepmd::hpp::DeepPotModelDevi::dim_aparam (C++ function), 732	deepmd::hpp::InputNlist::ilist (C++ member), 722
deepmd::hpp::DeepPotModelDevi::dim_fparam (C++ function), 732	deepmd::hpp::InputNlist::InputNlist (C++ function), 721
deepmd::hpp::DeepPotModelDevi::init (C++ function), 731	deepmd::hpp::InputNlist::inum (C++ member), 722
deepmd::hpp::DeepPotModelDevi::numb_types (C++ function), 732	deepmd::hpp::InputNlist::nl (C++ member), 722
deepmd::hpp::DeepPotModelDevi::numb_types_spin (C++ function), 732	deepmd::hpp::InputNlist::numneigh (C++ member), 722
deepmd::hpp::DeepTensor (C++ class), 734	deepmd::hpp::read_file_to_string (C++ function), 748
deepmd::hpp::DeepTensor::~DeepTensor (C++ function), 734	deepmd::hpp::select_by_type (C++ function), 749
deepmd::hpp::DeepTensor::compute (C++ function), 734-736	deepmd::hpp::select_map (C++ function), 749
deepmd::hpp::DeepTensor::cutoff (C++ function), 737	deepmd::init_region_cpu (C++ function), 820
deepmd::hpp::DeepTensor::DeepTensor (C++ function), 734	deepmd::InputNlist (C++ struct), 797
deepmd::hpp::DeepTensor::get_type_map (C++ function), 737	deepmd::InputNlist::~InputNlist (C++ function), 797
deepmd::hpp::DeepTensor::init (C++ function), 734	deepmd::InputNlist::firstneigh (C++ member), 797
deepmd::hpp::DeepTensor::numb_types (C++ function), 737	deepmd::InputNlist::ilist (C++ member), 797
deepmd::hpp::DeepTensor::output_dim (C++ function), 737	deepmd::InputNlist::InputNlist (C++ function), 797
deepmd::hpp::DeepTensor::print_summary (C++ function), 737	deepmd::InputNlist::inum (C++ member), 797
deepmd::hpp::DeepTensor::sel_types (C++ function), 737	deepmd::InputNlist::numneigh (C++ member), 797
deepmd::hpp::DipoleChargeModifier (C++ class), 737	deepmd::invsqrt (C++ function), 821
deepmd::hpp::DipoleChargeModifier::~DipoleChargeModifier (C++ function), 737	deepmd::invsqrt<double> (C++ function), 821
deepmd::hpp::DipoleChargeModifier::compute (C++ function), 738	deepmd::invsqrt<float> (C++ function), 821
deepmd::hpp::DipoleChargeModifier::cutoff (C++ function), 738	deepmd::load_op_library (C++ function), 709
deepmd::hpp::DipoleChargeModifier::DipoleChargeModifier (C++ function), 737	deepmd::malloc_device_memory (C++ function), 821, 822
deepmd::hpp::DipoleChargeModifier::init (C++ function), 738	deepmd::malloc_device_memory_sync (C++ function), 822
deepmd::hpp::DipoleChargeModifier::numb_types (C++ function), 738	deepmd::map_aparam_cpu (C++ function), 823
deepmd::hpp::DipoleChargeModifier::print_summary (C++ function), 738	deepmd::max_numneigh (C++ function), 823
deepmd::hpp::DipoleChargeModifier::sel_types (C++ function), 738	deepmd::memcpy_device_to_host (C++ function), 823, 824
deepmd::hpp::InputNlist (C++ struct), 721	deepmd::memcpy_host_to_device (C++ function), 824
	deepmd::memset_device_memory (C++ function), 824
	deepmd::model_compatible (C++ function), 709
	deepmd::name_prefix (C++ function), 710
	deepmd::neighbor_stat_gpu (C++ function), 825
	deepmd::NeighborListData (C++ struct), 672
	deepmd::NeighborListData::copy_from_nlist (C++ function), 672
	deepmd::NeighborListData::firstneigh (C++ member), 673

---

---

```

deepmd::NeighborListData::ilist (C++ member), 673
deepmd::NeighborListData::jlist (C++ member), 673
deepmd::NeighborListData::make_inlist (C++ function), 672
deepmd::NeighborListData::numneigh (C++ member), 673
deepmd::NeighborListData::shuffle (C++ function), 672
deepmd::NeighborListData::shuffle_exclude_empty (C++ function), 672
deepmd::normalize_coord_cpu (C++ function), 825
deepmd::normalize_coord_gpu (C++ function), 825
deepmd::pair_tab_cpu (C++ function), 825
deepmd::print_summary (C++ function), 710
deepmd::prod_env_mat_a_cpu (C++ function), 826
deepmd::prod_env_mat_a_gpu (C++ function), 826
deepmd::prod_env_mat_a_nvmd_quantize_cpu (C++ function), 826
deepmd::prod_env_mat_r_cpu (C++ function), 827
deepmd::prod_env_mat_r_gpu (C++ function), 827
deepmd::prod_force_a_cpu (C++ function), 827, 828
deepmd::prod_force_a_gpu (C++ function), 829
deepmd::prod_force_grad_a_cpu (C++ function), 829
deepmd::prod_force_grad_a_gpu (C++ function), 829
deepmd::prod_force_grad_r_cpu (C++ function), 829
deepmd::prod_force_grad_r_gpu (C++ function), 830
deepmd::prod_force_r_cpu (C++ function), 830
deepmd::prod_force_r_gpu (C++ function), 830
deepmd::prod_virial_a_cpu (C++ function), 830
deepmd::prod_virial_a_gpu (C++ function), 831
deepmd::prod_virial_grad_a_cpu (C++ function), 831
deepmd::prod_virial_grad_a_gpu (C++ function), 831
deepmd::prod_virial_grad_r_cpu (C++ function), 831
deepmd::prod_virial_grad_r_gpu (C++ function), 832
deepmd::prod_virial_r_cpu (C++ function), 832
deepmd::prod_virial_r_gpu (C++ function), 832
deepmd::read_file_to_string (C++ function), 710
deepmd::Region (C++ struct), 797
deepmd::Region::~Region (C++ function), 798
deepmd::Region::boxt (C++ member), 798
deepmd::Region::rec_boxt (C++ member), 798
deepmd::Region::Region (C++ function), 798
deepmd::select_by_type (C++ function), 710
deepmd::select_map (C++ function), 711, 712
deepmd::select_map_inv (C++ function), 712
deepmd::select_real_atoms (C++ function), 713
deepmd::select_real_atoms_coord (C++ function), 713
deepmd::session_get_dtype (C++ function), 713
deepmd::session_get_scalar (C++ function), 714
deepmd::session_get_vector (C++ function), 714
deepmd::session_input_tensors (C++ function), 715
deepmd::session_input_tensors_mixed_type (C++ function), 716
deepmd::soft_min_switch_cpu (C++ function), 832
deepmd::soft_min_switch_force_cpu (C++ function), 833
deepmd::soft_min_switch_force_grad_cpu (C++ function), 833
deepmd::soft_min_switch_virial_cpu (C++ function), 833
deepmd::soft_min_switch_virial_grad_cpu (C++ function), 833
deepmd::spline3_switch (C++ function), 834
deepmd::spline5_switch (C++ function), 834
deepmd::STRINGTYPE (C++ type), 717
deepmd::tabulate_fusion_se_a_cpu (C++ function), 834
deepmd::tabulate_fusion_se_a_gpu (C++ function), 834
deepmd::tabulate_fusion_se_a_grad_cpu (C++ function), 835
deepmd::tabulate_fusion_se_a_grad_gpu (C++ function), 835
deepmd::tabulate_fusion_se_a_grad_grad_cpu (C++ function), 835
deepmd::tabulate_fusion_se_a_grad_grad_gpu (C++ function), 836
deepmd::tabulate_fusion_se_r_cpu (C++ function), 836
deepmd::tabulate_fusion_se_r_gpu (C++ function), 836
deepmd::tabulate_fusion_se_r_grad_cpu (C++ function), 837
deepmd::tabulate_fusion_se_r_grad_gpu (C++ function), 837
deepmd::tabulate_fusion_se_r_grad_grad_cpu (C++ function), 837
deepmd::tabulate_fusion_se_r_grad_grad_gpu (C++ function), 837
deepmd::tabulate_fusion_se_t_cpu (C++ function), 838

```

---

```

deepmd::tabulate_fusion_se_t_gpu (C++ function), 838
deepmd::tabulate_fusion_se_t_grad_cpu (C++ function), 838
deepmd::tabulate_fusion_se_t_grad_gpu (C++ function), 839
deepmd::tabulate_fusion_se_t_grad_grad_cpu (C++ function), 839
deepmd::tabulate_fusion_se_t_grad_grad_gpu (C++ function), 839
deepmd::test_encoding_decoding_nbor_info_gpu (C++ function), 840
deepmd::tf_exception (C++ struct), 673
deepmd::tf_exception::tf_exception (C++ function), 673
deepmd::use_nei_info_cpu (C++ function), 840
deepmd::use_nei_info_gpu (C++ function), 840
deepmd::use_nlist_map (C++ function), 840
deepmd::volume_cpu (C++ function), 841
deepmd::volume_gpu (C++ function), 841
deepmd_utils
  module, 560
deepmd_utils.common
  module, 601
deepmd_utils.entrypoints
  module, 560
deepmd_utils.entrypoints.doc
  module, 560
deepmd_utils.entrypoints.gui
  module, 560
deepmd_utils.env
  module, 603
deepmd_utils.loggers
  module, 561
deepmd_utils.loggers.loggers
  module, 562
deepmd_utils.main
  module, 604
deepmd_utils.model_format
  module, 563
deepmd_utils.model_format.common
  module, 571
deepmd_utils.model_format.env_mat
  module, 571
deepmd_utils.model_format.network
  module, 572
deepmd_utils.model_format.output_def
  module, 576
deepmd_utils.model_format.se_e2_a
  module, 578
deepmd_utils.utils
  module, 580
deepmd_utils.utils.argcheck
  module, 580
deepmd_utils.utils.argcheck_nvnmmd
  module, 583
deepmd_utils.utils.batch_size
  module, 584
deepmd_utils.utils.compat
  module, 586
deepmd_utils.utils.data
  module, 587
deepmd_utils.utils.data_system
  module, 590
deepmd_utils.utils.errors
  module, 593
deepmd_utils.utils.model_stat
  module, 594
deepmd_utils.utils.pair_tab
  module, 594
deepmd_utils.utils.path
  module, 595
deepmd_utils.utils.plugin
  module, 598
deepmd_utils.utils.random
  module, 600
deepmd_utils.utils.weight_avg
  module, 601
DeepmdData (class in deepmd.utils), 510
DeepmdData (class in deepmd.utils.data), 523
DeepmdData (class in deepmd_utils.utils.data), 587
DeepmdDataSystem (class in deepmd.utils), 513
DeepmdDataSystem (class in
  deepmd.utils.data_system), 526
DeepmdDataSystem (class in
  deepmd_utils.utils.data_system), 590
DeepPolar (class in deepmd.infer), 382
DeepPolar (class in deepmd.infer.deep_polar), 404
DeepPot (class in deepmd.infer), 383
DeepPot (class in deepmd.infer.deep_pot), 405
DeepPotential() (in module deepmd), 222
DeepPotential() (in module deepmd.infer), 386
DeepTensor (class in deepmd.infer.deep_tensor), 409
DeepWFC (class in deepmd.infer), 387
DeepWFC (class in deepmd.infer.deep_wfc), 412
default_mesh (deepmd.utils.data_system.DeepmdDataSystem
  property), 528
default_mesh (deepmd.utils.DeepmdDataSystem
  property), 514
default_mesh (deepmd_utils.utils.data_system.DeepmdDataSystem
  property), 592
deprecate_numb_test() (in module
  deepmd.utils.compat), 520
deprecate_numb_test() (in module
  deepmd_utils.utils.compat), 586
Descriptor (class in deepmd.descriptor), 226
Descriptor (class in deepmd.descriptor.descriptor),
  278

```

---

descriptor:  
   model[multi]/descriptor (Argument), 116  
   model[standard]/descriptor (Argument), 92  
 Descript() (in module deepmd.env.op\_module), 608  
 descrpt() (in module deepmd.env.op\_module), 633  
 descrpt2r4() (in module deepmd.nvnmd.descriptor.se\_a), 482  
 descrpt2r4() (in module deepmd.nvnmd.descriptor.se\_atten), 482  
 descrpt\_hybrid\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_local\_frame\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_norot() (in module deepmd.env.op\_module), 634  
 descrpt\_se\_a() (in module deepmd.env.op\_module), 635  
 descrpt\_se\_a\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_a\_ebd\_v2\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_a\_ef() (in module deepmd.env.op\_module), 635  
 descrpt\_se\_a\_ef\_para() (in module deepmd.env.op\_module), 636  
 descrpt\_se\_a\_ef\_vert() (in module deepmd.env.op\_module), 636  
 descrpt\_se\_a\_mask() (in module deepmd.env.op\_module), 637  
 descrpt\_se\_a\_mask\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_a\_tpe\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_atten\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_atten\_common\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_atten\_v2\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_r() (in module deepmd.env.op\_module), 637  
 descrpt\_se\_r\_args() (in module deepmd\_utils.utils.argcheck), 581  
 descrpt\_se\_t\_args() (in module deepmd\_utils.utils.argcheck), 582  
 descrpt\_variant\_type\_args() (in module deepmd\_utils.utils.argcheck), 582  
 DescriptHybrid (class in deepmd.descriptor), 233  
 DescriptHybrid (class in deepmd.descriptor.hybrid), 285  
 DescriptLocFrame (class in deepmd.descriptor), 238  
 DescriptLocFrame (class in deepmd.descriptor.loc\_frame), 291  
 DescriptNorot() (in module deepmd.env.op\_module), 609  
 DescriptSe (class in deepmd.descriptor.se), 295  
 DescriptSeA (class in deepmd.descriptor), 242  
 DescriptSeA (class in deepmd.descriptor.se\_a), 297  
 DescriptSeA (class in deepmd\_utils.model\_format), 563  
 DescriptSeA (class in deepmd\_utils.model\_format.se\_e2\_a), 578  
 DescriptSeA() (in module deepmd.env.op\_module), 609  
 DescriptSeAEbd (class in deepmd.descriptor), 248  
 DescriptSeAEbd (class in deepmd.descriptor.se\_a\_ebd), 303  
 DescriptSeAEbdV2 (class in deepmd.descriptor), 251  
 DescriptSeAEbdV2 (class in deepmd.descriptor.se\_a\_ebd\_v2), 305  
 DescriptSeAEf (class in deepmd.descriptor), 252  
 DescriptSeAEf (class in deepmd.descriptor.se\_a\_ef), 306  
 DescriptSeAEf() (in module deepmd.env.op\_module), 610  
 DescriptSeAEfLower (class in deepmd.descriptor), 256  
 DescriptSeAEfLower (class in deepmd.descriptor.se\_a\_ef), 310  
 DescriptSeAEfPara() (in module deepmd.env.op\_module), 610  
 DescriptSeAEfVert() (in module deepmd.env.op\_module), 611  
 DescriptSeAMask (class in deepmd.descriptor), 258  
 DescriptSeAMask (class in deepmd.descriptor.se\_a\_mask), 312  
 DescriptSeAMask() (in module deepmd.env.op\_module), 612  
 DescriptSeAtten (class in deepmd.descriptor), 262  
 DescriptSeAtten (class in deepmd.descriptor.se\_atten), 316  
 DescriptSeAttenV2 (class in deepmd.descriptor), 267  
 DescriptSeAttenV2 (class in deepmd.descriptor.se\_atten\_v2), 321  
 DescriptSeR (class in deepmd.descriptor), 269  
 DescriptSeR (class in deepmd.descriptor.se\_r), 323  
 DescriptSeR() (in module deepmd.env.op\_module), 612  
 DescriptSeRGPUExecuteFunctor (C++ struct), 798  
 DescriptSeRGPUExecuteFunctor::operator() (C++ function), 798  
 DescriptSeT (class in deepmd.descriptor), 274  
 DescriptSeT (class in deepmd.descriptor.se\_t), 328  
 deserialize() (deepmd\_utils.model\_format.DescriptSeA class method), 565  
 deserialize() (deepmd\_utils.model\_format.env\_mat.EnvMat class method), 572

`deserialize()` (deepmd\_utils.model\_format.EnvMat class method), 565  
`deserialize()` (deepmd\_utils.model\_format.NativeLayer class method), 568  
`deserialize()` (deepmd\_utils.model\_format.network.NativeLayer class method), 574  
`deserialize()` (deepmd\_utils.model\_format.network.NetworkCollection class method), 575  
`deserialize()` (deepmd\_utils.model\_format.NetworkCollection class method), 569  
`deserialize()` (deepmd\_utils.model\_format.se\_e2\_a.DipoleChargeModifier class method), 580  
`detect_model_version()` (in module deepmd\_utils.convert), 523  
`dim_in()` (deepmd\_utils.model\_format.NativeLayer method), 568  
`dim_in()` (deepmd\_utils.model\_format.network.NativeLayer method), 574  
`dim_out()` (deepmd\_utils.model\_format.NativeLayer method), 568  
`dim_out()` (deepmd\_utils.model\_format.network.NativeLayer method), 574  
DipoleChargeModifier (class in deepmd), 223  
DipoleChargeModifier (class in deepmd.infer), 388  
DipoleChargeModifier (class in deepmd.infer.data\_modifier), 392  
DipoleFittingSeA (class in deepmd.fit), 345  
DipoleFittingSeA (class in deepmd.fit.dipole), 357  
DipoleModel (class in deepmd.model), 436  
DipoleModel (class in deepmd.model.tensor), 475  
disp\_file:  
  training/disp\_file (Argument), 128  
disp\_freq:  
  training/disp\_freq (Argument), 128  
disp\_message() (deepmd.nvnmd.utils.config.NvnmdConfig class method), 498  
disp\_training:  
  training/disp\_training (Argument), 128  
display\_if\_exist() (deepmd.loss.loss.Loss static method), 432  
do\_derivative() (in module deepmd\_utils.model\_format.output\_def), 578  
do\_reduce() (in module deepmd\_utils.model\_format.output\_def), 578  
doc\_train\_input() (in module deepmd.entrypoints), 333  
doc\_train\_input() (in module deepmd.entrypoints.doc), 338  
doc\_train\_input() (in module deepmd\_utils.entrypoints.doc), 560  
DOSFitting (class in deepmd.fit), 343  
DOSFitting (class in deepmd.fit.dos), 360  
DOSLoss (class in deepmd.loss), 420  
DOSLoss (class in deepmd.loss.dos), 426  
DOSModel (class in deepmd.model), 434  
DOSModel (class in deepmd.model.dos), 446  
NativeLayer\_nvnmd (C++ function), 841  
dotmul\_fit\_nvnmd() (in module deepmd.env.op\_module), 638  
DotmulFitNvnmd() (in module deepmd.env.op\_module), 612  
DP (class in deepmd.calculator), 552  
DP\_CHECK\_OK (C macro), 786  
DP\_ConvertPbtxtToPb (C++ function), 750  
DP\_DeepPot (C++ struct), 722  
DP\_DeepPot (C++ type), 787  
DP\_DeepPot::aparam\_nall (C++ member), 722  
DP\_DeepPot::daparam (C++ member), 722  
DP\_DeepPot::dfparam (C++ member), 722  
DP\_DeepPot::dp (C++ member), 722  
DP\_DeepPot::DP\_DeepPot (C++ function), 722  
DP\_DeepPot::exception (C++ member), 722  
DP\_DeepPotCheckOK (C++ function), 750  
DP\_DeepPotCompute (C++ function), 750  
DP\_DeepPotCompute2 (C++ function), 751  
DP\_DeepPotComputeef (C++ function), 752  
DP\_DeepPotComputeef2 (C++ function), 753  
DP\_DeepPotComputeMixedType (C++ function), 754  
DP\_DeepPotComputeMixedTypeef (C++ function), 755  
DP\_DeepPotComputeNList (C++ function), 755  
DP\_DeepPotComputeNList2 (C++ function), 756  
DP\_DeepPotComputeNListf (C++ function), 757  
DP\_DeepPotComputeNListf2 (C++ function), 758  
DP\_DeepPotGetCutoff (C++ function), 759  
DP\_DeepPotGetDimAParam (C++ function), 759  
DP\_DeepPotGetDimFParam (C++ function), 760  
DP\_DeepPotGetNumbTypes (C++ function), 760  
DP\_DeepPotGetNumbTypesSpin (C++ function), 760  
DP\_DeepPotGetTypeMap (C++ function), 761  
DP\_DeepPotIsAParamNall (C++ function), 761  
DP\_DeepPotModelDevi (C++ struct), 723  
DP\_DeepPotModelDevi (C++ type), 787  
DP\_DeepPotModelDevi::aparam\_nall (C++ member), 723  
DP\_DeepPotModelDevi::daparam (C++ member), 723  
DP\_DeepPotModelDevi::dfparam (C++ member), 723  
DP\_DeepPotModelDevi::dp (C++ member), 723  
DP\_DeepPotModelDevi::DP\_DeepPotModelDevi (C++ function), 723  
DP\_DeepPotModelDevi::exception (C++ member), 723  
DP\_DeepPotModelDeviCheckOK (C++ function), 761



DP\_DeepPotModelDeviComputeNList (C++ function), 762  
 DP\_DeepPotModelDeviComputeNList2 (C++ function), 763  
 DP\_DeepPotModelDeviComputeNListf (C++ function), 764  
 DP\_DeepPotModelDeviComputeNListf2 (C++ function), 765  
 DP\_DeepPotModelDeviGetCutoff (C++ function), 766  
 DP\_DeepPotModelDeviGetDimAParam (C++ function), 766  
 DP\_DeepPotModelDeviGetDimFParam (C++ function), 766  
 DP\_DeepPotModelDeviGetNumbTypes (C++ function), 767  
 DP\_DeepPotModelDeviGetNumbTypesSpin (C++ function), 767  
 DP\_DeepPotModelDeviIsAParamNAll (C++ function), 767  
 DP\_DeepTensor (C++ struct), 723  
 DP\_DeepTensor (C++ type), 788  
 DP\_DeepTensor::DP\_DeepTensor (C++ function), 723  
 DP\_DeepTensor::dt (C++ member), 724  
 DP\_DeepTensor::exception (C++ member), 724  
 DP\_DeepTensorCheckOK (C++ function), 768  
 DP\_DeepTensorCompute (C++ function), 768  
 DP\_DeepTensorComputef (C++ function), 769  
 DP\_DeepTensorComputeNList (C++ function), 769  
 DP\_DeepTensorComputeNListf (C++ function), 770  
 DP\_DeepTensorComputeTensor (C++ function), 771  
 DP\_DeepTensorComputeTensorf (C++ function), 771  
 DP\_DeepTensorComputeTensorNList (C++ function), 772  
 DP\_DeepTensorComputeTensorNListf (C++ function), 772  
 DP\_DeepTensorGetCutoff (C++ function), 773  
 DP\_DeepTensorGetNumbSelTypes (C++ function), 773  
 DP\_DeepTensorGetNumbTypes (C++ function), 774  
 DP\_DeepTensorGetOutputDim (C++ function), 774  
 DP\_DeepTensorGetSelTypes (C++ function), 774  
 DP\_DeepTensorGetTypeMap (C++ function), 775  
 DP\_DeleteChar (C++ function), 775  
 DP\_DeleteDeepPot (C++ function), 775  
 DP\_DeleteDeepPotModelDevi (C++ function), 775  
 DP\_DeleteDeepTensor (C++ function), 776  
 DP\_DeleteDipoleChargeModifier (C++ function), 776  
 DP\_DeleteNlist (C++ function), 776  
 DP\_DipoleChargeModifier (C++ struct), 724  
 DP\_DipoleChargeModifier (C++ type), 788  
 DP\_DipoleChargeModifier::dcm (C++ member), 724  
 DP\_DipoleChargeModifier::DP\_DipoleChargeModifier (C++ function), 724  
 DP\_DipoleChargeModifier::exception (C++ member), 724  
 DP\_DipoleChargeModifierCheckOK (C++ function), 777  
 DP\_DipoleChargeModifierComputeNList (C++ function), 777  
 DP\_DipoleChargeModifierComputeNListf (C++ function), 778  
 DP\_DipoleChargeModifierGetCutoff (C++ function), 779  
 DP\_DipoleChargeModifierGetNumbSelTypes (C++ function), 779  
 DP\_DipoleChargeModifierGetNumbTypes (C++ function), 779  
 DP\_DipoleChargeModifierGetSelTypes (C++ function), 780  
 dp\_ipi() (in module deepmd.entypoints.ipi), 339  
 DP\_NEW\_OK (C macro), 787  
 DP\_NewDeepPot (C++ function), 780  
 DP\_NewDeepPotModelDevi (C++ function), 780  
 DP\_NewDeepPotModelDeviWithParam (C++ function), 781  
 DP\_NewDeepPotWithParam (C++ function), 781  
 DP\_NewDeepPotWithParam2 (C++ function), 782  
 DP\_NewDeepTensor (C++ function), 782  
 DP\_NewDeepTensorWithParam (C++ function), 782  
 DP\_NewDipoleChargeModifier (C++ function), 783  
 DP\_NewDipoleChargeModifierWithParam (C++ function), 783  
 DP\_NewNlist (C++ function), 784  
 DP\_Nlist (C++ struct), 724  
 DP\_Nlist (C++ type), 788  
 DP\_Nlist::DP\_Nlist (C++ function), 725  
 DP\_Nlist::exception (C++ member), 725  
 DP\_Nlist::nl (C++ member), 725  
 DP\_NlistCheckOK (C++ function), 784  
 DP\_PrintSummary (C++ function), 784  
 DP\_ReadFileToChar (C++ function), 785  
 DP\_ReadFileToChar2 (C++ function), 785  
 DP\_REQUIRES\_OK (C macro), 787  
 DP\_SelectByType (C++ function), 785  
 DP\_SelectMapInt (C++ function), 786  
 DPAssert (C++ function), 841  
 DPerrcheck (C macro), 845  
 DPH5Path (class in deepmd.utils.path), 541  
 DPH5Path (class in deepmd\_utils.utils.path), 595  
 DPOSPath (class in deepmd.utils.path), 542  
 DPOSPath (class in deepmd\_utils.utils.path), 596  
 DPPPath (class in deepmd.utils.path), 543  
 DPPPath (class in deepmd\_utils.utils.path), 597

`dprc_pairwise_idx()` (in module `deepmd.env.op_module`), 638  
`DprcPairwiseIdx()` (in module `deepmd.env.op_module`), 613  
`DPTabulate` (class in `deepmd.utils.tabulate`), 548  
`DPTrainer` (class in `deepmd.train.trainer`), 508  
`dynamic_metadata()` (in module `back-end.dynamic_metadata`), 217

## E

`embed_atom_type()` (in module `deepmd.utils.type_embed`), 551  
`embedding_net()` (in module `deepmd.utils.network`), 538  
`embedding_net_rand_seed_shift()` (in module `deepmd.utils.network`), 539  
`EmbeddingNet` (in module `deepmd_utils.model_format`), 565  
`EmbeddingNet` (in module `deepmd_utils.model_format.network`), 573  
`enable:`  
   `nvnmmd/enable` (Argument), 130  
`enable_atom_ener_coeff:`  
   `loss[ener_spin]/enable_atom_ener_coeff` (Argument), 123  
   `loss[ener]/enable_atom_ener_coeff` (Argument), 121  
`enable_compression()` (`deepmd.descriptor.Descriptor` method), 230  
`enable_compression()` (`deepmd.descriptor.descriptor.Descriptor` method), 282  
`enable_compression()` (`deepmd.descriptor.DescrptHybrid` method), 235  
`enable_compression()` (`deepmd.descriptor.DescrptSeA` method), 246  
`enable_compression()` (`deepmd.descriptor.DescrptSeAtten` method), 266  
`enable_compression()` (`deepmd.descriptor.DescrptSeR` method), 272  
`enable_compression()` (`deepmd.descriptor.DescrptSeT` method), 276  
`enable_compression()` (`deepmd.descriptor.hybrid.DescrptHybrid` method), 287  
`enable_compression()` (`deepmd.descriptor.se_a.DescrptSeA` method), 300  
`enable_compression()` (`deepmd.descriptor.se_atten.DescrptSeAtten` method), 320  
`enable_compression()` (`deepmd.descriptor.se_r.DescrptSeR` method), 326  
`enable_compression()` (`deepmd.descriptor.se_t.DescrptSeT` method), 330  
`enable_compression()` (`deepmd.model.frozen.FrozenModel` method), 454  
`enable_compression()` (`deepmd.model.linear.LinearModel` method), 457  
`enable_compression()` (`deepmd.model.model.Model` method), 462  
`enable_compression()` (`deepmd.model.model.StandardModel` method), 464  
`enable_compression()` (`deepmd.model.pairtab.PairTabModel` method), 471  
`enable_compression()` (`deepmd.model.pairwise_dprc.PairwiseDPRc` method), 473  
`enable_mixed_precision()` (`deepmd.descriptor.Descriptor` method), 230  
`enable_mixed_precision()` (`deepmd.descriptor.descriptor.Descriptor` method), 282  
`enable_mixed_precision()` (`deepmd.descriptor.DescrptHybrid` method), 236  
`enable_mixed_precision()` (`deepmd.descriptor.DescrptSeA` method), 247  
`enable_mixed_precision()` (`deepmd.descriptor.hybrid.DescrptHybrid` method), 288  
`enable_mixed_precision()` (`deepmd.descriptor.se_a.DescrptSeA` method), 301  
`enable_mixed_precision()` (`deepmd.fit.dipole.DipoleFittingSeA` method), 359  
`enable_mixed_precision()` (`deepmd.fit.DipoleFittingSeA` method), 347  
`enable_mixed_precision()` (`deepmd.fit.dos.DOSFitting` method),

362  
 enable\_mixed\_precision() (deepmd.fit.DOSFitting method), 345  
 enable\_mixed\_precision() (deepmd.fit.ener.EnerFitting method), 366  
 enable\_mixed\_precision() (deepmd.fit.EnerFitting method), 351  
 enable\_mixed\_precision() (deepmd.fit.GlobalPolarFittingSeA method), 354  
 enable\_mixed\_precision() (deepmd.fit.polar.GlobalPolarFittingSeA method), 369  
 enable\_mixed\_precision() (deepmd.fit.polar.PolarFittingSeA method), 372  
 enable\_mixed\_precision() (deepmd.fit.PolarFittingSeA method), 357  
 enable\_mixed\_precision() (deepmd.model.model.Model method), 462  
 enable\_mixed\_precision() (deepmd.model.model.StandardModel method), 464  
 enable\_mixed\_precision() (deepmd.model.multi.MultiModel method), 468  
 enable\_mixed\_precision() (deepmd.model.MultiModel method), 443  
 enable\_profiler:  
   training/enable\_profiler (Argument), 129  
 Encode (class in deepmd.nvnmd.utils), 493  
 Encode (class in deepmd.nvnmd.utils.encode), 500  
 EnerDipoleLoss (class in deepmd.loss), 421  
 EnerDipoleLoss (class in deepmd.loss.ener), 427  
 EnerFitting (class in deepmd.fit), 348  
 EnerFitting (class in deepmd.fit.ener), 363  
 EnerModel (class in deepmd.model), 437  
 EnerModel (class in deepmd.model.ener), 449  
 EnerSpinLoss (class in deepmd.loss), 422  
 EnerSpinLoss (class in deepmd.loss.ener), 428  
 EnerStdLoss (class in deepmd.loss), 423  
 EnerStdLoss (class in deepmd.loss.ener), 429  
 env\_mat\_a (C++ function), 842  
 env\_mat\_r (C++ function), 842  
 EnvMat (class in deepmd\_utils.model\_format), 565  
 EnvMat (class in deepmd\_utils.model\_format.env\_mat), 571  
 eval() (deepmd.DipoleChargeModifier method), 224  
 eval() (deepmd.infer.data\_modifier.DipoleChargeModifier method), 393  
 eval() (deepmd.infer.deep\_dos.DeepDOS method), 396  
 eval() (deepmd.infer.deep\_polar.DeepGlobalPolar method), 403  
 eval() (deepmd.infer.deep\_pot.DeepPot method), 407  
 eval() (deepmd.infer.deep\_tensor.DeepTensor method), 410  
 eval() (deepmd.infer.DeepDOS method), 373  
 eval() (deepmd.infer.DeepGlobalPolar method), 381  
 eval() (deepmd.infer.DeepPot method), 384  
 eval() (deepmd.infer.DipoleChargeModifier method), 389  
 eval() (deepmd.infer.ewald\_recip.EwaldRecp method), 414  
 eval() (deepmd.infer.EwaldRecp method), 390  
 eval() (deepmd.loss.dos.DOSLoss method), 427  
 eval() (deepmd.loss.DOSLoss method), 420  
 eval() (deepmd.loss.ener.EnerDipoleLoss method), 428  
 eval() (deepmd.loss.ener.EnerSpinLoss method), 429  
 eval() (deepmd.loss.ener.EnerStdLoss method), 431  
 eval() (deepmd.loss.EnerDipoleLoss method), 421  
 eval() (deepmd.loss.EnerSpinLoss method), 423  
 eval() (deepmd.loss.EnerStdLoss method), 425  
 eval() (deepmd.loss.loss.Loss method), 432  
 eval() (deepmd.loss.tensor.TensorLoss method), 433  
 eval() (deepmd.loss.TensorLoss method), 426  
 eval\_descriptor() (deepmd.infer.deep\_dos.DeepDOS method), 397  
 eval\_descriptor() (deepmd.infer.deep\_pot.DeepPot method), 408  
 eval\_descriptor() (deepmd.infer.DeepDOS method), 374  
 eval\_descriptor() (deepmd.infer.DeepPot method), 385  
 eval\_full() (deepmd.infer.deep\_tensor.DeepTensor method), 411  
 eval\_single\_list() (deepmd.train.trainer.DPTrainer static method), 508  
 eval\_typeebd() (deepmd.DeepEval method), 220  
 eval\_typeebd() (deepmd.infer.deep\_eval.DeepEval method), 400  
 eval\_typeebd() (deepmd.infer.DeepEval method), 378  
 ewald\_beta:  
   model/modifier[dipole\_charge]/ewald\_beta (Argument), 91  
   model/modifier[dipole\_charge]/ewald\_h (Argument), 92  
 ewald\_recip() (in module deepmd.env.op\_module), 638



EwaldRecp (class in deepmd.infer), 390  
 EwaldRecp (class in deepmd.infer.ewald\_rec), 413  
 EwaldRecp() (in module deepmd.env.op\_module), 613  
 exclude\_types:  
   model[standard]/descriptor[se\_a\_ebd\_v2]/exclude\_types (in module deepmd.nvnmd.descriptor.se\_a), 482  
   (model), 109  
   model[standard]/descriptor[se\_a\_mask]/exclude\_types (in module deepmd.nvnmd.descriptor.se\_atten), 482  
   (model), 110  
   model[standard]/descriptor[se\_a\_tpe]/exclude\_types (in module deepmd.nvnmd.descriptor.se\_a), 482  
   (model), 99  
   model[standard]/descriptor[se\_atten\_v2]/exclude\_types (in module deepmd.nvnmd.descriptor.se\_atten), 482  
   (model), 106  
   model[standard]/descriptor[se\_atten]/exclude\_types (in module deepmd.nvnmd.entrypoints.freeze), 487  
   (model), 103  
   model[standard]/descriptor[se\_e2\_a]/exclude\_types (in module deepmd.nvnmd.entrypoints.freeze), 487  
   (model), 96  
   model[standard]/descriptor[se\_e2\_r]/exclude\_types (in module deepmd.nvnmd.entrypoints.freeze), 487  
   (model), 101  
 execute() (deepmd\_utils.utils.batch\_size.AutoBatchSize method), 584  
 execute\_all() (deepmd\_utils.utils.batch\_size.AutoBatchSize method), 585  
 exits() (deepmd.nvnmd.utils.fio.Fio method), 502  
 expand\_sys\_str() (in module deepmd.common), 555  
 expand\_sys\_str() (in module deepmd\_utils.common), 602  
 explicit\_ntypes (deepmd.descriptor.Descriptor property), 230  
 explicit\_ntypes (deepmd.descriptor.descriptor.Descriptor property), 282  
 explicit\_ntypes (deepmd.descriptor.DescriptHybrid property), 236  
 explicit\_ntypes (deepmd.descriptor.DescriptSeA property), 247  
 explicit\_ntypes (deepmd.descriptor.DescriptSeAtten property), 267  
 explicit\_ntypes (deepmd.descriptor.hybrid.DescriptHybrid property), 288  
 explicit\_ntypes (deepmd.descriptor.se\_a.DescriptSeA property), 301  
 explicit\_ntypes (deepmd.descriptor.se\_atten.DescriptSeAtten property), 321  
 extend\_bin() (deepmd.nvnmd.utils.Encode method), 494  
 extend\_bin() (deepmd.nvnmd.utils.encode.Encode method), 501  
 extend\_hex() (deepmd.nvnmd.utils.Encode method), 494  
 extend\_hex() (deepmd.nvnmd.utils.encode.Encode method), 501  
 extend\_list() (deepmd.nvnmd.utils.Encode method), 494  
 extend\_list() (deepmd.nvnmd.utils.encode.Encode method), 501  
 F  
 filter\_GR2D() (in module deepmd.nvnmd.descriptor.se\_a), 482  
 filter\_GR2D() (in module deepmd.nvnmd.descriptor.se\_atten), 482  
 filter\_lower\_R42GR() (in module deepmd.nvnmd.descriptor.se\_a), 482  
 filter\_lower\_R42GR() (in module deepmd.nvnmd.descriptor.se\_atten), 482  
 filter\_tensorVariableList() (in module deepmd.nvnmd.entrypoints.freeze), 487  
 find\_max\_expo (C++ function), 842  
 find\_max\_expo() (deepmd.nvnmd.utils.Encode method), 494  
 find\_max\_expo() (deepmd.nvnmd.utils.encode.Encode method), 501  
 find\_tensorflow() (in module back-end.find\_tensorflow), 217  
 FioBin (class in deepmd.nvnmd.utils), 495  
 FioBin (class in deepmd.nvnmd.utils.fio), 502  
 FioDic (class in deepmd.nvnmd.utils), 495  
 FioDic (class in deepmd.nvnmd.utils.fio), 503  
 FioJsonDic (class in deepmd.nvnmd.utils.fio), 503  
 FioNpyDic (class in deepmd.nvnmd.utils.fio), 504  
 FioTxt (class in deepmd.nvnmd.utils), 496  
 FioTxt (class in deepmd.nvnmd.utils.fio), 504  
 fit\_diag:  
   model[standard]/fitting\_net[polar]/fit\_diag (Argument), 115  
 Fitting (class in deepmd.fit), 351  
 Fitting (class in deepmd.fit.fitting), 366  
 fitting\_check\_output() (in module deepmd\_utils.model\_format), 570  
 fitting\_check\_output() (in module deepmd\_utils.model\_format.output\_def), 578  
 fitting\_dipole() (in module deepmd\_utils.utils.argcheck), 582  
 fitting\_dos() (in module deepmd\_utils.utils.argcheck), 582  
 fitting\_ener() (in module deepmd\_utils.utils.argcheck), 582  
 fitting\_net:  
   model[standard]/fitting\_net (Argument), 111  
 fitting\_net\_dict:  
   model[multi]/fitting\_net\_dict (Argument), 117  
 fitting\_polar() (in module deepmd\_utils.utils.argcheck), 582

fitting\_variant\_type\_args() (in module  
 deepmd\_utils.utils.argcheck), 582  
 fitting\_weight:  
 training/fitting\_weight (Argument), 129  
 FittingNet (in module  
 deepmd\_utils.model\_format), 566  
 FittingNet (in module  
 deepmd\_utils.model\_format.network),  
 573  
 FittingOutputDef (class in  
 deepmd\_utils.model\_format), 566  
 FittingOutputDef (class in  
 deepmd\_utils.model\_format.output\_def),  
 576  
 flt2bin() (deepmd.nvnmd.utils.Encode method),  
 494  
 flt2bin() (deepmd.nvnmd.utils.encode.Encode  
 method), 501  
 flt2bin\_one() (deepmd.nvnmd.utils.Encode  
 method), 494  
 flt2bin\_one() (deepmd.nvnmd.utils.encode.Encode  
 method), 501  
 FLT\_MASK (C macro), 845  
 flt\_nvnmd() (in module deepmd.env.op\_module),  
 639  
 FltNvnmd() (in module deepmd.env.op\_module), 613  
 format\_nlist\_i\_cpu (C++ function), 843  
 format\_nlist\_i\_fill\_a (C++ function), 843  
 freeze() (in module deepmd.entripoints), 333  
 freeze() (in module deepmd.entripoints.freeze),  
 338  
 frozen\_model\_args() (in module  
 deepmd\_utils.utils.argcheck), 582  
 FrozenModel (class in deepmd.model.frozen), 452  
**G**  
 gather\_placeholder() (in module  
 deepmd.model.pairwise\_dprc), 474  
 gelu() (in module deepmd.common), 556  
 Gelu() (in module deepmd.env.op\_module), 614  
 gelu() (in module deepmd.env.op\_module), 639  
 gelu\_custom() (in module deepmd.env.op\_module),  
 639  
 gelu\_grad() (in module deepmd.env.op\_module),  
 639  
 gelu\_grad\_custom() (in module  
 deepmd.env.op\_module), 640  
 gelu\_grad\_grad() (in module  
 deepmd.env.op\_module), 640  
 gelu\_grad\_grad\_custom() (in module  
 deepmd.env.op\_module), 640  
 gelu\_tf() (in module deepmd.common), 556  
 GeluCustom() (in module deepmd.env.op\_module),  
 614  
 GeluGPUExecuteFunctor (C++ struct), 798  
 GeluGPUExecuteFunctor::operator() (C++ func-  
 tion), 799  
 GeluGrad() (in module deepmd.env.op\_module), 614  
 GeluGradCustom() (in module  
 deepmd.env.op\_module), 614  
 GeluGradGPUExecuteFunctor (C++ struct), 799  
 GeluGradGPUExecuteFunctor::operator() (C++  
 function), 799  
 GeluGradGrad() (in module  
 deepmd.env.op\_module), 614  
 GeluGradGradCustom() (in module  
 deepmd.env.op\_module), 615  
 GeluGradGradGPUExecuteFunctor (C++ struct), 799  
 GeluGradGradGPUExecuteFunctor::operator()  
 (C++ function), 799  
 gen\_args() (in module deepmd.utils.argcheck), 519  
 gen\_args() (in module  
 deepmd\_utils.utils.argcheck), 582  
 gen\_doc() (in module deepmd.utils.argcheck), 519  
 gen\_doc() (in module deepmd\_utils.utils.argcheck),  
 582  
 gen\_json() (in module deepmd.utils.argcheck), 519  
 gen\_json() (in module  
 deepmd\_utils.utils.argcheck), 582  
 generate() (deepmd.utils.parallel\_op.ParallelOp  
 method), 541  
 get() (deepmd.nvnmd.utils.fio.FioDic method), 503  
 get() (deepmd.nvnmd.utils.FioDic method), 495  
 get() (deepmd.utils.pair\_tab.PairTab method), 539  
 get() (deepmd.utils.PairTab method), 517  
 get() (deepmd\_utils.utils.pair\_tab.PairTab  
 method), 594  
 get\_activation\_func() (in module  
 deepmd.common), 556  
 get\_all\_argument() (deepmd\_utils.utils.argcheck.ArgsPlugin  
 method), 581  
 get\_argument\_from\_env() (in module back-  
 end.read\_env), 218  
 get\_atom\_type() (deepmd.utils.data.DeepmdData  
 method), 525  
 get\_atom\_type() (deepmd.utils.DeepmdData  
 method), 512  
 get\_atom\_type() (deepmd\_utils.utils.data.DeepmdData  
 method), 589  
 get\_attention\_layer\_nodes\_from\_graph\_def()  
 (in module deepmd.utils.graph), 530  
 get\_attention\_layer\_variables\_from\_graph\_def()  
 (in module deepmd.utils.graph), 530  
 get\_batch() (deepmd.utils.data.DeepmdData  
 method), 525  
 get\_batch() (deepmd.utils.data\_system.DeepmdDataSystem  
 method), 528  
 get\_batch() (deepmd.utils.DeepmdData method),

512  
 get\_batch() (deepmd.utils.DeepmdDataSystem method), 514  
 get\_batch() (deepmd\_utils.utils.data.DeepmdData method), 589  
 get\_batch() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 592  
 get\_batch\_mixed() (deepmd.utils.data\_system.DeepmdDataSystem method), 528  
 get\_batch\_mixed() (deepmd.utils.DeepmdDataSystem method), 515  
 get\_batch\_mixed() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 592  
 get\_batch\_size() (deepmd.utils.data\_system.DeepmdDataSystem method), 528  
 get\_batch\_size() (deepmd.utils.DeepmdDataSystem method), 515  
 get\_batch\_size() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 592  
 get\_batch\_standard() (deepmd.utils.data\_system.DeepmdDataSystem method), 528  
 get\_batch\_standard() (deepmd.utils.DeepmdDataSystem method), 515  
 get\_batch\_standard() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 592  
 get\_class\_by\_input() (deepmd.descriptor.Descriptor class method), 230  
 get\_class\_by\_input() (deepmd.descriptor.descriptor.Descriptor class method), 282  
 get\_class\_by\_input() (deepmd.model.model.Model class method), 462  
 get\_constant\_initializer() (in module deepmd.nvnmd.utils.weight), 505  
 get\_data() (deepmd\_utils.model\_format.FittingOutputDef method), 566  
 get\_data() (deepmd\_utils.model\_format.ModelOutputDef method), 566  
 get\_data() (deepmd\_utils.model\_format.output\_def.FittingOutputDef method), 576  
 get\_data() (deepmd\_utils.model\_format.output\_def.ModelOutputDef method), 577  
 get\_data\_dict() (deepmd.train.trainer.DatasetLoader method), 509  
 get\_data\_dict() (deepmd.utils.data.DeepmdData method), 525  
 get\_data\_dict() (deepmd.utils.data\_system.DeepmdDataSystem method), 528  
 get\_data\_dict() (deepmd.utils.DeepmdData method), 512  
 get\_data\_dict() (deepmd.utils.DeepmdDataSystem method), 515  
 get\_data\_dict() (deepmd\_utils.utils.data.DeepmdData method), 589  
 get\_data\_dict() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 592  
 get\_data\_system() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498  
 get\_deriv\_name() (in module deepmd\_utils.model\_format), 570  
 get\_descriptor() (in module deepmd\_utils.model\_format.output\_def), 578  
 get\_descriptor\_type() (deepmd.infer.deep\_pot.DeepPot method), 409  
 get\_descriptor\_type() (deepmd.infer.DeepPot method), 386  
 get\_dim\_aparam() (deepmd.infer.deep\_dipole.DeepDipole method), 395  
 get\_dim\_aparam() (deepmd.infer.deep\_dos.DeepDOS method), 398  
 get\_dim\_aparam() (deepmd.infer.deep\_polar.DeepGlobalPolar method), 403  
 get\_dim\_aparam() (deepmd.infer.deep\_polar.DeepPolar method), 405  
 get\_dim\_aparam() (deepmd.infer.deep\_pot.DeepPot method), 409  
 get\_dim\_aparam() (deepmd.infer.deep\_tensor.DeepTensor method), 412  
 get\_dim\_aparam() (deepmd.infer.deep\_wfc.DeepWFC method), 413  
 get\_dim\_aparam() (deepmd.infer.DeepDipole method), 376  
 get\_dim\_aparam() (deepmd.infer.DeepDOS method), 375  
 get\_dim\_aparam() (deepmd.infer.DeepGlobalPolar method), 381  
 get\_dim\_aparam() (deepmd.infer.DeepPolar method), 383  
 get\_dim\_aparam() (deepmd.infer.DeepPot method), 386  
 get\_dim\_aparam() (deepmd.infer.DeepWFC method), 388  
 get\_dim\_aparam() (deepmd.infer.deep\_dipole.DeepDipole method), 395  
 get\_dim\_fparam() (deepmd.infer.deep\_dos.DeepDOS method), 398  
 get\_dim\_fparam() (deepmd.infer.deep\_polar.DeepGlobalPolar method), 404  
 get\_dim\_fparam() (deepmd.infer.deep\_polar.DeepPolar method), 405  
 get\_dim\_fparam() (deepmd.infer.deep\_pot.DeepPot method), 405

method), 409

get\_dim\_fparam() (deepmd.infer.deep\_tensor.DeepTensor method), 412

get\_dim\_fparam() (deepmd.infer.deep\_wfc.DeepWFC method), 413

get\_dim\_fparam() (deepmd.infer.DeepDipole method), 376

get\_dim\_fparam() (deepmd.infer.DeepDOS method), 375

get\_dim\_fparam() (deepmd.infer.DeepGlobalPolar method), 382

get\_dim\_fparam() (deepmd.infer.DeepPolar method), 383

get\_dim\_fparam() (deepmd.infer.DeepPot method), 386

get\_dim\_fparam() (deepmd.infer.DeepWFC method), 388

get\_dim\_out() (deepmd.descriptor.Descriptor method), 230

get\_dim\_out() (deepmd.descriptor.descriptor.Descriptor method), 282

get\_dim\_out() (deepmd.descriptor.DescriptHybrid method), 236

get\_dim\_out() (deepmd.descriptor.DescriptLocFrame method), 241

get\_dim\_out() (deepmd.descriptor.DescriptSeA method), 247

get\_dim\_out() (deepmd.descriptor.DescriptSeAEf method), 255

get\_dim\_out() (deepmd.descriptor.DescriptSeR method), 273

get\_dim\_out() (deepmd.descriptor.DescriptSeT method), 277

get\_dim\_out() (deepmd.descriptor.hybrid.DescriptHybrid method), 288

get\_dim\_out() (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 293

get\_dim\_out() (deepmd.descriptor.se\_a.DescriptSeA method), 301

get\_dim\_out() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 309

get\_dim\_out() (deepmd.descriptor.se\_r.DescriptSeR method), 327

get\_dim\_out() (deepmd.descriptor.se\_t.DescriptSeT method), 331

get\_dim\_rot\_mat\_1() (deepmd.descriptor.Descriptor method), 231

get\_dim\_rot\_mat\_1() (deepmd.descriptor.descriptor.Descriptor method), 283

get\_dim\_rot\_mat\_1() (deepmd.descriptor.DescriptSeA method), 247

get\_dim\_rot\_mat\_1() (deepmd.descriptor.DescriptSeAEf method), 255

get\_dim\_rot\_mat\_1() (deepmd.descriptor.se\_a.DescriptSeA method), 301

get\_dim\_rot\_mat\_1() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 309

get\_dp\_init\_weights() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498

get\_dscp\_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498

get\_embedding\_net\_nodes() (in module deepmd.utils.graph), 530

get\_embedding\_net\_nodes\_from\_graph\_def() (in module deepmd.utils.graph), 531

get\_embedding\_net\_variables() (in module deepmd.utils.graph), 531

get\_embedding\_net\_variables\_from\_graph\_def() (in module deepmd.utils.graph), 531

get\_env() (in module deepmd.lmp), 559

get\_evaluation\_results() (deepmd.train.trainer.DPTrainer method), 508

get\_extra\_embedding\_net\_suffix() (in module deepmd.utils.graph), 531

get\_extra\_embedding\_net\_variables\_from\_graph\_def() (in module deepmd.utils.graph), 532

get\_extra\_side\_embedding\_net\_variable() (in module deepmd.utils.compress), 521

get\_feed\_dict() (deepmd.model.model.Model method), 462

get\_feed\_dict() (deepmd.model.pairwise\_dprc.PairwiseDPRc method), 473

get\_feed\_dict() (deepmd.train.trainer.DPTrainer method), 508

get\_file\_list() (deepmd.nvnmd.utils.fio.Fio method), 502

get\_filter\_type\_weight() (in module deepmd.nvnmd.utils.weight), 505

get\_filter\_weight() (in module deepmd.nvnmd.utils), 496

get\_filter\_weight() (in module deepmd.nvnmd.utils.weight), 505

get\_fitn\_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 498

get\_fitnet\_weight() (in module deepmd.nvnmd.utils), 496

get\_fitnet\_weight() (in module deepmd.nvnmd.utils.weight), 506

get\_fitting() (deepmd.model.frozen.FrozenModel method), 454

`get_fitting()` (deepmd.model.linear.LinearModel method), 457  
`get_fitting()` (deepmd.model.model.Model method), 462  
`get_fitting()` (deepmd.model.model.StandardModel method), 465  
`get_fitting()` (deepmd.model.multi.MultiModel method), 468  
`get_fitting()` (deepmd.model.MultiModel method), 443  
`get_fitting()` (deepmd.model.pairtab.PairTabModel method), 471  
`get_fitting()` (deepmd.model.pairwise\_dprc.PairwiseDPRc method), 474  
`get_fitting_net_nodes()` (in module deepmd.utils.graph), 532  
`get_fitting_net_nodes_from_graph_def()` (in module deepmd.utils.graph), 532  
`get_fitting_net_variables()` (in module deepmd.utils.graph), 533  
`get_fitting_net_variables_from_graph_def()` (in module deepmd.utils.graph), 533  
`get_global_step()` (deepmd.train.trainer.DPTrainer method), 508  
`get_gpus()` (in module deepmd.cluster.local), 225  
`get_learning_rate_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 498  
`get_library_path()` (in module deepmd.lmp), 559  
`get_ll()` (in module deepmd.entrypoints.main), 339  
`get_ll()` (in module deepmd\_utils.main), 604  
`get_loss()` (deepmd.fit.dipole.DipoleFittingSeA method), 359  
`get_loss()` (deepmd.fit.DipoleFittingSeA method), 347  
`get_loss()` (deepmd.fit.dos.DOSFitting method), 362  
`get_loss()` (deepmd.fit.DOSFitting method), 345  
`get_loss()` (deepmd.fit.ener.EnerFitting method), 366  
`get_loss()` (deepmd.fit.EnerFitting method), 351  
`get_loss()` (deepmd.fit.Fitting method), 352  
`get_loss()` (deepmd.fit.fitting.Fitting method), 367  
`get_loss()` (deepmd.fit.GlobalPolarFittingSeA method), 354  
`get_loss()` (deepmd.fit.polar.GlobalPolarFittingSeA method), 369  
`get_loss()` (deepmd.fit.polar.PolarFittingSeA method), 372  
`get_loss()` (deepmd.fit.PolarFittingSeA method), 357  
`get_loss()` (deepmd.model.frozen.FrozenModel method), 454  
`get_loss()` (deepmd.model.linear.LinearModel method), 457  
`get_loss()` (deepmd.model.model.Model method), 463  
`get_loss()` (deepmd.model.model.StandardModel method), 465  
`get_loss()` (deepmd.model.multi.MultiModel method), 468  
`get_loss()` (deepmd.model.MultiModel method), 443  
`get_loss()` (deepmd.model.pairtab.PairTabModel method), 471  
`get_loss()` (deepmd.model.pairwise\_dprc.PairwiseDPRc method), 474  
`get_loss_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 498  
`get_model_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 498  
`get_natoms()` (deepmd.utils.data.DeepmdData method), 525  
`get_natoms()` (deepmd.utils.DeepmdData method), 512  
`get_natoms()` (deepmd\_utils.utils.data.DeepmdData method), 589  
`get_natoms_vec()` (deepmd.utils.data.DeepmdData method), 525  
`get_natoms_vec()` (deepmd.utils.DeepmdData method), 512  
`get_natoms_vec()` (deepmd\_utils.utils.data.DeepmdData method), 589  
`get_nbatches()` (deepmd.utils.data\_system.DeepmdDataSystem method), 528  
`get_nbatches()` (deepmd.utils.DeepmdDataSystem method), 515  
`get_nbatches()` (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 592  
`get_nlist()` (deepmd.descriptor.Descriptor method), 231  
`get_nlist()` (deepmd.descriptor.descriptor.Descriptor method), 283  
`get_nlist()` (deepmd.descriptor.DescriptHybrid method), 236  
`get_nlist()` (deepmd.descriptor.DescriptLocFrame method), 241  
`get_nlist()` (deepmd.descriptor.DescriptSeA method), 247  
`get_nlist()` (deepmd.descriptor.DescriptSeAEf method), 255  
`get_nlist()` (deepmd.descriptor.DescriptSeR method), 273  
`get_nlist()` (deepmd.descriptor.DescriptSeT method), 277  
`get_nlist()` (deepmd.descriptor.hybrid.DescriptHybrid method), 288  
`get_nlist()` (deepmd.descriptor.loc\_frame.DescriptLocFrame



method), 293

get\_nlist() (deepmd.descriptor.se\_a.DescriptSeA method), 301

get\_nlist() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 309

get\_nlist() (deepmd.descriptor.se\_r.DescriptSeR method), 327

get\_nlist() (deepmd.descriptor.se\_t.DescriptSeT method), 331

get\_nlist\_i() (deepmd.descriptor.DescriptHybrid method), 236

get\_nlist\_i() (deepmd.descriptor.hybrid.DescriptHybrid method), 288

get\_normalize() (in module deepmd.nvnmd.utils.weight), 506

get\_np\_precision() (in module deepmd.common), 556

get\_np\_precision() (in module deepmd\_utils.common), 602

get\_nsystems() (deepmd.utils.data\_system.DeepmdDataSystem method), 529

get\_nsystems() (deepmd.utils.DeepmdDataSystem method), 515

get\_nsystems() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 593

get\_ntypes() (deepmd.descriptor.Descriptor method), 231

get\_ntypes() (deepmd.descriptor.descriptor.Descriptor method), 283

get\_ntypes() (deepmd.descriptor.DescriptHybrid method), 237

get\_ntypes() (deepmd.descriptor.DescriptLocFrame method), 241

get\_ntypes() (deepmd.descriptor.DescriptSeA method), 247

get\_ntypes() (deepmd.descriptor.DescriptSeAEf method), 255

get\_ntypes() (deepmd.descriptor.DescriptSeR method), 273

get\_ntypes() (deepmd.descriptor.DescriptSeT method), 277

get\_ntypes() (deepmd.descriptor.hybrid.DescriptHybrid method), 289

get\_ntypes() (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 293

get\_ntypes() (deepmd.descriptor.se\_a.DescriptSeA method), 301

get\_ntypes() (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 309

get\_ntypes() (deepmd.descriptor.se\_r.DescriptSeR method), 327

get\_ntypes() (deepmd.descriptor.se\_t.DescriptSeT method), 331

get\_ntypes() (deepmd.infer.deep\_dos.DeepDOS method), 398

get\_ntypes() (deepmd.infer.deep\_pot.DeepPot method), 409

get\_ntypes() (deepmd.infer.deep\_tensor.DeepTensor method), 412

get\_ntypes() (deepmd.infer.DeepDOS method), 375

get\_ntypes() (deepmd.infer.DeepPot method), 386

get\_ntypes() (deepmd.model.dos.DOSModel method), 448

get\_ntypes() (deepmd.model.DOSModel method), 436

get\_ntypes() (deepmd.model.ener.EnerModel method), 451

get\_ntypes() (deepmd.model.EnerModel method), 440

get\_ntypes() (deepmd.model.frozen.FrozenModel method), 454

get\_ntypes() (deepmd.model.linear.LinearModel method), 457

get\_ntypes() (deepmd.model.model.Model method), 463

get\_ntypes() (deepmd.model.model.StandardModel method), 465

get\_ntypes() (deepmd.model.multi.MultiModel method), 468

get\_ntypes() (deepmd.model.MultiModel method), 443

get\_ntypes() (deepmd.model.pairtab.PairTabModel method), 471

get\_ntypes() (deepmd.model.pairwise\_dprc.PairwiseDPRC method), 474

get\_ntypes() (deepmd.model.tensor.TensorModel method), 479

get\_ntypes() (deepmd.utils.data.DeepmdData method), 525

get\_ntypes() (deepmd.utils.data\_system.DeepmdDataSystem method), 529

get\_ntypes() (deepmd.utils.DeepmdData method), 512

get\_ntypes() (deepmd.utils.DeepmdDataSystem method), 515

get\_ntypes() (deepmd\_utils.utils.data.DeepmdData method), 589

get\_ntypes() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 593

get\_ntypes\_spin() (deepmd.infer.deep\_pot.DeepPot method), 409

get\_ntypes\_spin() (deepmd.infer.DeepPot method), 386

get\_ntypes\_spin() (deepmd.utils.spin.Spin method), 548

get\_num\_aparam() (deepmd.fit.dos.DOSFitting method), 362

get\_num\_aparam() (deepmd.fit.DOSFitting method), 362

method), 345

get\_num\_aparam() (deepmd.fit.ener.EnerFitting method), 366

get\_num\_aparam() (deepmd.fit.EnerFitting method), 351

get\_num\_aparam() (deepmd.model.dos.DOSModel method), 448

get\_num\_aparam() (deepmd.model.DOSModel method), 436

get\_num\_aparam() (deepmd.model.ener.EnerModel method), 451

get\_num\_aparam() (deepmd.model.EnerModel method), 440

get\_num\_aparam() (deepmd.model.model.Model method), 463

get\_num\_aparam() (deepmd.model.multi.MultiModel method), 468

get\_num\_aparam() (deepmd.model.MultiModel method), 443

get\_num\_batch() (deepmd.utils.data.DeepmdData method), 525

get\_num\_batch() (deepmd.utils.DeepmdData method), 512

get\_num\_batch() (deepmd\_utils.utils.data.DeepmdData method), 589

get\_num\_dos() (deepmd.fit.dos.DOSFitting method), 362

get\_num\_dos() (deepmd.fit.DOSFitting method), 345

get\_num\_dos() (deepmd.infer.deep\_dos.DeepDOS method), 398

get\_num\_dos() (deepmd.infer.DeepDOS method), 375

get\_num\_dos() (deepmd.model.dos.DOSModel method), 448

get\_num\_dos() (deepmd.model.DOSModel method), 436

get\_num\_dos() (deepmd.model.model.Model method), 463

get\_num\_dos() (deepmd.model.multi.MultiModel method), 468

get\_num\_dos() (deepmd.model.MultiModel method), 443

get\_num\_fparam() (deepmd.fit.dos.DOSFitting method), 362

get\_num\_fparam() (deepmd.fit.DOSFitting method), 345

get\_num\_fparam() (deepmd.fit.ener.EnerFitting method), 366

get\_num\_fparam() (deepmd.fit.EnerFitting method), 351

get\_num\_fparam() (deepmd.model.dos.DOSModel method), 448

get\_num\_fparam() (deepmd.model.DOSModel method), 436

get\_num\_fparam() (deepmd.model.ener.EnerModel method), 451

get\_num\_fparam() (deepmd.model.EnerModel method), 440

get\_num\_fparam() (deepmd.model.model.Model method), 463

get\_num\_fparam() (deepmd.model.multi.MultiModel method), 468

get\_num\_fparam() (deepmd.model.MultiModel method), 443

get\_num\_set() (deepmd.utils.data.DeepmdData method), 525

get\_num\_set() (deepmd.utils.DeepmdData method), 512

get\_num\_set() (deepmd\_utils.utils.data.DeepmdData method), 589

get\_nvnm\_jdata() (deepmd.nvnm.utils.config.NvnmConfig method), 499

get\_op\_dir() (in module deepmd.lmp), 560

get\_out\_size() (deepmd.fit.dipole.DipoleFittingSeA method), 359

get\_out\_size() (deepmd.fit.DipoleFittingSeA method), 347

get\_out\_size() (deepmd.fit.GlobalPolarFittingSeA method), 355

get\_out\_size() (deepmd.fit.polar.GlobalPolarFittingSeA method), 370

get\_out\_size() (deepmd.fit.polar.PolarFittingSeA method), 372

get\_out\_size() (deepmd.fit.PolarFittingSeA method), 357

get\_out\_size() (deepmd.model.tensor.TensorModel method), 479

get\_pattern\_nodes\_from\_graph\_def() (in module deepmd.utils.graph), 533

get\_plugin() (deepmd.utils.Plugin method), 518

get\_plugin() (deepmd.utils.plugin.Plugin method), 545

get\_plugin() (deepmd\_utils.utils.plugin.Plugin method), 599

get\_precision() (in module deepmd.common), 557

get\_rcut() (deepmd.descriptor.Descriptor method), 231

get\_rcut() (deepmd.descriptor.descriptor.Descriptor method), 283

get\_rcut() (deepmd.descriptor.DescriptHybrid method), 237

get\_rcut() (deepmd.descriptor.DescriptLocFrame method), 242

get\_rcut() (deepmd.descriptor.DescriptSeA method), 247

get\_rcut() (deepmd.descriptor.DescriptSeAEf method), 256

`get_rcut()` (deepmd.descriptor.DescriptSeAMask method), 261  
`get_rcut()` (deepmd.descriptor.DescriptSeR method), 273  
`get_rcut()` (deepmd.descriptor.DescriptSeT method), 277  
`get_rcut()` (deepmd.descriptor.hybrid.DescriptHybrid method), 289  
`get_rcut()` (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 294  
`get_rcut()` (deepmd.descriptor.se\_a.DescriptSeA method), 301  
`get_rcut()` (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 310  
`get_rcut()` (deepmd.descriptor.se\_a\_mask.DescriptSeAMask method), 315  
`get_rcut()` (deepmd.descriptor.se\_r.DescriptSeR method), 327  
`get_rcut()` (deepmd.descriptor.se\_t.DescriptSeT method), 331  
`get_rcut()` (deepmd.infer.deep\_dos.DeepDOS method), 398  
`get_rcut()` (deepmd.infer.deep\_pot.DeepPot method), 409  
`get_rcut()` (deepmd.infer.deep\_tensor.DeepTensor method), 412  
`get_rcut()` (deepmd.infer.DeepDOS method), 375  
`get_rcut()` (deepmd.infer.DeepPot method), 386  
`get_rcut()` (deepmd.model.dos.DOSModel method), 448  
`get_rcut()` (deepmd.model.DOSModel method), 436  
`get_rcut()` (deepmd.model.ener.EnerModel method), 451  
`get_rcut()` (deepmd.model.EnerModel method), 440  
`get_rcut()` (deepmd.model.frozen.FrozenModel method), 454  
`get_rcut()` (deepmd.model.linear.LinearModel method), 457  
`get_rcut()` (deepmd.model.model.Model method), 463  
`get_rcut()` (deepmd.model.model.StandardModel method), 465  
`get_rcut()` (deepmd.model.multi.MultiModel method), 468  
`get_rcut()` (deepmd.model.MultiModel method), 443  
`get_rcut()` (deepmd.model.pairtab.PairTabModel method), 471  
`get_rcut()` (deepmd.model.pairwise\_dprc.PairwiseDP method), 474  
`get_rcut()` (deepmd.model.tensor.TensorModel method), 479  
`get_reduce_name()` (in module deepmd\_utils.model\_format), 570  
`get_reduce_name()` (in module deepmd\_utils.model\_format.output\_def), 578  
`get_resource()` (in module deepmd.cluster), 225  
`get_resource()` (in module deepmd.cluster.local), 225  
`get_resource()` (in module deepmd.cluster.slurm), 226  
`get_rot_mat()` (deepmd.descriptor.DescriptLocFrame method), 242  
`get_rot_mat()` (deepmd.descriptor.DescriptSeA method), 247  
`get_rot_mat()` (deepmd.descriptor.DescriptSeAEf method), 256  
`get_rot_mat()` (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 294  
`get_rot_mat()` (deepmd.descriptor.se\_a.DescriptSeA method), 301  
`get_rot_mat()` (deepmd.descriptor.se\_a\_ef.DescriptSeAEf method), 310  
`get_s_range()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`get_sel_type()` (deepmd.fit.dipole.DipoleFittingSeA method), 359  
`get_sel_type()` (deepmd.fit.DipoleFittingSeA method), 347  
`get_sel_type()` (deepmd.fit.GlobalPolarFittingSeA method), 355  
`get_sel_type()` (deepmd.fit.polar.GlobalPolarFittingSeA method), 370  
`get_sel_type()` (deepmd.fit.polar.PolarFittingSeA method), 372  
`get_sel_type()` (deepmd.fit.PolarFittingSeA method), 357  
`get_sel_type()` (deepmd.infer.deep\_dos.DeepDOS method), 398  
`get_sel_type()` (deepmd.infer.deep\_pot.DeepPot method), 409  
`get_sel_type()` (deepmd.infer.deep\_tensor.DeepTensor method), 412  
`get_sel_type()` (deepmd.infer.DeepDOS method), 375  
`get_sel_type()` (deepmd.infer.DeepPot method), 386  
`get_sel_type()` (deepmd.model.tensor.TensorModel method), 479  
`get_sess()` (in module deepmd.nvnmd.utils.network), 504  
`get_spin_norm()` (deepmd.utils.spin.Spin method), 548  
`get_stat()` (deepmd.utils.neighbor\_stat.NeighborStat method), 537  
`get_sys()` (deepmd.utils.data\_system.DeepmdDataSystem



method), 529

get\_sys() (deepmd.utils.DeepmdDataSystem method), 515

get\_sys() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 593

get\_sys\_n\_test() (deepmd.utils.data\_system.DeepmdDataSystem method), 529

get\_sys\_n\_test() (deepmd.utils.DeepmdDataSystem method), 515

get\_sys\_n\_test() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 593

get\_sys\_num\_batch() (deepmd.utils.data.DeepmdData method), 526

get\_sys\_num\_batch() (deepmd.utils.DeepmdData method), 512

get\_sys\_num\_batch() (deepmd\_utils.utils.data.DeepmdData method), 589

get\_tensor\_by\_name() (in module deepmd.utils.graph), 533

get\_tensor\_by\_name\_from\_graph() (in module deepmd.utils.graph), 534

get\_tensor\_by\_type() (in module deepmd.utils.graph), 534

get\_tensor\_names() (deepmd.descriptor.Descriptor method), 232

get\_tensor\_names() (deepmd.descriptor.descriptor.Descriptor method), 440

get\_tensor\_names() (deepmd.descriptor.descriptor.Descriptor method), 284

get\_tensor\_names() (deepmd.descriptor.DescriptHybrid method), 237

get\_tensor\_names() (deepmd.descriptor.hybrid.DescriptHybrid method), 289

get\_tensor\_names() (deepmd.descriptor.se.DescriptSe method), 296

get\_test() (deepmd.utils.data.DeepmdData method), 526

get\_test() (deepmd.utils.data\_system.DeepmdDataSystem method), 529

get\_test() (deepmd.utils.DeepmdData method), 512

get\_test() (deepmd.utils.DeepmdDataSystem method), 515

get\_test() (deepmd\_utils.utils.data.DeepmdData method), 589

get\_test() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 593

get\_tf\_requirement() (in module back-end.find\_tensorflow), 217

get\_tf\_version() (in module back-end.find\_tensorflow), 218

get\_training\_jdata() (deepmd.nvnmd.utils.config.NvnmdConfig method), 499

get\_two\_side\_type\_embedding() (in module deepmd.utils.compress), 521

get\_type\_embedding() (in module deepmd.utils.compress), 521

get\_type\_embedding\_net\_nodes\_from\_graph\_def() (in module deepmd.utils.graph), 534

get\_type\_embedding\_net\_variables\_from\_graph\_def() (in module deepmd.utils.graph), 534

get\_type\_embedding\_weight() (in module deepmd.nvnmd.utils.weight), 506

get\_type\_map() (deepmd.infer.deep\_dos.DeepDOS method), 398

get\_type\_map() (deepmd.infer.deep\_pot.DeepPot method), 409

get\_type\_map() (deepmd.infer.deep\_tensor.DeepTensor method), 412

get\_type\_map() (deepmd.infer.DeepDOS method), 375

get\_type\_map() (deepmd.infer.DeepPot method), 386

get\_type\_map() (deepmd.model.dos.DOSModel method), 448

get\_type\_map() (deepmd.model.DOSModel method), 436

get\_type\_map() (deepmd.model.ener.EnerModel method), 451

get\_type\_map() (deepmd.model.EnerModel method), 454

get\_type\_map() (deepmd.model.frozen.FrozenModel method), 454

get\_type\_map() (deepmd.model.linear.LinearModel method), 457

get\_type\_map() (deepmd.model.model.Model method), 463

get\_type\_map() (deepmd.model.multi.MultiModel method), 468

get\_type\_map() (deepmd.model.MultiModel method), 444

get\_type\_map() (deepmd.model.tensor.TensorModel method), 479

get\_type\_map() (deepmd.utils.data.DeepmdData method), 526

get\_type\_map() (deepmd.utils.data\_system.DeepmdDataSystem method), 529

get\_type\_map() (deepmd.utils.DeepmdData method), 513

get\_type\_map() (deepmd.utils.DeepmdDataSystem method), 516

get\_type\_map() (deepmd\_utils.utils.data.DeepmdData method), 590

get\_type\_map() (deepmd\_utils.utils.data\_system.DeepmdDataSystem method), 593

get\_type\_weight() (in module deepmd.nvnmd.utils.weight), 506

`get_use_spin()` (deepmd.utils.spin.Spin method), 548  
`get_variables_from_graph_def_as_numpy_array()` (in module deepmd.utils.graph), 535  
`get_virtual_len()` (deepmd.utils.spin.Spin method), 548  
`get_weight()` (in module deepmd.nvnmd.utils.weight), 506  
`glob()` (deepmd.utils.path.DPH5Path method), 541  
`glob()` (deepmd.utils.path.DPOSPath method), 543  
`glob()` (deepmd.utils.path.DPPPath method), 544  
`glob()` (deepmd\_utils.utils.path.DPH5Path method), 595  
`glob()` (deepmd\_utils.utils.path.DPOSPath method), 596  
`glob()` (deepmd\_utils.utils.path.DPPPath method), 597  
`global_cvt_2_ener_float()` (in module deepmd.env), 559  
`global_cvt_2_tf_float()` (in module deepmd.env), 559  
`GLOBAL_ENER_FLOAT_PRECISION` (in module deepmd.env), 559  
`GLOBAL_ENER_FLOAT_PRECISION` (in module deepmd\_utils.env), 603  
`GLOBAL_NP_FLOAT_PRECISION` (in module deepmd.env), 559  
`GLOBAL_NP_FLOAT_PRECISION` (in module deepmd\_utils.env), 603  
`GlobalPolarFittingSeA` (class in deepmd.fit), 353  
`GlobalPolarFittingSeA` (class in deepmd.fit.polar), 368  
`GlobalPolarModel` (class in deepmd.model), 440  
`GlobalPolarModel` (class in deepmd.model.tensor), 475  
`GPU_MAX_NBOR_SIZE` (C macro), 845, 846  
`gpuDeviceSynchronize` (C macro), 846  
`gpuGetLastError` (C macro), 846, 847  
`gpuMemcpy` (C macro), 847  
`gpuMemcpyDeviceToDevice` (C macro), 847  
`gpuMemcpyDeviceToHost` (C macro), 848  
`gpuMemcpyHostToDevice` (C macro), 848  
`gpuMemset` (C macro), 849  
`gpus` (deepmd.train.run\_options.RunOptions attribute), 508  
`GraphTooLargeError`, 529  
`GraphWithoutTensorError`, 529  
**H**  
`hex2bin()` (deepmd.nvnmd.utils.Encode method), 494  
`hex2bin()` (deepmd.nvnmd.utils.encode.Encode method), 501  
`hex2bin_str()` (deepmd.nvnmd.utils.Encode method), 494  
`hex2bin_str()` (deepmd.nvnmd.utils.encode.Encode method), 501  
`implemented_properties` (deepmd.calculator.DP attribute), 554  
`import_ops()` (in module deepmd.op), 507  
`init_config_by_version()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_ctrl()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_dpfn()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_dscpr()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_fitn()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_from_config()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_from_deepmd_input()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_from_jdata()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_nbit()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_net_size()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_size()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_train_mode()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_value()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`init_variables()` (deepmd.descriptor.Descriptor method), 232  
`init_variables()` (deepmd.descriptor.descriptor.Descriptor method), 284  
`init_variables()` (deepmd.descriptor.DescriptHybrid method), 237  
`init_variables()` (deepmd.descriptor.DescriptLocFrame method), 242  
`init_variables()` (deepmd.descriptor.DescriptSeA method), 247  
`init_variables()` (deepmd.descriptor.DescriptSeAtten method), 267  
`init_variables()` (deepmd.descriptor.hybrid.DescriptHybrid method), 289  
`init_variables()` (deepmd.descriptor.loc\_frame.DescriptLocFrame method), 294

[init\\_variables\(\)](#) (deepmd.descriptor.se.DescriptSe method), 296  
[init\\_variables\(\)](#) (deepmd.descriptor.se\_a.DescriptSe method), 301  
[init\\_variables\(\)](#) (deepmd.descriptor.se\_atten.DescriptSe method), 321  
[init\\_variables\(\)](#) (deepmd.fit.dipole.DipoleFittingSe method), 359  
[init\\_variables\(\)](#) (deepmd.fit.DipoleFittingSe method), 347  
[init\\_variables\(\)](#) (deepmd.fit.dos.DOSFitting method), 362  
[init\\_variables\(\)](#) (deepmd.fit.DOSFitting method), 345  
[init\\_variables\(\)](#) (deepmd.fit.ener.EnerFitting method), 366  
[init\\_variables\(\)](#) (deepmd.fit.EnerFitting method), 351  
[init\\_variables\(\)](#) (deepmd.fit.Fitting method), 352  
[init\\_variables\(\)](#) (deepmd.fit.fitting.Fitting method), 367  
[init\\_variables\(\)](#) (deepmd.fit.GlobalPolarFittingSe method), 355  
[init\\_variables\(\)](#) (deepmd.fit.polar.GlobalPolarFittingSe method), 370  
[init\\_variables\(\)](#) (deepmd.fit.polar.PolarFittingSe method), 372  
[init\\_variables\(\)](#) (deepmd.fit.PolarFittingSe method), 357  
[init\\_variables\(\)](#) (deepmd.model.dos.DOSModel method), 448  
[init\\_variables\(\)](#) (deepmd.model.DOSModel method), 436  
[init\\_variables\(\)](#) (deepmd.model.ener.EnerModel method), 451  
[init\\_variables\(\)](#) (deepmd.model.EnerModel method), 440  
[init\\_variables\(\)](#) (deepmd.model.frozen.FrozenModel method), 454  
[init\\_variables\(\)](#) (deepmd.model.linear.LinearModel method), 457  
[init\\_variables\(\)](#) (deepmd.model.model.Model method), 463  
[init\\_variables\(\)](#) (deepmd.model.multi.MultiModel method), 468  
[init\\_variables\(\)](#) (deepmd.model.MultiModel method), 444  
[init\\_variables\(\)](#) (deepmd.model.pairtab.PairTabModel method), 471  
[init\\_variables\(\)](#) (deepmd.model.pairwise\_dprc.PairwiseDPRC method), 474  
[init\\_variables\(\)](#) (deepmd.model.tensor.TensorModel method), 479  
[init\\_variables\(\)](#) (deepmd.utils.type\_embed.TypeEmbedNet method), 551  
[int\\_64](#) (C++ type), 851  
[is\\_chief](#) (deepmd.train.run\_options.RunOptions property), 508  
[is\\_atten](#) (deepmd.utils.path.DPH5Path method), 542  
[is\\_dir\(\)](#) (deepmd.utils.path.DPOSPath method), 543  
[is\\_dir\(\)](#) (deepmd.utils.path.DPPath method), 544  
[is\\_dir\(\)](#) (deepmd\_utils.utils.path.DPH5Path method), 595  
[is\\_dir\(\)](#) (deepmd\_utils.utils.path.DPOSPath method), 597  
[is\\_dir\(\)](#) (deepmd\_utils.utils.path.DPPath method), 598  
[is\\_file\(\)](#) (deepmd.nvnmd.utils.fio.Fio method), 502  
[is\\_file\(\)](#) (deepmd.utils.path.DPH5Path method), 542  
[is\\_file\(\)](#) (deepmd.utils.path.DPOSPath method), 543  
[is\\_file\(\)](#) (deepmd.utils.path.DPPath method), 544  
[is\\_file\(\)](#) (deepmd\_utils.utils.path.DPH5Path method), 595  
[is\\_file\(\)](#) (deepmd\_utils.utils.path.DPOSPath method), 597  
[is\\_file\(\)](#) (deepmd\_utils.utils.path.DPPath method), 598  
[is\\_gpu\\_available\(\)](#) (deepmd.utils.batch\_size.AutoBatchSize method), 519  
[is\\_gpu\\_available\(\)](#) (deepmd\_utils.utils.batch\_size.AutoBatchSize method), 585  
[is\\_oom\\_error\(\)](#) (deepmd.utils.batch\_size.AutoBatchSize method), 519  
[is\\_oom\\_error\(\)](#) (deepmd\_utils.utils.batch\_size.AutoBatchSize method), 585  
[is\\_path\(\)](#) (deepmd.nvnmd.utils.fio.Fio method), 502  

## J

[j\\_loader\(\)](#) (in module deepmd.common), 557  
[j\\_loader\(\)](#) (in module deepmd\_utils.common), 602  
[j\\_must\\_have\(\)](#) (in module deepmd.common), 557  
[j\\_must\\_have\(\)](#) (in module deepmd\_utils.common), 602  

## K

[keys\(\)](#) (deepmd\_utils.model\_format.FittingOutputDef method), 566  
[keys\(\)](#) (deepmd\_utils.model\_format.ModelOutputDef method), 566  
[keys\(\)](#) (deepmd\_utils.model\_format.output\_def.FittingOutputDef method), 576

```

keys() (deepmd_utils.model_format.output_def.ModelOutputDef
      method), 577
keys_derv_c() (deepmd_utils.model_format.ModelOutputDef
      method), 566
keys_derv_c() (deepmd_utils.model_format.output_def.ModelOutputDef
      method), 577
keys_derv_r() (deepmd_utils.model_format.ModelOutputDef
      method), 567
keys_derv_r() (deepmd_utils.model_format.output_def.ModelOutputDef
      method), 577
keys_outp() (deepmd_utils.model_format.ModelOutputDef
      method), 567
keys_outp() (deepmd_utils.model_format.output_def.ModelOutputDef
      method), 577
keys_redu() (deepmd_utils.model_format.ModelOutputDef
      method), 567
keys_redu() (deepmd_utils.model_format.output_def.ModelOutputDef
      method), 577

L
layer_name:
    model[standard]/fitting_net[ener]/layer_name
    (Argument), 113
learning_rate (Argument)
    learning_rate:, 118
learning_rate/scale_by_worker (Argument)
    scale_by_worker:, 118
learning_rate/type (Argument)
    type:, 118
learning_rate:
    learning_rate (Argument), 118
learning_rate_args() (in module
    deepmd_utils.utils.argcheck), 582
learning_rate_dict (Argument)
    learning_rate_dict:, 119
learning_rate_dict:
    learning_rate_dict (Argument), 119
learning_rate_dict_args() (in module
    deepmd_utils.utils.argcheck), 582
learning_rate_exp() (in module
    deepmd_utils.utils.argcheck), 582
learning_rate_variant_type_args() (in module
    deepmd_utils.utils.argcheck), 582
learning_rate[exp]/decay_steps (Argument)
    decay_steps:, 119
learning_rate[exp]/start_lr (Argument)
    start_lr:, 118
learning_rate[exp]/stop_lr (Argument)
    stop_lr:, 118
LearningRateExp (class in deepmd.utils), 516
LearningRateExp (class in
    deepmd_utils.learning_rate), 535
limit_pref() (in module
    deepmd_utils.utils.argcheck), 582
limit_pref_acdf:
    loss[dos]/limit_pref_acdf (Argument), 124
limit_pref_adof:
    loss[dos]/limit_pref_adof (Argument), 124
limit_pref_ae:
    loss[ener_spin]/limit_pref_ae (Argument),
    122
    loss[ener]/limit_pref_ae (Argument), 120
limit_pref_cdf:
    loss[dos]/limit_pref_cdf (Argument), 124
limit_pref_dos:
    loss[dos]/limit_pref_dos (Argument), 123
limit_pref_e:
    loss[ener_spin]/limit_pref_e (Argument),
    121
    loss[ener]/limit_pref_e (Argument), 119
limit_pref_f:
    loss[ener]/limit_pref_f (Argument), 119
limit_pref_fm:
    loss[ener_spin]/limit_pref_fm (Argument),
    122
limit_pref_fr:
    loss[ener_spin]/limit_pref_fr (Argument),
    122
limit_pref_gf:
    loss[ener]/limit_pref_gf (Argument), 121
limit_pref_pf:
    loss[ener_spin]/limit_pref_pf (Argument),
    123
    loss[ener]/limit_pref_pf (Argument), 120
limit_pref_v:
    loss[ener_spin]/limit_pref_v (Argument),
    122
    loss[ener]/limit_pref_v (Argument), 120
linear_ener_model_args() (in module
    deepmd_utils.utils.argcheck), 582
LinearEnergyModel (class in deepmd.model.linear),
    455
LinearModel (class in deepmd.model.linear), 456
list:
    model[standard]/descriptor[hybrid]/list
    (Argument), 102
list_to_doc() (in module deepmd_utils.argcheck),
    519
list_to_doc() (in module
    deepmd_utils.utils.argcheck), 582
load() (deepmd.nvnmd.utils.fio.FioBin method), 503
load() (deepmd.nvnmd.utils.fio.FioDic method), 503
load() (deepmd.nvnmd.utils.fio.FioJsonDic
    method), 504
load() (deepmd.nvnmd.utils.fio.FioNpyDic
    method), 504
load() (deepmd.nvnmd.utils.fio.FioTxt method),
    504

```

`load()` (deepmd.nvnmd.utils.FioBin method), 495  
`load()` (deepmd.nvnmd.utils.FioDic method), 495  
`load()` (deepmd.nvnmd.utils.FioTxt method), 496  
`load_dp_model()` (in module deepmd\_utils.model\_format), 570  
`load_dp_model()` (in module deepmd\_utils.model\_format.network), 575  
`load_graph_def()` (in module deepmd\_utils.graph), 535  
`load_numpy()` (deepmd\_utils.path.DPH5Path method), 542  
`load_numpy()` (deepmd\_utils.path.DPOSPath method), 543  
`load_numpy()` (deepmd\_utils.path.DPPPath method), 544  
`load_numpy()` (deepmd\_utils.path.DPH5Path method), 596  
`load_numpy()` (deepmd\_utils.path.DPOSPath method), 597  
`load_numpy()` (deepmd\_utils.path.DPPPath method), 598  
`load_prefix` (deepmd.DeepEval attribute), 221  
`load_prefix` (deepmd.infer.deep\_dos.DeepDOS attribute), 398  
`load_prefix` (deepmd.infer.deep\_eval.DeepEval attribute), 400  
`load_prefix` (deepmd.infer.deep\_pot.DeepPot attribute), 409  
`load_prefix` (deepmd.infer.deep\_wfc.DeepWFC attribute), 413  
`load_prefix` (deepmd.infer.DeepDOS attribute), 375  
`load_prefix` (deepmd.infer.DeepEval attribute), 378  
`load_prefix` (deepmd.infer.DeepPot attribute), 386  
`load_txt()` (deepmd\_utils.path.DPH5Path method), 542  
`load_txt()` (deepmd\_utils.path.DPOSPath method), 543  
`load_txt()` (deepmd\_utils.path.DPPPath method), 544  
`load_txt()` (deepmd\_utils.path.DPH5Path method), 596  
`load_txt()` (deepmd\_utils.path.DPOSPath method), 597  
`load_txt()` (deepmd\_utils.path.DPPPath method), 598  
`loss` (Argument)  
    `loss:`, 119  
`Loss` (class in deepmd.loss.loss), 432  
`loss/type` (Argument)  
    `type:`, 119  
`loss:`  
    `loss` (Argument), 119  
`loss_args()` (in module deepmd\_utils.utils.argcheck), 582  
`loss_dict` (Argument)  
    `loss_dict:`, 125  
`loss_dict:`  
    `loss_dict` (Argument), 125  
`loss_dict_args()` (in module deepmd\_utils.utils.argcheck), 582  
`loss_dos()` (in module deepmd\_utils.utils.argcheck), 582  
`loss_ener()` (in module deepmd\_utils.utils.argcheck), 582  
`loss_ener_spin()` (in module deepmd\_utils.utils.argcheck), 582  
`loss_tensor()` (in module deepmd\_utils.utils.argcheck), 582  
`loss_variant_type_args()` (in module deepmd\_utils.utils.argcheck), 582  
`loss[dos]/limit_pref_acdf` (Argument)  
    `limit_pref_acdf:`, 124  
`loss[dos]/limit_pref_ados` (Argument)  
    `limit_pref_ados:`, 124  
`loss[dos]/limit_pref_cdf` (Argument)  
    `limit_pref_cdf:`, 124  
`loss[dos]/limit_pref_dos` (Argument)  
    `limit_pref_dos:`, 123  
`loss[dos]/start_pref_acdf` (Argument)  
    `start_pref_acdf:`, 124  
`loss[dos]/start_pref_ados` (Argument)  
    `start_pref_ados:`, 124  
`loss[dos]/start_pref_cdf` (Argument)  
    `start_pref_cdf:`, 123  
`loss[dos]/start_pref_dos` (Argument)  
    `start_pref_dos:`, 123  
`loss[ener_spin]/enable_atom_ener_coeff` (Argument)  
    `enable_atom_ener_coeff:`, 123  
`loss[ener_spin]/limit_pref_ae` (Argument)  
    `limit_pref_ae:`, 122  
`loss[ener_spin]/limit_pref_e` (Argument)  
    `limit_pref_e:`, 121  
`loss[ener_spin]/limit_pref_fm` (Argument)  
    `limit_pref_fm:`, 122  
`loss[ener_spin]/limit_pref_fr` (Argument)  
    `limit_pref_fr:`, 122  
`loss[ener_spin]/limit_pref_pf` (Argument)  
    `limit_pref_pf:`, 123  
`loss[ener_spin]/limit_pref_v` (Argument)  
    `limit_pref_v:`, 122  
`loss[ener_spin]/relative_f` (Argument)  
    `relative_f:`, 123  
`loss[ener_spin]/start_pref_ae` (Argument)  
    `start_pref_ae:`, 122  
`loss[ener_spin]/start_pref_e` (Argument)  
    `start_pref_e:`, 121



`loss[ener_spin]/start_pref_fm (Argument)`  
`start_pref_fm:`, 122  
`loss[ener_spin]/start_pref_fr (Argument)`  
`start_pref_fr:`, 121  
`loss[ener_spin]/start_pref_pf (Argument)`  
`start_pref_pf:`, 123  
`loss[ener_spin]/start_pref_v (Argument)`  
`start_pref_v:`, 122  
`loss[ener]/enable_atom_ener_coeff (Argument)`  
`enable_atom_ener_coeff:`, 121  
`loss[ener]/limit_pref_ae (Argument)`  
`limit_pref_ae:`, 120  
`loss[ener]/limit_pref_e (Argument)`  
`limit_pref_e:`, 119  
`loss[ener]/limit_pref_f (Argument)`  
`limit_pref_f:`, 119  
`loss[ener]/limit_pref_gf (Argument)`  
`limit_pref_gf:`, 121  
`loss[ener]/limit_pref_pf (Argument)`  
`limit_pref_pf:`, 120  
`loss[ener]/limit_pref_v (Argument)`  
`limit_pref_v:`, 120  
`loss[ener]/numb_generalized_coord (Argument)`  
`numb_generalized_coord:`, 121  
`loss[ener]/relative_f (Argument)`  
`relative_f:`, 120  
`loss[ener]/start_pref_ae (Argument)`  
`start_pref_ae:`, 120  
`loss[ener]/start_pref_e (Argument)`  
`start_pref_e:`, 119  
`loss[ener]/start_pref_f (Argument)`  
`start_pref_f:`, 119  
`loss[ener]/start_pref_gf (Argument)`  
`start_pref_gf:`, 121  
`loss[ener]/start_pref_pf (Argument)`  
`start_pref_pf:`, 120  
`loss[ener]/start_pref_v (Argument)`  
`start_pref_v:`, 120  
`loss[tensor]/pref (Argument)`  
`pref:`, 124  
`loss[tensor]/pref_atomic (Argument)`  
`pref_atomic:`, 125

## M

`main() (in module deepmd.entrypoints.main)`, 339  
`main() (in module deepmd_utils.main)`, 604  
`main_parser() (in module deepmd.entrypoints.main)`, 339  
`main_parser() (in module deepmd_utils.main)`, 604  
`make_data() (in module deepmd_utils.compress)`, 521  
`make_default_mesh() (in module deepmd.common)`, 558  
`make_default_mesh() (in module deepmd_utils.common)`, 603  
`make_embedding_network() (in module deepmd_utils.model_format)`, 570  
`make_embedding_network() (in module deepmd_utils.model_format.network)`, 575  
`make_fitting_network() (in module deepmd_utils.model_format)`, 570  
`make_fitting_network() (in module deepmd_utils.model_format.network)`, 575  
`make_index() (in module deepmd_utils.utils.argcheck)`, 582  
`make_link() (in module deepmd_utils.utils.argcheck)`, 582  
`make_model_devi() (in module deepmd.entrypoints)`, 333  
`make_model_devi() (in module deepmd.infer.model_devi)`, 417  
`make_multilayer_network() (in module deepmd_utils.model_format)`, 570  
`make_multilayer_network() (in module deepmd_utils.model_format.network)`, 575  
`make_natoms_vec() (deepmd.DeepEval method)`, 221  
`make_natoms_vec() (deepmd.infer.deep_eval.DeepEval method)`, 400  
`make_natoms_vec() (deepmd.infer.DeepEval method)`, 378  
`make_stat_input() (in module deepmd.model.model_stat)`, 465  
`make_stat_input() (in module deepmd_utils.utils.model_stat)`, 594  
`map_aparam() (in module deepmd.env.op_module)`, 640  
`map_file:`  
`nvnm/ map_file (Argument)`, 130  
`mapflt_nvnm() (in module deepmd.env.op_module)`, 641  
`map_nvnm() (in module deepmd.nvnm.utils)`, 496  
`map_nvnm() (in module deepmd.nvnm.utils.op)`, 505  
`MapAparam() (in module deepmd.env.op_module)`, 615  
`MapFltNvnm() (in module deepmd.env.op_module)`, 615  
`mapping() (deepmd.nvnm.entrypoints.mapt.MapTable method)`, 490  
`mapping() (deepmd.nvnm.entrypoints.MapTable method)`, 485  
`mapping2() (deepmd.nvnm.entrypoints.mapt.MapTable method)`, 490  
`mapping2() (deepmd.nvnm.entrypoints.MapTable method)`, 485

`mapt()` (in module `deepmd.nvnmd.entrypoints.mapt`), 490  
`MapTable` (class in `deepmd.nvnmd.entrypoints`), 483  
`MapTable` (class in `deepmd.nvnmd.entrypoints.mapt`), 488  
`matmul2_qq()` (in module `deepmd.nvnmd.utils.network`), 504  
`matmul3_qq()` (in module `deepmd.nvnmd.utils.network`), 504  
`matmul_fitnet_nvnmd()` (in module `deepmd.env.op_module`), 641  
`matmul_flt2fix_nvnmd()` (in module `deepmd.env.op_module`), 641  
`matmul_flt_nvnmd()` (in module `deepmd.env.op_module`), 642  
`MatmulFitnetNvnmd()` (in module `deepmd.env.op_module`), 615  
`MatmulFlt2fixNvnmd()` (in module `deepmd.env.op_module`), 616  
`MatmulFltNvnmd()` (in module `deepmd.env.op_module`), 616  
`max_nnei:`  
`nvnmd/max_nnei` (Argument), 130  
`merge_bin()` (`deepmd.nvnmd.utils.Encode` method), 494  
`merge_bin()` (`deepmd.nvnmd.utils.encode.Encode` method), 501  
`merge_input_stats()`  
(`deepmd.descriptor.DescrptHybrid` method), 237  
`merge_input_stats()`  
(`deepmd.descriptor.DescrptSeA` method), 248  
`merge_input_stats()`  
(`deepmd.descriptor.DescrptSeR` method), 273  
`merge_input_stats()`  
(`deepmd.descriptor.DescrptSeT` method), 277  
`merge_input_stats()`  
(`deepmd.descriptor.hybrid.DescrptHybrid` method), 289  
`merge_input_stats()`  
(`deepmd.descriptor.se_a.DescrptSeA` method), 302  
`merge_input_stats()`  
(`deepmd.descriptor.se_r.DescrptSeR` method), 327  
`merge_input_stats()`  
(`deepmd.descriptor.se_t.DescrptSeT` method), 331  
`merge_sys_stat()` (in module `deepmd.model.model_stat`), 465  
`merge_sys_stat()` (in module `deepmd_utils.utils.model_stat`), 594  
`mixed_precision:`  
`training/mixed_precision` (Argument), 127  
`mixed_precision_args()` (in module `deepmd_utils.utils.argcheck`), 582  
`mkdir()` (`deepmd.nvnmd.utils.fio.Fio` method), 502  
`MOASPDIM` (C macro), 849  
`model` (Argument)  
`model:`, 89  
`Model` (class in `deepmd.model.model`), 458  
`model/compress` (Argument)  
`compress:`, 92  
`model/data_bias_nsample` (Argument)  
`data_bias_nsample:`, 89  
`model/data_stat_nbatch` (Argument)  
`data_stat_nbatch:`, 89  
`model/data_stat_protect` (Argument)  
`data_stat_protect:`, 89  
`model/modifier` (Argument)  
`modifier:`, 91  
`model/modifier/type` (Argument)  
`type:`, 91  
`model/modifier[dipole_charge]/ewald_beta`  
(Argument)  
`ewald_beta:`, 91  
`model/modifier[dipole_charge]/ewald_h`  
(Argument)  
`ewald_h:`, 92  
`model/modifier[dipole_charge]/model_charge_map`  
(Argument)  
`model_charge_map:`, 91  
`model/modifier[dipole_charge]/model_name`  
(Argument)  
`model_name:`, 91  
`model/modifier[dipole_charge]/sys_charge_map`  
(Argument)  
`sys_charge_map:`, 91  
`model/smin_alpha` (Argument)  
`smin_alpha:`, 89  
`model/spin` (Argument)  
`spin:`, 92  
`model/spin/spin_norm` (Argument)  
`spin_norm:`, 92  
`model/spin/use_spin` (Argument)  
`use_spin:`, 92  
`model/spin/virtual_len` (Argument)  
`virtual_len:`, 92  
`model/srtab_add_bias` (Argument)  
`srtab_add_bias:`, 90  
`model/sw_rmax` (Argument)  
`sw_rmax:`, 90  
`model/sw_rmin` (Argument)  
`sw_rmin:`, 90  
`model/type` (Argument)

```

    type:, 92
model/type_embedding (Argument)
    type_embedding:, 90
model/type_embedding/activation_function
    (Argument)
    activation_function:, 90
model/type_embedding/neuron (Argument)
    neuron:, 90
model/type_embedding/precision (Argument)
    precision:, 90
model/type_embedding/resnet_dt (Argument)
    resnet_dt:, 90
model/type_embedding/seed (Argument)
    seed:, 91
model/type_embedding/trainable (Argument)
    trainable:, 91
model/type_map (Argument)
    type_map:, 89
model/use_srtab (Argument)
    use_srtab:, 89
model:
    model (Argument), 89
model_args() (in module
    deepmd_utils.utils.argcheck), 582
model_charge_map:
    model/modifier[dipole_charge]/model_charge_mapmodels:, 118
    (Argument), 91
model_check_output() (in module
    deepmd_utils.model_format), 570
model_check_output() (in module
    deepmd_utils.model_format.output_def),
    578
model_compression() (in module
    deepmd_utils.utils.argcheck), 582
model_compression_type_args() (in module
    deepmd_utils.utils.argcheck), 582
model_file:
    model[frozen]/model_file (Argument), 117
model_name:
    model/modifier[dipole_charge]/model_name
    (Argument), 91
model_type (deepmd.DeepEval property), 221
model_type (deepmd.infer.deep_eval.DeepEval
    property), 400
model_type (deepmd.infer.DeepEval property), 379
model_type (deepmd.model.dos.DOSModel at-
    tribute), 448
model_type (deepmd.model.DOSModel attribute),
    436
model_type (deepmd.model.ener.EnerModel at-
    tribute), 452
model_type (deepmd.model.EnerModel attribute),
    440
model_type (deepmd.model.linear.LinearEnergyModel
    attribute), 456
model_type (deepmd.model.multi.MultiModel
    attribute), 468
model_type (deepmd.model.MultiModel attribute),
    444
model_type (deepmd.model.pairtab.PairTabModel
    attribute), 471
model_type (deepmd.model.pairwise_dprc.PairwiseDPRc
    attribute), 474
model_version (deepmd.DeepEval property), 221
model_version (deepmd.infer.deep_eval.DeepEval
    property), 401
model_version (deepmd.infer.DeepEval property),
    379
ModelOutputDef (class in
    deepmd_utils.model_format), 566
ModelOutputDef (class in
    deepmd_utils.model_format.output_def),
    576
models:
    model[linear_ener]/models (Argument), 118
model[frozen]/model_file (Argument)
    model_file:, 117
model[linear_ener]/models (Argument)
models:, 118
model[linear_ener]/weights (Argument)
    weights:, 118
model[multi]/descriptor (Argument)
    descriptor:, 116
model[multi]/fitting_net_dict (Argument)
    fitting_net_dict:, 117
model[pairtab]/rcut (Argument)
    rcut:, 117
model[pairtab]/sel (Argument)
    sel:, 117
model[pairtab]/tab_file (Argument)
    tab_file:, 117
model[pairwise_dprc]/qm_model (Argument)
    qm_model:, 118
model[pairwise_dprc]/qmmm_model (Argument)
    qmmm_model:, 118
model[standard]/descriptor (Argument)
    descriptor:, 92
model[standard]/descriptor/type (Argument)
    type:, 92
model[standard]/descriptor[hybrid]/list
    (Argument)
    list:, 102
model[standard]/descriptor[loc_frame]/axis_rule
    (Argument)
    axis_rule:, 93
model[standard]/descriptor[loc_frame]/rcut
    (Argument)

```



```

    rcut:, 93
model[standard]/descriptor[loc_frame]/sel_a
    (Argument)
    sel_a:, 93
model[standard]/descriptor[loc_frame]/sel_r
    (Argument)
    sel_r:, 93
model[standard]/descriptor[se_a_ebd_v2]/activation_function
    (Argument)
    activation_function:, 108
model[standard]/descriptor[se_a_ebd_v2]/axis_neuron
    (Argument)
    axis_neuron:, 108
model[standard]/descriptor[se_a_ebd_v2]/exclude_types
    (Argument)
    exclude_types:, 109
model[standard]/descriptor[se_a_ebd_v2]/neuron
    (Argument)
    neuron:, 108
model[standard]/descriptor[se_a_ebd_v2]/precision
    (Argument)
    precision:, 108
model[standard]/descriptor[se_a_ebd_v2]/rcut
    (Argument)
    rcut:, 107
model[standard]/descriptor[se_a_ebd_v2]/rcut_smth
    (Argument)
    rcut_smth:, 107
model[standard]/descriptor[se_a_ebd_v2]/resnet_dt
    (Argument)
    resnet_dt:, 108
model[standard]/descriptor[se_a_ebd_v2]/seed
    (Argument)
    seed:, 109
model[standard]/descriptor[se_a_ebd_v2]/sel
    (Argument)
    sel:, 107
model[standard]/descriptor[se_a_ebd_v2]/set_davg_zero
    (Argument)
    set_davg_zero:, 109
model[standard]/descriptor[se_a_ebd_v2]/trainable
    (Argument)
    trainable:, 109
model[standard]/descriptor[se_a_ebd_v2]/type_one_side
    (Argument)
    type_one_side:, 108
model[standard]/descriptor[se_a_mask]/activation_function
    (Argument)
    activation_function:, 110
model[standard]/descriptor[se_a_mask]/axis_neuron
    (Argument)
    axis_neuron:, 110
model[standard]/descriptor[se_a_mask]/exclude_types
    (Argument)
    exclude_types:, 110
model[standard]/descriptor[se_a_mask]/neuron
    (Argument)
    neuron:, 109
model[standard]/descriptor[se_a_mask]/precision
    (Argument)
    precision:, 110
model[standard]/descriptor[se_a_mask]/resnet_dt
    (Argument)
    resnet_dt:, 110
model[standard]/descriptor[se_a_mask]/seed
    (Argument)
    seed:, 111
model[standard]/descriptor[se_a_mask]/sel
    (Argument)
    sel:, 109
model[standard]/descriptor[se_a_mask]/trainable
    (Argument)
    trainable:, 110
model[standard]/descriptor[se_a_mask]/type_one_side
    (Argument)
    type_one_side:, 110
model[standard]/descriptor[se_a_tpe]/activation_function
    (Argument)
    activation_function:, 98
model[standard]/descriptor[se_a_tpe]/axis_neuron
    (Argument)
    axis_neuron:, 98
model[standard]/descriptor[se_a_tpe]/exclude_types
    (Argument)
    exclude_types:, 99
model[standard]/descriptor[se_a_tpe]/neuron
    (Argument)
    neuron:, 98
model[standard]/descriptor[se_a_tpe]/numb_aparam
    (Argument)
    numb_aparam:, 100
model[standard]/descriptor[se_a_tpe]/precision
    (Argument)
    precision:, 99
model[standard]/descriptor[se_a_tpe]/rcut
    (Argument)
    rcut:, 98
model[standard]/descriptor[se_a_tpe]/rcut_smth
    (Argument)
    rcut_smth:, 98
model[standard]/descriptor[se_a_tpe]/resnet_dt
    (Argument)
    resnet_dt:, 98
model[standard]/descriptor[se_a_tpe]/seed
    (Argument)
    seed:, 99
model[standard]/descriptor[se_a_tpe]/sel
    (Argument)
    sel:, 109

```

```

sel:, 97
model[standard]/descriptor[se_a_tpe]/set_davg_zero:
    (Argument)
set_davg_zero:, 99
model[standard]/descriptor[se_a_tpe]/trainable
    (Argument)
trainable:, 99
model[standard]/descriptor[se_a_tpe]/type_nchan
    (Argument)
type_nchanl:, 99
model[standard]/descriptor[se_a_tpe]/type_nlayer
    (Argument)
type_nlayer:, 99
model[standard]/descriptor[se_a_tpe]/type_one_side
    (Argument)
type_one_side:, 98
model[standard]/descriptor[se_atten_v2]/activation
    (Argument)
activation_function:, 105
model[standard]/descriptor[se_atten_v2]/attn
    (Argument)
attn:, 106
model[standard]/descriptor[se_atten_v2]/attn_dotr
    (Argument)
attn_dotr:, 107
model[standard]/descriptor[se_atten_v2]/attn_layer
    (Argument)
attn_layer:, 106
model[standard]/descriptor[se_atten_v2]/attn_mask
    (Argument)
attn_mask:, 107
model[standard]/descriptor[se_atten_v2]/axis_neuron
    (Argument)
axis_neuron:, 105
model[standard]/descriptor[se_atten_v2]/exclude_types
    (Argument)
exclude_types:, 106
model[standard]/descriptor[se_atten_v2]/neuron
    (Argument)
neuron:, 105
model[standard]/descriptor[se_atten_v2]/precision
    (Argument)
precision:, 106
model[standard]/descriptor[se_atten_v2]/rcut
    (Argument)
rcut:, 105
model[standard]/descriptor[se_atten_v2]/rcut_smth
    (Argument)
rcut_smth:, 105
model[standard]/descriptor[se_atten_v2]/resnet_dt
    (Argument)
resnet_dt:, 106
model[standard]/descriptor[se_atten_v2]/seed
    (Argument)

seed:, 106
model[standard]/descriptor[se_atten_v2]/sel
    (Argument)
sel:, 104
model[standard]/descriptor[se_atten_v2]/set_davg_zero
    (Argument)
set_davg_zero:, 107
model[standard]/descriptor[se_atten_v2]/trainable
    (Argument)
trainable:, 106
model[standard]/descriptor[se_atten_v2]/type_one_side
    (Argument)
type_one_side:, 106
model[standard]/descriptor[se_atten]/activation_function
    (Argument)
activation_function:, 103
model[standard]/descriptor[se_atten]/attn
    (Argument)
attn:, 103
model[standard]/descriptor[se_atten]/attn_dotr
    (Argument)
attn_dotr:, 104
model[standard]/descriptor[se_atten]/attn_layer
    (Argument)
attn_layer:, 104
model[standard]/descriptor[se_atten]/attn_mask
    (Argument)
attn_mask:, 104
model[standard]/descriptor[se_atten]/axis_neuron
    (Argument)
axis_neuron:, 102
model[standard]/descriptor[se_atten]/exclude_types
    (Argument)
exclude_types:, 103
model[standard]/descriptor[se_atten]/neuron
    (Argument)
neuron:, 102
model[standard]/descriptor[se_atten]/precision
    (Argument)
precision:, 103
model[standard]/descriptor[se_atten]/rcut
    (Argument)
rcut:, 102
model[standard]/descriptor[se_atten]/rcut_smth
    (Argument)
rcut_smth:, 102
model[standard]/descriptor[se_atten]/resnet_dt
    (Argument)
resnet_dt:, 103
model[standard]/descriptor[se_atten]/seed
    (Argument)
seed:, 103
model[standard]/descriptor[se_atten]/sel
    (Argument)

```

```

sel:, 102
model[standard]/descriptor[se_atten]/set_davg_zero:
    (Argument)
set_davg_zero:, 104
model[standard]/descriptor[se_atten]/smooth_type_embedding:
    (Argument)
smooth_type_embedding:, 104
model[standard]/descriptor[se_atten]/stripped_type_embedding:
    (Argument)
stripped_type_embedding:, 104
model[standard]/descriptor[se_atten]/trainable:
    (Argument)
trainable:, 103
model[standard]/descriptor[se_atten]/type_one_side:
    (Argument)
type_one_side:, 103
model[standard]/descriptor[se_e2_a]/activation_function:
    (Argument)
activation_function:, 95
model[standard]/descriptor[se_e2_a]/axis_neuron:
    (Argument)
axis_neuron:, 95
model[standard]/descriptor[se_e2_a]/exclude_types:
    (Argument)
exclude_types:, 96
model[standard]/descriptor[se_e2_a]/neuron:
    (Argument)
neuron:, 94
model[standard]/descriptor[se_e2_a]/precision:
    (Argument)
precision:, 95
model[standard]/descriptor[se_e2_a]/rcut:
    (Argument)
rcut:, 94
model[standard]/descriptor[se_e2_a]/rcut_smth:
    (Argument)
rcut_smth:, 94
model[standard]/descriptor[se_e2_a]/resnet_dt:
    (Argument)
resnet_dt:, 95
model[standard]/descriptor[se_e2_a]/seed:
    (Argument)
seed:, 95
model[standard]/descriptor[se_e2_a]/sel:
    (Argument)
sel:, 94
model[standard]/descriptor[se_e2_a]/set_davg_zero:
    (Argument)
set_davg_zero:, 96
model[standard]/descriptor[se_e2_a]/trainable:
    (Argument)
trainable:, 95
model[standard]/descriptor[se_e2_a]/type_one_side:
    (Argument)
type_one_side:, 95
model[standard]/descriptor[se_e2_r]/activation_function:
    (Argument)
activation_function:, 100
model[standard]/descriptor[se_e2_r]/exclude_types:
    (Argument)
exclude_types:, 101
model[standard]/descriptor[se_e2_r]/neuron:
    (Argument)
neuron:, 100
model[standard]/descriptor[se_e2_r]/precision:
    (Argument)
precision:, 101
model[standard]/descriptor[se_e2_r]/rcut:
    (Argument)
rcut:, 100
model[standard]/descriptor[se_e2_r]/rcut_smth:
    (Argument)
rcut_smth:, 100
model[standard]/descriptor[se_e2_r]/resnet_dt:
    (Argument)
resnet_dt:, 101
model[standard]/descriptor[se_e2_r]/seed:
    (Argument)
seed:, 101
model[standard]/descriptor[se_e2_r]/sel:
    (Argument)
sel:, 100
model[standard]/descriptor[se_e2_r]/set_davg_zero:
    (Argument)
set_davg_zero:, 101
model[standard]/descriptor[se_e2_r]/trainable:
    (Argument)
trainable:, 101
model[standard]/descriptor[se_e2_r]/type_one_side:
    (Argument)
type_one_side:, 101
model[standard]/descriptor[se_e3]/activation_function:
    (Argument)
activation_function:, 97
model[standard]/descriptor[se_e3]/neuron:
    (Argument)
neuron:, 96
model[standard]/descriptor[se_e3]/precision:
    (Argument)
precision:, 97
model[standard]/descriptor[se_e3]/rcut:
    (Argument)
rcut:, 96
model[standard]/descriptor[se_e3]/rcut_smth:
    (Argument)
rcut_smth:, 96
model[standard]/descriptor[se_e3]/resnet_dt:
    (Argument)
resnet_dt:, 95

```

resnet_dt:, 97	(Argument)
model[standard]/descriptor[se_e3]/seed	rcond:, 114
(Argument)	model[standard]/fitting_net[dos]/resnet_dt
seed:, 97	(Argument)
model[standard]/descriptor[se_e3]/sel	resnet_dt:, 114
(Argument)	model[standard]/fitting_net[dos]/seed
sel:, 96	(Argument)
model[standard]/descriptor[se_e3]/set_davg_zero	seed:, 114
(Argument)	model[standard]/fitting_net[dos]/trainable
set_davg_zero:, 97	(Argument)
model[standard]/descriptor[se_e3]/trainable	trainable:, 114
(Argument)	model[standard]/fitting_net[ener]/activation_function
trainable:, 97	(Argument)
model[standard]/fitting_net (Argument)	activation_function:, 112
fitting_net:, 111	model[standard]/fitting_net[ener]/atom_ener
model[standard]/fitting_net/type (Argument)	(Argument)
type:, 111	atom_ener:, 113
model[standard]/fitting_net[dipole]/activation_function	model[standard]/fitting_net[ener]/layer_name
(Argument)	(Argument)
activation_function:, 116	layer_name:, 113
model[standard]/fitting_net[dipole]/neuron	model[standard]/fitting_net[ener]/neuron
(Argument)	(Argument)
neuron:, 116	neuron:, 112
model[standard]/fitting_net[dipole]/precision	model[standard]/fitting_net[ener]/numb_aparam
(Argument)	(Argument)
precision:, 116	numb_aparam:, 111
model[standard]/fitting_net[dipole]/resnet_dt	model[standard]/fitting_net[ener]/numb_fparam
(Argument)	(Argument)
resnet_dt:, 116	numb_fparam:, 111
model[standard]/fitting_net[dipole]/seed	model[standard]/fitting_net[ener]/precision
(Argument)	(Argument)
seed:, 116	precision:, 112
model[standard]/fitting_net[dipole]/sel_type	model[standard]/fitting_net[ener]/rcond
(Argument)	(Argument)
sel_type:, 116	rcond:, 112
model[standard]/fitting_net[dos]/activation_function	model[standard]/fitting_net[ener]/resnet_dt
(Argument)	(Argument)
activation_function:, 113	resnet_dt:, 112
model[standard]/fitting_net[dos]/neuron	model[standard]/fitting_net[ener]/seed
(Argument)	(Argument)
neuron:, 113	seed:, 112
model[standard]/fitting_net[dos]/numb_aparam	model[standard]/fitting_net[ener]/trainable
(Argument)	(Argument)
numb_aparam:, 113	trainable:, 112
model[standard]/fitting_net[dos]/numb_dos	model[standard]/fitting_net[ener]/use_aparam_as_mask
(Argument)	(Argument)
numb_dos:, 114	use_aparam_as_mask:, 113
model[standard]/fitting_net[dos]/numb_fparam	model[standard]/fitting_net[polar]/activation_function
(Argument)	(Argument)
numb_fparam:, 113	activation_function:, 115
model[standard]/fitting_net[dos]/precision	model[standard]/fitting_net[polar]/fit_diag
(Argument)	(Argument)
precision:, 114	fit_diag:, 115
model[standard]/fitting_net[dos]/rcond	model[standard]/fitting_net[polar]/neuron

```

    (Argument)
    neuron:, 114
model[standard]/fitting_net[polar]/precision
    (Argument)
    precision:, 115
model[standard]/fitting_net[polar]/resnet_dt
    (Argument)
    resnet_dt:, 115
model[standard]/fitting_net[polar]/scale
    (Argument)
    scale:, 115
model[standard]/fitting_net[polar]/seed
    (Argument)
    seed:, 115
model[standard]/fitting_net[polar]/sel_type
    (Argument)
    sel_type:, 115
model[standard]/fitting_net[polar]/shift_diag
    (Argument)
    shift_diag:, 115
modifier:
    model/modifier (Argument), 91
modifier_dipole_charge() (in module
    deepmd_utils.utils.argcheck), 583
modifier_variant_type_args() (in module
    deepmd_utils.utils.argcheck), 583
modify_data() (deepmd.DipoleChargeModifier
    method), 224
modify_data() (deepmd.infer.data_modifier.DipoleChargeModifier
    method), 393
modify_data() (deepmd.infer.DipoleChargeModifier
    method), 390
module
    backend, 217
    backend.dynamic_metadata, 217
    backend.find_tensorflow, 217
    backend.read_env, 218
    deepmd, 219
    deepmd.calculator, 552
    deepmd.cluster, 225
    deepmd.cluster.local, 225
    deepmd.cluster.slurm, 226
    deepmd.common, 554
    deepmd.descriptor, 226
    deepmd.descriptor.descriptor, 278
    deepmd.descriptor.hybrid, 285
    deepmd.descriptor.loc_frame, 291
    deepmd.descriptor.se, 295
    deepmd.descriptor.se_a, 297
    deepmd.descriptor.se_a_ebd, 303
    deepmd.descriptor.se_a_ebd_v2, 305
    deepmd.descriptor.se_a_ef, 306
    deepmd.descriptor.se_a_mask, 312
    deepmd.descriptor.se_atten, 316
    deepmd.descriptor.se_atten_v2, 321
    deepmd.descriptor.se_r, 323
    deepmd.descriptor.se_t, 328
    deepmd.entrypoints, 332
    deepmd.entrypoints.compress, 337
    deepmd.entrypoints.convert, 338
    deepmd.entrypoints.doc, 338
    deepmd.entrypoints.freeze, 338
    deepmd.entrypoints.gui, 338
    deepmd.entrypoints.ipi, 339
    deepmd.entrypoints.main, 339
    deepmd.entrypoints.neighbor_stat, 340
    deepmd.entrypoints.test, 340
    deepmd.entrypoints.train, 341
    deepmd.entrypoints.transfer, 342
    deepmd.env, 559
    deepmd.env.op_grads_module, 659
    deepmd.env.op_module, 607
    deepmd.fit, 343
    deepmd.fit.dipole, 357
    deepmd.fit.dos, 360
    deepmd.fit.ener, 363
    deepmd.fit.fitting, 366
    deepmd.fit.polar, 368
    deepmd.infer, 372
    deepmd.infer.data_modifier, 392
    deepmd.infer.deep_dipole, 394
    deepmd.infer.deep_dos, 395
    deepmd.infer.deep_eval, 398
    deepmd.infer.deep_polar, 402
    deepmd.infer.deep_pot, 405
    deepmd.infer.deep_tensor, 409
    deepmd.infer.deep_wfc, 412
    deepmd.infer.ewald_recip, 413
    deepmd.infer.model_devi, 414
    deepmd.lmp, 559
    deepmd.loggers, 418
    deepmd.loggers.loggers, 419
    deepmd.loss, 420
    deepmd.loss.dos, 426
    deepmd.loss.ener, 427
    deepmd.loss.loss, 432
    deepmd.loss.tensor, 433
    deepmd.model, 434
    deepmd.model.dos, 446
    deepmd.model.ener, 449
    deepmd.model.frozen, 452
    deepmd.model.linear, 455
    deepmd.model.model, 458
    deepmd.model.model_stat, 465
    deepmd.model.multi, 466
    deepmd.model.pairtab, 469
    deepmd.model.pairwise_dprc, 472
    deepmd.model.tensor, 475

```

deepmd.nvnmd, 480  
 deepmd.nvnmd.data, 480  
 deepmd.nvnmd.data.data, 481  
 deepmd.nvnmd.descriptor, 481  
 deepmd.nvnmd.descriptor.se\_a, 482  
 deepmd.nvnmd.descriptor.se\_atten, 482  
 deepmd.nvnmd.entrypoints, 483  
 deepmd.nvnmd.entrypoints.freeze, 487  
 deepmd.nvnmd.entrypoints.mapt, 488  
 deepmd.nvnmd.entrypoints.train, 490  
 deepmd.nvnmd.entrypoints.wrap, 491  
 deepmd.nvnmd.fit, 492  
 deepmd.nvnmd.fit.ener, 492  
 deepmd.nvnmd.utils, 493  
 deepmd.nvnmd.utils.argcheck, 497  
 deepmd.nvnmd.utils.config, 497  
 deepmd.nvnmd.utils.encode, 500  
 deepmd.nvnmd.utils.fio, 502  
 deepmd.nvnmd.utils.network, 504  
 deepmd.nvnmd.utils.op, 505  
 deepmd.nvnmd.utils.weight, 505  
 deepmd.op, 507  
 deepmd.train, 507  
 deepmd.train.run\_options, 507  
 deepmd.train.trainer, 508  
 deepmd.utils, 510  
 deepmd.utils.argcheck, 519  
 deepmd.utils.batch\_size, 519  
 deepmd.utils.compat, 520  
 deepmd.utils.compress, 521  
 deepmd.utils.convert, 521  
 deepmd.utils.data, 523  
 deepmd.utils.data\_system, 526  
 deepmd.utils.errors, 529  
 deepmd.utils.finetune, 530  
 deepmd.utils.graph, 530  
 deepmd.utils.learning\_rate, 535  
 deepmd.utils.multi\_init, 536  
 deepmd.utils.neighbor\_stat, 537  
 deepmd.utils.network, 538  
 deepmd.utils.pair\_tab, 539  
 deepmd.utils.parallel\_op, 540  
 deepmd.utils.path, 541  
 deepmd.utils.plugin, 545  
 deepmd.utils.random, 546  
 deepmd.utils.sess, 547  
 deepmd.utils.spin, 547  
 deepmd.utils.tabulate, 548  
 deepmd.utils.type\_embed, 550  
 deepmd.utils.weight\_avg, 552  
 deepmd\_utils, 560  
 deepmd\_utils.common, 601  
 deepmd\_utils.entrypoints, 560  
 deepmd\_utils.entrypoints.doc, 560  
 deepmd\_utils.entrypoints.gui, 560  
 deepmd\_utils.env, 603  
 deepmd\_utils.loggers, 561  
 deepmd\_utils.loggers.loggers, 562  
 deepmd\_utils.main, 604  
 deepmd\_utils.model\_format, 563  
 deepmd\_utils.model\_format.common, 571  
 deepmd\_utils.model\_format.env\_mat, 571  
 deepmd\_utils.model\_format.network, 572  
 deepmd\_utils.model\_format.output\_def, 576  
 deepmd\_utils.model\_format.se\_e2\_a, 578  
 deepmd\_utils.utils, 580  
 deepmd\_utils.utils.argcheck, 580  
 deepmd\_utils.utils.argcheck\_nvnmd, 583  
 deepmd\_utils.utils.batch\_size, 584  
 deepmd\_utils.utils.compat, 586  
 deepmd\_utils.utils.data, 587  
 deepmd\_utils.utils.data\_system, 590  
 deepmd\_utils.utils.errors, 593  
 deepmd\_utils.utils.model\_stat, 594  
 deepmd\_utils.utils.pair\_tab, 594  
 deepmd\_utils.utils.path, 595  
 deepmd\_utils.utils.plugin, 598  
 deepmd\_utils.utils.random, 600  
 deepmd\_utils.utils.weight\_avg, 601  
 mul\_flt\_nvnmd (C++ function), 843  
 mul\_flt\_nvnmd() (in module  
 deepmd.env.op\_module), 642  
 MulFltNvnmd() (in module deepmd.env.op\_module),  
 616  
 multi\_model\_args() (in module  
 deepmd\_utils.utils.argcheck), 583  
 MultiModel (class in deepmd.model), 441  
 MultiModel (class in deepmd.model.multi), 466  
 my\_device (deepmd.train.run\_options.RunOptions  
 attribute), 508  
 my\_rank (deepmd.train.run\_options.RunOptions at-  
 tribute), 508  
**N**  
 name (deepmd.calculator.DP attribute), 554  
 NativeLayer (class in deepmd\_utils.model\_format),  
 567  
 NativeLayer (class in  
 deepmd\_utils.model\_format.network),  
 573  
 NativeNet (in module deepmd\_utils.model\_format),  
 568  
 NativeNet (in module  
 deepmd\_utils.model\_format.network),  
 574  
 NativeOP (class in deepmd\_utils.model\_format), 568  
 NativeOP (class in deepmd\_utils.model\_format.common),  
 571



`natoms_match()` (deepmd.model.ener.EnerModel method), 452  
`natoms_match()` (deepmd.model.EnerModel method), 440  
`natoms_not_match()` (deepmd.model.ener.EnerModel method), 452  
`natoms_not_match()` (deepmd.model.EnerModel method), 440  
`NBIT_CUTF` (C macro), 849  
`NBIT_FLTF` (C macro), 849  
`nborAssert` (C++ function), 843, 844  
`nborErrcheck` (C macro), 850  
`neighbor_stat()` (in module deepmd.entypoints), 334  
`neighbor_stat()` (in module deepmd.entypoints.neighbor\_stat), 340  
`neighbor_stat()` (in module deepmd.env.op\_module), 642  
`NeighborStat` (class in deepmd.utils.neighbor\_stat), 537  
`NeighborStat()` (in module deepmd.env.op\_module), 616  
`net_size:`  
`nvnmd/net_size` (Argument), 130  
`NETWORK_TYPE_MAP` (deepmd\_utils.model\_format.network\_collection attribute), 575  
`NETWORK_TYPE_MAP` (deepmd\_utils.model\_format.NetworkCollection attribute), 569  
`NetworkCollection` (class in deepmd\_utils.model\_format), 568  
`NetworkCollection` (class in deepmd\_utils.model\_format.network), 574  
`neuron:`  
`model/type_embedding/neuron` (Argument), 90  
`model[standard]/descriptor[se_a_ebd_v2]/neuron` (Argument), 108  
`model[standard]/descriptor[se_a_mask]/neuron` (Argument), 109  
`model[standard]/descriptor[se_a_tpe]/neuron` (Argument), 98  
`model[standard]/descriptor[se_atten_v2]/neuron` (Argument), 105  
`model[standard]/descriptor[se_atten]/neuron` (Argument), 102  
`model[standard]/descriptor[se_e2_a]/neuron` (Argument), 94  
`model[standard]/descriptor[se_e2_r]/neuron` (Argument), 100  
`model[standard]/descriptor[se_e3]/neuron` (Argument), 96  
`model[standard]/fitting_net[dipole]/neuron` (Argument), 116  
`model[standard]/fitting_net[dos]/neuron` (Argument), 113  
`model[standard]/fitting_net[ener]/neuron` (Argument), 112  
`model[standard]/fitting_net[polar]/neuron` (Argument), 114  
`modelist` (deepmd.train.run\_options.RunOptions attribute), 508  
`nodename` (deepmd.train.run\_options.RunOptions attribute), 508  
`norm_expo()` (deepmd.nvnmd.utils.Encode method), 494  
`norm_expo()` (deepmd.nvnmd.utils.encode.Encode method), 501  
`normalize()` (in module deepmd\_utils.utils.argcheck), 519  
`normalize()` (in module deepmd\_utils.utils.argcheck), 583  
`normalize_data_dict()` (in module deepmd\_utils.utils.argcheck), 583  
`normalize_fitting_net_dict()` (in module deepmd\_utils.utils.argcheck), 583  
`normalize_fitting_weight()` (in module deepmd\_utils.utils.argcheck), 583  
`normalize_learning_rate_dict()` (in module deepmd\_utils.utils.argcheck), 583  
`normalize_learning_rate_dict_with_single_learning_rate()` (in module deepmd\_utils.utils.argcheck), 583  
`normalize_loss_dict()` (in module deepmd\_utils.utils.argcheck), 583  
`normalize_multi_task()` (in module deepmd\_utils.utils.argcheck), 583  
`normalized_input()` (in module deepmd.nvnmd.entypoints.train), 490  
`normalized_input_qnn()` (in module deepmd.nvnmd.entypoints.train), 490  
`numb_aparam:`  
`model[standard]/descriptor[se_a_tpe]/numb_aparam` (Argument), 100  
`model[standard]/fitting_net[dos]/numb_aparam` (Argument), 113  
`model[standard]/fitting_net[ener]/numb_aparam` (Argument), 111  
`numb_btch:`  
`training/validation_data/numb_btch` (Argument), 127  
`numb_dos:`  
`model[standard]/fitting_net[dos]/numb_dos` (Argument), 114  
`numb_fparam:`  
`model[standard]/fitting_net[dos]/numb_fparam` (Argument), 113  
`model[standard]/fitting_net[ener]/numb_fparam` (Argument), 111  
`numb_generalized_coord:`

loss[ener]/numb\_generalized\_coord  
 (Argument), 121  
 numb\_steps:  
 training/numb\_steps (Argument), 128  
 nvnmmd (Argument)  
 nvnmmd:, 129  
 nvnmmd/config\_file (Argument)  
 config\_file:, 130  
 nvnmmd/enable (Argument)  
 enable:, 130  
 nvnmmd/map\_file (Argument)  
 map\_file:, 130  
 nvnmmd/max\_nnei (Argument)  
 max\_nnei:, 130  
 nvnmmd/net\_size (Argument)  
 net\_size:, 130  
 nvnmmd/quantize\_descriptor (Argument)  
 quantize\_descriptor:, 131  
 nvnmmd/quantize\_fitting\_net (Argument)  
 quantize\_fitting\_net:, 131  
 nvnmmd/restore\_descriptor (Argument)  
 restore\_descriptor:, 130  
 nvnmmd/restore\_fitting\_net (Argument)  
 restore\_fitting\_net:, 130  
 nvnmmd/version (Argument)  
 version:, 130  
 nvnmmd/weight\_file (Argument)  
 weight\_file:, 130  
 nvnmmd:  
 nvnmmd (Argument), 129  
 nvnmmd\_args() (in module deepmd.nvnmmd.utils), 497  
 nvnmmd\_args() (in module  
 deepmd.nvnmmd.utils.argcheck), 497  
 nvnmmd\_args() (in module  
 deepmd\_utils.utils.argcheck\_nvnmmd),  
 583  
 NvnmmdConfig (class in deepmd.nvnmmd.utils.config),  
 497

## O

omp\_get\_num\_threads (C++ function), 844  
 omp\_get\_thread\_num (C++ function), 844  
 one\_layer() (in module deepmd.nvnmmd.utils), 497  
 one\_layer() (in module  
 deepmd.nvnmmd.utils.network), 505  
 one\_layer() (in module deepmd.utils.network), 539  
 one\_layer\_nvnmmd() (in module  
 deepmd.nvnmmd.fit.ener), 492  
 one\_layer\_rand\_seed\_shift() (in module  
 deepmd.utils.network), 539  
 one\_layer\_t() (in module  
 deepmd.nvnmmd.utils.network), 505  
 one\_layer\_wb() (in module  
 deepmd.nvnmmd.utils.network), 505

OutOfMemoryError, 529, 593  
 output\_prec:  
 training/mixed\_precision/output\_prec  
 (Argument), 127  
 OutputVariableDef (class in  
 deepmd\_utils.model\_format), 569  
 OutputVariableDef (class in  
 deepmd\_utils.model\_format.output\_def),  
 577

## P

pair\_tab() (in module deepmd.env.op\_module), 642  
 PairTab (class in deepmd.utils), 517  
 PairTab (class in deepmd.utils.pair\_tab), 539  
 PairTab (class in deepmd\_utils.utils.pair\_tab), 594  
 PairTab() (in module deepmd.env.op\_module), 617  
 pairtab\_model\_args() (in module  
 deepmd\_utils.utils.argcheck), 583  
 PairTabModel (class in deepmd.model.pairtab), 469  
 pairwise\_dprc() (in module  
 deepmd\_utils.utils.argcheck), 583  
 PairwiseDPRc (class in  
 deepmd.model.pairwise\_dprc), 472  
 parallel\_prod\_force\_se\_a() (in module  
 deepmd.env.op\_module), 643  
 ParallelOp (class in deepmd.utils.parallel\_op), 540  
 ParallelProdForceSeA() (in module  
 deepmd.env.op\_module), 617  
 parse\_args() (in module  
 deepmd.entrypoints.main), 339  
 parse\_args() (in module deepmd\_utils.main), 604  
 pass\_tensors\_from\_frz\_model()  
 (deepmd.descriptor.Descriptor method),  
 232  
 pass\_tensors\_from\_frz\_model()  
 (deepmd.descriptor.descriptor.Descriptor  
 method), 284  
 pass\_tensors\_from\_frz\_model()  
 (deepmd.descriptor.DescriptHybrid  
 method), 238  
 pass\_tensors\_from\_frz\_model()  
 (deepmd.descriptor.hybrid.DescriptHybrid  
 method), 290  
 pass\_tensors\_from\_frz\_model()  
 (deepmd.descriptor.se.DescriptSe method),  
 296  
 plot\_lines() (deepmd.nvnmmd.entrypoints.mapt.MapTable  
 method), 490  
 plot\_lines() (deepmd.nvnmmd.entrypoints.MapTable  
 method), 485  
 Plugin (class in deepmd.utils), 517  
 Plugin (class in deepmd.utils.plugin), 545  
 Plugin (class in deepmd\_utils.utils.plugin), 598  
 PluginVariant (class in deepmd.utils), 518



---

```

PluginVariant (class in deepmd.utils.plugin), 546
PluginVariant (class in deepmd_utils.utils.plugin), 599
PolarFittingSeA (class in deepmd.fit), 355
PolarFittingSeA (class in deepmd.fit.polar), 370
PolarModel (class in deepmd.model), 444
PolarModel (class in deepmd.model.tensor), 476
precision (deepmd.descriptor.se.DescriptSe property), 296
precision (deepmd.fit.Fitting property), 352
precision (deepmd.fit.fitting.Fitting property), 367
precision:
  model/type_embedding/precision (Argument), 90
  model[standard]/descriptor[se_a_ebd_v2]/precision deepmd_utils.utils.data_system), 593
  (Argument), 108
  model[standard]/descriptor[se_a_mask]/precision deepmd_utils.data_system), 529
  (Argument), 110
  model[standard]/descriptor[se_a_tpe]/precision deepmd_utils.utils.data_system), 593
  (Argument), 99
  model[standard]/descriptor[se_atten_v2]/precision deepmd.env.op_module), 643
  (Argument), 106
  model[standard]/descriptor[se_atten]/precision deepmd.env.op_module), 644
  (Argument), 103
  model[standard]/descriptor[se_e2_a]/precision deepmd.env.op_module), 646
  (Argument), 95
  model[standard]/descriptor[se_e2_r]/precision deepmd.env.op_module), 646
  (Argument), 101
  model[standard]/descriptor[se_e3]/precision deepmd.env.op_module), 646
  (Argument), 97
  model[standard]/fitting_net[dipole]/precision 647
  (Argument), 116
  model[standard]/fitting_net[dos]/precision deepmd.env.op_grads_module), 663
  (Argument), 114
  model[standard]/fitting_net[ener]/precision deepmd.env.op_module), 647
  (Argument), 112
  model[standard]/fitting_net[polar]/precision deepmd.env.op_module), 648
  (Argument), 115
pref:
  loss[tensor]/pref (Argument), 124
pref_atomic:
  loss[tensor]/pref_atomic (Argument), 125
print_header() (deepmd.loss.ener.EnerSpinLoss method), 429
print_header() (deepmd.loss.ener.EnerSpinLoss method), 423
print_header() (deepmd.train.trainer.DPTrainer static method), 509
print_on_training() (deepmd.loss.ener.EnerSpinLoss method), 429
print_on_training() (deepmd.loss.ener.EnerSpinLoss method), 423
print_on_training()
  (deepmd.train.trainer.DPTrainer static method), 509
print_resource_summary() (deepmd.train.run_options.RunOptions method), 508
print_summary() (deepmd.utils.data_system.DeepmdDataSystem method), 529
print_summary() (deepmd.utils.DeepmdDataSystem method), 516
print_summary() (deepmd_utils.utils.data_system.DeepmdDataSystem method), 593
prob_sys_size_ext() (in module deepmd.utils.data_system), 529
prob_sys_size_ext() (in module
process_sys_probs() (in module deepmd_utils.data_system), 529
process_sys_probs() (in module deepmd_utils.utils.data_system), 593
prod_env_mat_a() (in module
prod_env_mat_a_mix() (in module
prod_env_mat_a_mix_nvnmmd_quantize() (in module deepmd.env.op_module), 645
prod_env_mat_a_nvnmmd_quantize() (in module deepmd.env.op_module), 646
prod_env_mat_r() (in module
prod_force() (in module deepmd.env.op_module), 647
prod_force_grad() (in module deepmd.env.op_grads_module), 663
prod_force_norot() (in module
prod_force_se_a() (in module
prod_force_se_a_grad() (in module
prod_force_se_a_mask() (in module
prod_force_se_a_mask_grad() (in module
prod_force_se_r() (in module
prod_force_se_r_grad() (in module
prod_force_virial() (deepmd.descriptor.Descriptor method), 232
prod_force_virial() (deepmd.descriptor.descriptor.Descriptor method), 284
prod_force_virial()

```

---

(deepmd.descriptor.DescriptHybrid method), 238	deepmd.env.op_grads_module), 665
prod_force_virial() (deepmd.descriptor.DescriptLocFrame method), 242	ProdEnvMatA() (in module deepmd.env.op_module), 618
prod_force_virial() (deepmd.descriptor.DescriptSeA method), 248	ProdEnvMatAMix() (in module deepmd.env.op_module), 619
prod_force_virial() (deepmd.descriptor.DescriptSeAEf method), 256	ProdEnvMatAMixNvnmdQuantize() (in module deepmd.env.op_module), 619
prod_force_virial() (deepmd.descriptor.DescriptSeAMask method), 261	ProdEnvMatANvnmdQuantize() (in module deepmd.env.op_module), 620
prod_force_virial() (deepmd.descriptor.DescriptSeR method), 273	ProdEnvMatR() (in module deepmd.env.op_module), 621
prod_force_virial() (deepmd.descriptor.DescriptSeT method), 278	ProdForce() (in module deepmd.env.op_module), 621
prod_force_virial() (deepmd.descriptor.hybrid.DescriptHybrid method), 290	ProdForceGrad() (in module deepmd.env.op_grads_module), 660
prod_force_virial() (deepmd.descriptor.loc_frame.DescriptLocFrame method), 294	ProdForceNorot() (in module deepmd.env.op_module), 621
prod_force_virial() (deepmd.descriptor.se_a.DescriptSeA method), 302	ProdForceSeA() (in module deepmd.env.op_module), 622
prod_force_virial() (deepmd.descriptor.se_a_ef.DescriptSeAEf method), 310	ProdForceSeAGPUExecuteFuncutor (C++ struct), 799
prod_force_virial() (deepmd.descriptor.se_a_mask.DescriptSeAMask method), 315	ProdForceSeAGPUExecuteFuncutor::operator() (C++ function), 800
prod_force_virial() (deepmd.descriptor.se_r.DescriptSeR method), 327	ProdForceSeAGrad() (in module deepmd.env.op_grads_module), 660
prod_force_virial() (deepmd.descriptor.se_t.DescriptSeT method), 332	ProdForceSeAMask() (in module deepmd.env.op_module), 622
prod_virial() (in module deepmd.env.op_module), 649	ProdForceSeAMaskGrad() (in module deepmd.env.op_grads_module), 660
prod_virial_grad() (in module deepmd.env.op_grads_module), 665	ProdForceSeR() (in module deepmd.env.op_module), 622
prod_virial_norot() (in module deepmd.env.op_module), 649	ProdForceSeRGPUExecuteFuncutor (C++ struct), 800
prod_virial_se_a() (in module deepmd.env.op_module), 650	ProdForceSeRGPUExecuteFuncutor::operator() (C++ function), 800
prod_virial_se_a_grad() (in module deepmd.env.op_grads_module), 665	ProdForceSeRGrad() (in module deepmd.env.op_grads_module), 661
prod_virial_se_r() (in module deepmd.env.op_module), 650	ProdVirial() (in module deepmd.env.op_module), 623
prod_virial_se_r_grad() (in module deepmd.env.op_grads_module), 665	ProdVirialGrad() (in module deepmd.env.op_grads_module), 661
	ProdVirialNorot() (in module deepmd.env.op_module), 623
	ProdVirialSeA() (in module deepmd.env.op_module), 624
	ProdVirialSeAGPUExecuteFuncutor (C++ struct), 800
	ProdVirialSeAGPUExecuteFuncutor::operator() (C++ function), 800
	ProdVirialSeAGrad() (in module deepmd.env.op_grads_module), 662
	ProdVirialSeR() (in module deepmd.env.op_module), 624
	ProdVirialSeRGPUExecuteFuncutor (C++ struct), 801
	ProdVirialSeRGPUExecuteFuncutor::operator() (C++ function), 801

(C++ function), 801  
 ProdVirialSeRGrad() (in module  
 deepmd.env.op\_grads\_module), 662  
 profiling:  
 training/profiling (Argument), 129  
 profiling\_file:  
 training/profiling\_file (Argument), 129

## Q

qc() (deepmd.nvnmd.utils.Encode method), 494  
 qc() (deepmd.nvnmd.utils.encode.Encode method),  
 501  
 qf() (deepmd.nvnmd.utils.Encode method), 495  
 qf() (deepmd.nvnmd.utils.encode.Encode method),  
 501  
 qf() (in module deepmd.nvnmd.utils.network), 505  
 qm\_model:  
 model[pairwise\_dprc]/qm\_model (Argument),  
 118  
 qmmm\_model:  
 model[pairwise\_dprc]/qmmm\_model  
 (Argument), 118  
 qr() (deepmd.nvnmd.utils.Encode method), 495  
 qr() (deepmd.nvnmd.utils.encode.Encode method),  
 501  
 qr() (in module deepmd.nvnmd.utils.network), 505  
 quantize\_descriptor:  
 nvnm/quantize\_descriptor (Argument), 131  
 quantize\_fitting\_net:  
 nvnm/quantize\_fitting\_net (Argument), 131  
 quantize\_nvnm() (in module  
 deepmd.env.op\_module), 650  
 QuantizeNvnm() (in module  
 deepmd.env.op\_module), 624

## R

r2s() (in module deepmd.nvnmd.utils.op), 505  
 random() (in module deepmd.utils.random), 546  
 random() (in module deepmd\_utils.utils.random),  
 600  
 RawTextArgumentDefaultsHelpFormatter (class in  
 deepmd\_utils.main), 604  
 rcond:  
 model[standard]/fitting\_net[dos]/rcond  
 (Argument), 114  
 model[standard]/fitting\_net[ener]/rcond  
 (Argument), 112  
 rcut:  
 model[pairtab]/rcut (Argument), 117  
 model[standard]/descriptor[loc\_frame]/rcut  
 (Argument), 93  
 model[standard]/descriptor[se\_a\_ebd\_v2]/rcut  
 (Argument), 107

model[standard]/descriptor[se\_a\_tpe]/rcut  
 (Argument), 98  
 model[standard]/descriptor[se\_atten\_v2]/rcut  
 (Argument), 105  
 model[standard]/descriptor[se\_atten]/rcut  
 (Argument), 102  
 model[standard]/descriptor[se\_e2\_a]/rcut  
 (Argument), 94  
 model[standard]/descriptor[se\_e2\_r]/rcut  
 (Argument), 100  
 model[standard]/descriptor[se\_e3]/rcut  
 (Argument), 96  
 rcut\_smth:  
 model[standard]/descriptor[se\_a\_ebd\_v2]/rcut\_smth  
 (Argument), 107  
 model[standard]/descriptor[se\_a\_tpe]/rcut\_smth  
 (Argument), 98  
 model[standard]/descriptor[se\_atten\_v2]/rcut\_smth  
 (Argument), 105  
 model[standard]/descriptor[se\_atten]/rcut\_smth  
 (Argument), 102  
 model[standard]/descriptor[se\_e2\_a]/rcut\_smth  
 (Argument), 94  
 model[standard]/descriptor[se\_e2\_r]/rcut\_smth  
 (Argument), 100  
 model[standard]/descriptor[se\_e3]/rcut\_smth  
 (Argument), 96  
 reduce() (deepmd.utils.data.DeepmdData method),  
 526  
 reduce() (deepmd.utils.data\_system.DeepmdDataSystem  
 method), 529  
 reduce() (deepmd.utils.DeepmdData method), 513  
 reduce() (deepmd.utils.DeepmdDataSystem  
 method), 516  
 reduce() (deepmd\_utils.utils.data.DeepmdData  
 method), 590  
 reduce() (deepmd\_utils.utils.data\_system.DeepmdDataSystem  
 method), 593  
 register() (deepmd.descriptor.Descriptor static  
 method), 233  
 register() (deepmd.descriptor.descriptor.Descriptor  
 static method), 285  
 register() (deepmd.fit.Fitting static method), 352  
 register() (deepmd.fit.fitting.Fitting static  
 method), 367  
 register() (deepmd.utils.Plugin method), 518  
 register() (deepmd.utils.plugin.Plugin method),  
 545  
 register() (deepmd\_utils.utils.argcheck.ArgsPlugin  
 method), 581  
 register() (deepmd\_utils.utils.plugin.Plugin  
 method), 599  
 reinit() (deepmd.utils.pair\_tab.PairTab method),  
 539

`reinit()` (deepmd.utils.PairTab method), 517  
`reinit()` (deepmd\_utils.utils.pair\_tab.PairTab method), 594  
`relative_f`:  
    `loss[ener_spin]/relative_f` (Argument), 123  
    `loss[ener]/relative_f` (Argument), 120  
`remove_decay_rate()` (in module `deepmd_utils.utils.compat`), 586  
`replace_model_params_with_frz_multi_model()` (in module `deepmd.utils.multi_init`), 536  
`replace_model_params_with_pretrained_model()` (in module `deepmd.utils.finetune`), 530  
`reset_default_tf_session_config()` (in module `deepmd.env`), 559  
`reset_get_batch()` (deepmd.utils.data.DeepmdData method), 526  
`reset_get_batch()` (deepmd.utils.DeepmdData method), 513  
`reset_get_batch()` (deepmd\_utils.utils.data.DeepmdData method), 590  
`resnet_dt`:  
    `model/type_embedding/resnet_dt` (Argument), 90  
    `model[standard]/descriptor[se_a_ebd_v2]/resnet_dt` (Argument), 108  
    `model[standard]/descriptor[se_a_mask]/resnet_dt` (Argument), 110  
    `model[standard]/descriptor[se_a_tpe]/resnet_dt` (Argument), 98  
    `model[standard]/descriptor[se_atten_v2]/resnet_dt` (Argument), 106  
    `model[standard]/descriptor[se_atten]/resnet_dt` (Argument), 103  
    `model[standard]/descriptor[se_e2_a]/resnet_dt` (Argument), 95  
    `model[standard]/descriptor[se_e2_r]/resnet_dt` (Argument), 101  
    `model[standard]/descriptor[se_e3]/resnet_dt` (Argument), 97  
    `model[standard]/fitting_net[dipole]/resnet_dt` (Argument), 116  
    `model[standard]/fitting_net[dos]/resnet_dt` (Argument), 114  
    `model[standard]/fitting_net[ener]/resnet_dt` (Argument), 112  
    `model[standard]/fitting_net[polar]/resnet_dt` (Argument), 115  
`restore_descriptor`:  
    `nvnm/restore_descriptor` (Argument), 130  
`restore_fitting_net`:  
    `nvnm/restore_fitting_net` (Argument), 130  
`reverse_bin()` (deepmd.nvnmd.utils.Encode method), 495  
`reverse_bin()` (deepmd.nvnmd.utils.encode.Encode method), 502  
`reverse_map()` (deepmd.DeepEval static method), 221  
`reverse_map()` (deepmd.infer.deep\_eval.DeepEval static method), 401  
`reverse_map()` (deepmd.infer.DeepEval static method), 379  
`rglob()` (deepmd.utils.path.DPH5Path method), 542  
`rglob()` (deepmd.utils.path.DPOSPath method), 543  
`rglob()` (deepmd.utils.path.DPPPath method), 544  
`rglob()` (deepmd\_utils.utils.path.DPH5Path method), 596  
`rglob()` (deepmd\_utils.utils.path.DPOSPath method), 597  
`rglob()` (deepmd\_utils.utils.path.DPPPath method), 598  
`run_s2g()` (deepmd.nvnmd.entrypoints.maptable.MapTable method), 490  
`run_s2g()` (deepmd.nvnmd.entrypoints.MapTable method), 485  
`run_sess()` (in module `deepmd.utils.sess`), 547  
`run_t2g()` (deepmd.nvnmd.entrypoints.maptable.MapTable method), 490  
`run_t2g()` (deepmd.nvnmd.entrypoints.MapTable method), 485  
`run_t2s()` (deepmd.nvnmd.entrypoints.maptable.MapTable method), 490  
`run_t2s()` (deepmd.nvnmd.entrypoints.MapTable method), 485  
`RunOptions` (class in `deepmd.train.run_options`), 507  
`SafeDict`  
    `safe_cast_tensor()` (in module `deepmd.common`), 558  
`save()` (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
`save()` (deepmd.nvnmd.utils.fio.FioBin method), 503  
`save()` (deepmd.nvnmd.utils.fio.FioDic method), 503  
`save()` (deepmd.nvnmd.utils.fio.FioJsonDic method), 504  
`save()` (deepmd.nvnmd.utils.fio.FioNpyDic method), 504  
`save()` (deepmd.nvnmd.utils.fio.FioTxt method), 504  
`save()` (deepmd.nvnmd.utils.FioBin method), 495  
`save()` (deepmd.nvnmd.utils.FioDic method), 495  
`save()` (deepmd.nvnmd.utils.FioTxt method), 496  
`save_checkpoint()` (deepmd.train.trainer.DPTrainer method), 509  
`save_ckpt`:  
    `training/save_ckpt` (Argument), 128  
`save_compressed()` (deepmd.train.trainer.DPTrainer method), 509

---

```

save_dp_model()          (in module
    deepmd_utils.model_format), 570
save_dp_model()          (in module
    deepmd_utils.model_format.network),
    575
save_freq:
    training/save_freq (Argument), 128
save_weight()            (in module
    deepmd.nvnmd.entrypoints), 486
save_weight()            (in module
    deepmd.nvnmd.entrypoints.freeze), 487
scale:
    model[standard]/fitting_net[polar]/scale
        (Argument), 115
scale_by_worker:
    learning_rate/scale_by_worker (Argument),
        118
seed() (in module deepmd_utils.random), 547
seed() (in module deepmd_utils.utils.random), 600
seed:
    model/type_embedding/seed (Argument), 91
    model[standard]/descriptor[se_a_ebd_v2]/seed
        (Argument), 109
    model[standard]/descriptor[se_a_mask]/seed
        (Argument), 111
    model[standard]/descriptor[se_a_tpe]/seed
        (Argument), 99
    model[standard]/descriptor[se_atten_v2]/seed
        (Argument), 106
    model[standard]/descriptor[se_atten]/seed
        (Argument), 103
    model[standard]/descriptor[se_e2_a]/seed
        (Argument), 95
    model[standard]/descriptor[se_e2_r]/seed
        (Argument), 101
    model[standard]/descriptor[se_e3]/seed
        (Argument), 97
    model[standard]/fitting_net[dipole]/seed
        (Argument), 116
    model[standard]/fitting_net[dos]/seed
        (Argument), 114
    model[standard]/fitting_net[ener]/seed
        (Argument), 112
    model[standard]/fitting_net[polar]/seed
        (Argument), 115
    training/seed (Argument), 128
sel:
    model[pairtab]/sel (Argument), 117
    model[standard]/descriptor[se_a_ebd_v2]/sel
        (Argument), 107
    model[standard]/descriptor[se_a_mask]/sel
        (Argument), 109
    model[standard]/descriptor[se_a_tpe]/sel
        (Argument), 97
    model[standard]/descriptor[se_atten_v2]/sel
        (Argument), 104
    model[standard]/descriptor[se_atten]/sel
        (Argument), 102
    model[standard]/descriptor[se_e2_a]/sel
        (Argument), 94
    model[standard]/descriptor[se_e2_r]/sel
        (Argument), 100
    model[standard]/descriptor[se_e3]/sel
        (Argument), 96
    sel_a:
        model[standard]/descriptor[loc_frame]/sel_a
            (Argument), 93
    sel_r:
        model[standard]/descriptor[loc_frame]/sel_r
            (Argument), 93
    sel_type:
        model[standard]/fitting_net[dipole]/sel_type
            (Argument), 116
        model[standard]/fitting_net[polar]/sel_type
            (Argument), 115
    select_idx_map() (in module deepmd.common),
        558
    select_idx_map() (in module
        deepmd_utils.common), 603
    serialize() (deepmd_utils.model_format.DescriptSeA
        method), 565
    serialize() (deepmd_utils.model_format.env_mat.EnvMat
        method), 572
    serialize() (deepmd_utils.model_format.EnvMat
        method), 565
    serialize() (deepmd_utils.model_format.NativeLayer
        method), 568
    serialize() (deepmd_utils.model_format.network.NativeLayer
        method), 574
    serialize() (deepmd_utils.model_format.network.NetworkCollection
        method), 575
    serialize() (deepmd_utils.model_format.NetworkCollection
        method), 569
    serialize() (deepmd_utils.model_format.se_e2_a.DescriptSeA
        method), 580
    sess (deepmd.DeepEval property), 221
    sess (deepmd.infer.deep_eval.DeepEval property),
        401
    sess (deepmd.infer.DeepEval property), 379
    set_davg_zero:
        model[standard]/descriptor[se_a_ebd_v2]/set_davg_zero
            (Argument), 109
        model[standard]/descriptor[se_a_tpe]/set_davg_zero
            (Argument), 99
        model[standard]/descriptor[se_atten_v2]/set_davg_zero
            (Argument), 107
        model[standard]/descriptor[se_atten]/set_davg_zero
            (Argument), 104

```



model[standard]/descriptor[se\_e2\_a]/set\_davg\_zero function), 804  
 (Argument), 96 SimulationRegion::getNullShiftIndex (C++  
 model[standard]/descriptor[se\_e2\_r]/set\_davg\_zero function), 803  
 (Argument), 101 SimulationRegion::getNumShiftVec (C++ func-  
 model[standard]/descriptor[se\_e3]/set\_davg\_zero tion), 803  
 (Argument), 97 SimulationRegion::getRecBoxTensor (C++ func-  
 set\_log\_handles() (in module deepmd.loggers), 418 tion), 802  
 set\_log\_handles() (in module SimulationRegion::getShiftIndex (C++ func-  
 deepmd.loggers.loggers), 419 tion), 803  
 set\_log\_handles() (in module SimulationRegion::getShiftVec (C++ function),  
 deepmd\_utils.loggers), 561 803  
 set\_log\_handles() (in module SimulationRegion::getShiftVecTotalSize (C++  
 deepmd\_utils.loggers.loggers), 562 function), 803  
 set\_ntype() (deepmd.nvnmd.utils.config.NvnmdConfig SimulationRegion::getVolume (C++ function),  
 method), 499 803  
 set\_prefix: SimulationRegion::index3to1 (C++ function),  
 training/training\_data/set\_prefix 804  
 (Argument), 125 SimulationRegion::inter2Phys (C++ function),  
 training/validation\_data/set\_prefix 803  
 (Argument), 126 SimulationRegion::inter\_shift\_vec (C++ mem-  
 set\_scikit\_build\_env() (in module back- ber), 804  
 end.read\_env), 218 SimulationRegion::isPeriodic (C++ function),  
 set\_sys\_probs() (deepmd.utils.data\_system.DeepmdDataSystem 803  
 method), 529 SimulationRegion::NBOX\_XX (C++ member), 804  
 set\_sys\_probs() (deepmd.utils.DeepmdDataSystem SimulationRegion::NBOX\_YY (C++ member), 804  
 method), 516 SimulationRegion::NBOX\_ZZ (C++ member), 804  
 set\_sys\_probs() (deepmd\_utils.utils.data\_system.DeepmdDataSystem SimulationRegion::phys2Inter (C++ function),  
 method), 593 803  
 shift\_diag: SimulationRegion::recover (C++ function), 802  
 model[standard]/fitting\_net[polar]/shift\_diag SimulationRegion::reinitBox (C++ function),  
 (Argument), 115 802  
 shuffle() (in module deepmd.utils.random), 547 SimulationRegion::reinitOrigin (C++ function),  
 shuffle() (in module deepmd\_utils.utils.random), 802  
 600 SimulationRegion::shift\_info\_size (C++ mem-  
 SimulationRegion (C++ class), 802 ber), 804  
 SimulationRegion::~~SimulationRegion (C++ SimulationRegion::shift\_vec (C++ member), 804  
 function), 802 SimulationRegion::shift\_vec\_size (C++ mem-  
 SimulationRegion::affineTransform (C++ func- ber), 804  
 tion), 802 SimulationRegion::shiftCoord (C++ function),  
 SimulationRegion::backup (C++ function), 802 803  
 SimulationRegion::compactIndex (C++ function), SimulationRegion::SimulationRegion (C++  
 803 function), 802  
 SimulationRegion::computeShiftVec (C++ func- SimulationRegion::SPACENDIM (C++ member), 804  
 tion), 804 SimulationRegion::toFaceDistance (C++ func-  
 SimulationRegion::DBOX\_XX (C++ member), 804 tion), 803  
 SimulationRegion::DBOX\_YY (C++ member), 804 smin\_alpha:  
 SimulationRegion::DBOX\_ZZ (C++ member), 804 model/smin\_alpha (Argument), 89  
 SimulationRegion::diffNearestNeighbor (C++ smooth\_type\_embdding:  
 function), 803 model[standard]/descriptor[se\_atten]/smooth\_type\_embdding  
 SimulationRegion::getBoxOrigin (C++ function), (Argument), 104  
 802, 803 soft\_min\_force() (in module  
 SimulationRegion::getBoxTensor (C++ function), deepmd.env.op\_module), 651  
 802 soft\_min\_force\_grad() (in module  
 SimulationRegion::getInterShiftVec (C++ deepmd.env.op\_grads\_module), 666

---

<code>soft_min_switch()</code>	(in module <code>deepmd.env.op_module</code> ), 651	<code>learning_rate[exp]/start_lr</code> (Argument), 118
<code>soft_min_virial()</code>	(in module <code>deepmd.env.op_module</code> ), 652	<code>start_pref()</code> (in module <code>deepmd_utils.utils.argcheck</code> ), 583
<code>soft_min_virial_grad()</code>	(in module <code>deepmd.env.op_grads_module</code> ), 666	<code>start_pref_acdf:</code> <code>loss[dos]/start_pref_acdf</code> (Argument), 124
<code>SoftMinForce()</code>	(in module <code>deepmd.env.op_module</code> ), 625	<code>start_pref_ados:</code> <code>loss[dos]/start_pref_ados</code> (Argument), 124
<code>SoftMinForceGrad()</code>	(in module <code>deepmd.env.op_grads_module</code> ), 662	<code>start_pref_ae:</code> <code>loss[ener_spin]/start_pref_ae</code> (Argument), 122
<code>SoftMinSwitch()</code>	(in module <code>deepmd.env.op_module</code> ), 625	<code>loss[ener]/start_pref_ae</code> (Argument), 120
<code>SoftMinVirial()</code>	(in module <code>deepmd.env.op_module</code> ), 625	<code>start_pref_cdf:</code> <code>loss[dos]/start_pref_cdf</code> (Argument), 123
<code>SoftMinVirialGrad()</code>	(in module <code>deepmd.env.op_grads_module</code> ), 663	<code>start_pref_dos:</code> <code>loss[dos]/start_pref_dos</code> (Argument), 123
<code>sort_input()</code> ( <code>deepmd.DeepEval</code> static method), 222		<code>start_pref_e:</code> <code>loss[ener_spin]/start_pref_e</code> (Argument), 121
<code>sort_input()</code> ( <code>deepmd.infer.deep_eval.DeepEval</code> static method), 401		<code>loss[ener]/start_pref_e</code> (Argument), 119
<code>sort_input()</code> ( <code>deepmd.infer.DeepEval</code> static method), 379		<code>start_pref_f:</code> <code>loss[ener]/start_pref_f</code> (Argument), 119
<code>Spin</code> (class in <code>deepmd.utils.spin</code> ), 547		<code>start_pref_fm:</code> <code>loss[ener_spin]/start_pref_fm</code> (Argument), 122
<code>spin:</code> <code>model/spin</code> (Argument), 92		<code>start_pref_fr:</code> <code>loss[ener_spin]/start_pref_fr</code> (Argument), 121
<code>spin_args()</code> (in module <code>deepmd_utils.utils.argcheck</code> ), 583		<code>start_pref_gf:</code> <code>loss[ener]/start_pref_gf</code> (Argument), 121
<code>spin_norm:</code> <code>model/spin/spin_norm</code> (Argument), 92		<code>start_pref_pf:</code> <code>loss[ener_spin]/start_pref_pf</code> (Argument), 123
<code>split_bin()</code> ( <code>deepmd.nvnmd.utils.Encode</code> method), 495		<code>loss[ener]/start_pref_pf</code> (Argument), 120
<code>split_bin()</code> ( <code>deepmd.nvnmd.utils.encode.Encode</code> method), 502		<code>start_pref_v:</code> <code>loss[ener_spin]/start_pref_v</code> (Argument), 122
<code>split_expo_mant()</code> ( <code>deepmd.nvnmd.utils.Encode</code> method), 495		<code>loss[ener]/start_pref_v</code> (Argument), 120
<code>split_expo_mant()</code> ( <code>deepmd.nvnmd.utils.encode.Encode</code> method), 502		<code>stop_lr:</code> <code>learning_rate[exp]/stop_lr</code> (Argument), 118
<code>splitflt</code> (C++ function), 844		<code>stripped_type_embedding:</code> <code>model[standard]/descriptor[se_atten]/stripped_type_emb</code> (Argument), 104
<code>SQRT_2_PI</code> (C macro), 850		<code>sw_rmax:</code> <code>model/sw_rmax</code> (Argument), 90
<code>srtab_add_bias:</code> <code>model/srtab_add_bias</code> (Argument), 90		<code>sw_rmin:</code> <code>model/sw_rmin</code> (Argument), 90
<code>standard_model_args()</code> (in module <code>deepmd_utils.utils.argcheck</code> ), 583		<code>sys_charge_map:</code> <code>model/modifier[dipole_charge]/sys_charge_map</code> (Argument), 91
<code>StandardModel</code> (class in <code>deepmd.model.model</code> ), 464		<code>sys_probs:</code> <code>training/training_data/sys_probs</code> (Argument), 126
<code>start_dpGUI()</code> (in module <code>deepmd.entrypoints</code> ), 335		
<code>start_dpGUI()</code> (in module <code>deepmd.entrypoints.gui</code> ), 338		
<code>start_dpGUI()</code> (in module <code>deepmd_utils.entrypoints.gui</code> ), 560		
<code>start_lr()</code> ( <code>deepmd.utils.learning_rate.LearningRateExp</code> method), 536		
<code>start_lr()</code> ( <code>deepmd.utils.LearningRateExp</code> method), 517		
<code>start_lr:</code>		

---

training/validation\_data/sys\_probs  
 (Argument), 127  
 systems:  
 training/training\_data/systems  
 (Argument), 125  
 training/validation\_data/systems  
 (Argument), 126

## T

tab\_file:  
 model[pairtab]/tab\_file (Argument), 117  
 tabulate\_fusion() (in module  
 deepmd.env.op\_module), 652  
 tabulate\_fusion\_grad() (in module  
 deepmd.env.op\_module), 652  
 tabulate\_fusion\_grad\_grad() (in module  
 deepmd.env.op\_module), 653  
 tabulate\_fusion\_se\_a() (in module  
 deepmd.env.op\_module), 653  
 tabulate\_fusion\_se\_a\_grad() (in module  
 deepmd.env.op\_module), 653  
 tabulate\_fusion\_se\_a\_grad\_grad() (in module  
 deepmd.env.op\_module), 654  
 tabulate\_fusion\_se\_atten() (in module  
 deepmd.env.op\_module), 654  
 tabulate\_fusion\_se\_atten\_grad() (in module  
 deepmd.env.op\_module), 655  
 tabulate\_fusion\_se\_atten\_grad\_grad() (in mod-  
 ule deepmd.env.op\_module), 655  
 tabulate\_fusion\_se\_r() (in module  
 deepmd.env.op\_module), 656  
 tabulate\_fusion\_se\_r\_grad() (in module  
 deepmd.env.op\_module), 656  
 tabulate\_fusion\_se\_r\_grad\_grad() (in module  
 deepmd.env.op\_module), 656  
 tabulate\_fusion\_se\_t() (in module  
 deepmd.env.op\_module), 656  
 tabulate\_fusion\_se\_t\_grad() (in module  
 deepmd.env.op\_module), 657  
 tabulate\_fusion\_se\_t\_grad\_grad() (in module  
 deepmd.env.op\_module), 657  
 TabulateCheckerGPUExecuteFunctor (C++ struct),  
 801  
 TabulateCheckerGPUExecuteFunctor::operator()  
 (C++ function), 801  
 TabulateFusion() (in module  
 deepmd.env.op\_module), 626  
 TabulateFusionGPUExecuteFunctor (C++ struct),  
 801  
 TabulateFusionGPUExecuteFunctor::operator()  
 (C++ function), 801  
 TabulateFusionGrad() (in module  
 deepmd.env.op\_module), 626  
 TabulateFusionGradGPUExecuteFunctor (C++  
 struct), 802  
 TabulateFusionGradGPUExecuteFunctor::operator()  
 (C++ function), 802  
 TabulateFusionGradGrad() (in module  
 deepmd.env.op\_module), 627  
 TabulateFusionSeA() (in module  
 deepmd.env.op\_module), 627  
 TabulateFusionSeAGrad() (in module  
 deepmd.env.op\_module), 627  
 TabulateFusionSeAGradGrad() (in module  
 deepmd.env.op\_module), 628  
 TabulateFusionSeAtten() (in module  
 deepmd.env.op\_module), 628  
 TabulateFusionSeAttenGrad() (in module  
 deepmd.env.op\_module), 628  
 TabulateFusionSeAttenGradGrad() (in module  
 deepmd.env.op\_module), 629  
 TabulateFusionSeR() (in module  
 deepmd.env.op\_module), 629  
 TabulateFusionSeRGrad() (in module  
 deepmd.env.op\_module), 630  
 TabulateFusionSeRGradGrad() (in module  
 deepmd.env.op\_module), 630  
 TabulateFusionSeT() (in module  
 deepmd.env.op\_module), 630  
 TabulateFusionSeTGrad() (in module  
 deepmd.env.op\_module), 631  
 TabulateFusionSeTGradGrad() (in module  
 deepmd.env.op\_module), 631  
 tanh4() (in module deepmd.nvnmd.utils.network),  
 505  
 tanh4\_flt\_nvnmd() (in module  
 deepmd.env.op\_module), 658  
 Tanh4FltNvnmd() (in module  
 deepmd.env.op\_module), 631  
 tensorboard:  
 training/tensorboard (Argument), 129  
 tensorboard\_freq:  
 training/tensorboard\_freq (Argument), 129  
 tensorboard\_log\_dir:  
 training/tensorboard\_log\_dir (Argument),  
 129  
 TensorLoss (class in deepmd.loss), 425  
 TensorLoss (class in deepmd.loss.tensor), 433  
 TensorModel (class in deepmd.model.tensor), 477  
 tensors (deepmd.infer.deep\_tensor.DeepTensor at-  
 tribute), 412  
 test() (in module deepmd.entrypoints), 335  
 test() (in module deepmd.entrypoints.test), 340  
 time\_training:  
 training/time\_training (Argument), 128  
 TPB (C macro), 850  
 train() (deepmd.train.trainer.DPTrainer method),



[509](#)  
[train\(\)](#) (in module `deepmd.entrypoints.train`), [341](#)  
[train\\_dp\(\)](#) (in module `deepmd.entrypoints`), [336](#)  
[train\\_nvnmd\(\)](#) (in module `deepmd.nvnmd.entrypoints.train`), [490](#)  
[trainable:](#)  
    [model/type\\_embedding/trainable](#)  
        ([Argument](#)), [91](#)  
    [model\[standard\]/descriptor\[se\\_a\\_ebd\\_v2\]/trainable](#)  
        ([Argument](#)), [109](#)  
    [model\[standard\]/descriptor\[se\\_a\\_mask\]/trainable](#)  
        ([Argument](#)), [110](#)  
    [model\[standard\]/descriptor\[se\\_a\\_tpe\]/trainable](#)  
        ([Argument](#)), [99](#)  
    [model\[standard\]/descriptor\[se\\_atten\\_v2\]/trainable](#)  
        ([Argument](#)), [106](#)  
    [model\[standard\]/descriptor\[se\\_atten\]/trainable](#)  
        ([Argument](#)), [103](#)  
    [model\[standard\]/descriptor\[se\\_e2\\_a\]/trainable](#)  
        ([Argument](#)), [95](#)  
    [model\[standard\]/descriptor\[se\\_e2\\_r\]/trainable](#)  
        ([Argument](#)), [101](#)  
    [model\[standard\]/descriptor\[se\\_e3\]/trainable](#)  
        ([Argument](#)), [97](#)  
    [model\[standard\]/fitting\\_net\[dos\]/trainable](#)  
        ([Argument](#)), [114](#)  
    [model\[standard\]/fitting\\_net\[ener\]/trainable](#)  
        ([Argument](#)), [112](#)  
[training](#) ([Argument](#))  
    [training:](#), [125](#)  
[training/data\\_dict](#) ([Argument](#))  
    [data\\_dict:](#), [129](#)  
[training/disp\\_file](#) ([Argument](#))  
    [disp\\_file:](#), [128](#)  
[training/disp\\_freq](#) ([Argument](#))  
    [disp\\_freq:](#), [128](#)  
[training/disp\\_training](#) ([Argument](#))  
    [disp\\_training:](#), [128](#)  
[training/enable\\_profiler](#) ([Argument](#))  
    [enable\\_profiler:](#), [129](#)  
[training/fitting\\_weight](#) ([Argument](#))  
    [fitting\\_weight:](#), [129](#)  
[training/mixed\\_precision](#) ([Argument](#))  
    [mixed\\_precision:](#), [127](#)  
[training/mixed\\_precision/compute\\_prec](#)  
    ([Argument](#))  
        [compute\\_prec:](#), [128](#)  
[training/mixed\\_precision/output\\_prec](#)  
    ([Argument](#))  
        [output\\_prec:](#), [127](#)  
[training/numb\\_steps](#) ([Argument](#))  
    [numb\\_steps:](#), [128](#)  
[training/profiling](#) ([Argument](#))  
    [profiling:](#), [129](#)  
    [training/profiling\\_file](#) ([Argument](#))  
        [profiling\\_file:](#), [129](#)  
[training/save\\_ckpt](#) ([Argument](#))  
    [save\\_ckpt:](#), [128](#)  
[training/save\\_freq](#) ([Argument](#))  
    [save\\_freq:](#), [128](#)  
[training/seed](#) ([Argument](#))  
    [seed:](#), [128](#)  
[training/tensorboard](#) ([Argument](#))  
    [tensorboard:](#), [129](#)  
[training/tensorboard\\_freq](#) ([Argument](#))  
    [tensorboard\\_freq:](#), [129](#)  
[training/tensorboard\\_log\\_dir](#) ([Argument](#))  
    [tensorboard\\_log\\_dir:](#), [129](#)  
[training/time\\_training](#) ([Argument](#))  
    [time\\_training:](#), [128](#)  
[training/training\\_data](#) ([Argument](#))  
    [training\\_data:](#), [125](#)  
[training/training\\_data/auto\\_prob](#) ([Argument](#))  
    [auto\\_prob:](#), [126](#)  
[training/training\\_data/batch\\_size](#) ([Argument](#))  
    [batch\\_size:](#), [125](#)  
[training/training\\_data/set\\_prefix](#) ([Argument](#))  
    [set\\_prefix:](#), [125](#)  
[training/training\\_data/sys\\_probs](#) ([Argument](#))  
    [sys\\_probs:](#), [126](#)  
[training/training\\_data/systems](#) ([Argument](#))  
    [systems:](#), [125](#)  
[training/validation\\_data](#) ([Argument](#))  
    [validation\\_data:](#), [126](#)  
[training/validation\\_data/auto\\_prob](#)  
    ([Argument](#))  
    [auto\\_prob:](#), [127](#)  
[training/validation\\_data/batch\\_size](#)  
    ([Argument](#))  
    [batch\\_size:](#), [127](#)  
[training/validation\\_data/numb\\_btch](#)  
    ([Argument](#))  
    [numb\\_btch:](#), [127](#)  
[training/validation\\_data/set\\_prefix](#)  
    ([Argument](#))  
    [set\\_prefix:](#), [126](#)  
[training/validation\\_data/sys\\_probs](#)  
    ([Argument](#))  
    [sys\\_probs:](#), [127](#)  
[training/validation\\_data/systems](#) ([Argument](#))  
    [systems:](#), [126](#)  
[training:](#)  
    [training](#) ([Argument](#)), [125](#)  
[training\\_args\(\)](#) (in module `deepmd_utils.utils.argcheck`), [583](#)  
[training\\_data:](#)  
    [training/training\\_data](#) ([Argument](#)), [125](#)

training\_data\_args() (in module deepmd\_utils.utils.argcheck), 583  
 transfer() (in module deepmd.entypoints), 336  
 transfer() (in module deepmd.entypoints.transfer), 342  
 traverse\_model\_dict() (in module deepmd\_utils.model\_format), 570  
 traverse\_model\_dict() (in module deepmd\_utils.model\_format.network), 576  
 type:  
   learning\_rate/type (Argument), 118  
   loss/type (Argument), 119  
   model/modifier/type (Argument), 91  
   model/type (Argument), 92  
   model[standard]/descriptor/type (Argument), 92  
   model[standard]/fitting\_net/type (Argument), 111  
 type\_embedding:  
   model/type\_embedding (Argument), 90  
 type\_embedding\_args() (in module deepmd\_utils.argcheck), 519  
 type\_embedding\_args() (in module deepmd\_utils.utils.argcheck), 583  
 type\_map:  
   model/type\_map (Argument), 89  
 type\_nchanl:  
   model[standard]/descriptor[se\_a\_tpe]/type\_nchanl (Argument), 99  
 type\_nlayer:  
   model[standard]/descriptor[se\_a\_tpe]/type\_nlayer (Argument), 99  
 type\_one\_side:  
   model[standard]/descriptor[se\_a\_ebd\_v2]/type\_one\_side (Argument), 108  
   model[standard]/descriptor[se\_a\_mask]/type\_one\_side (Argument), 110  
   model[standard]/descriptor[se\_a\_tpe]/type\_one\_side (Argument), 98  
   model[standard]/descriptor[se\_atten\_v2]/type\_one\_side (Argument), 106  
   model[standard]/descriptor[se\_atten]/type\_one\_side (Argument), 103  
   model[standard]/descriptor[se\_e2\_a]/type\_one\_side (Argument), 95  
   model[standard]/descriptor[se\_e2\_r]/type\_one\_side (Argument), 101  
 TypeEmbedNet (class in deepmd\_utils.type\_embed), 550  
 U  
 U\_Flt64\_Int64 (C++ union), 805  
 U\_Flt64\_Int64::nint (C++ member), 805  
 uint\_64 (C++ type), 851  
 unaggregated\_dy2\_dx() (in module deepmd.env.op\_module), 658  
 unaggregated\_dy2\_dx\_s() (in module deepmd.env.op\_module), 658  
 unaggregated\_dy\_dx() (in module deepmd.env.op\_module), 658  
 unaggregated\_dy\_dx\_s() (in module deepmd.env.op\_module), 659  
 UnaggregatedDy2Dx() (in module deepmd.env.op\_module), 632  
 UnaggregatedDy2DxS() (in module deepmd.env.op\_module), 632  
 UnaggregatedDyDx() (in module deepmd.env.op\_module), 632  
 UnaggregatedDyDxS() (in module deepmd.env.op\_module), 632  
 update() (deepmd.nvnmd.utils.fio.FioDic method), 503  
 update() (deepmd.nvnmd.utils.FioDic method), 495  
 update\_config() (deepmd.nvnmd.utils.config.NvnmdConfig method), 499  
 update\_deepmd\_input() (in module deepmd\_utils.compat), 520  
 update\_deepmd\_input() (in module deepmd\_utils.utils.compat), 586  
 update\_sel() (deepmd.descriptor.Descriptor class method), 233  
 update\_sel() (deepmd.descriptor.descriptor.Descriptor class method), 285  
 update\_sel() (deepmd.descriptor.DescriptHybrid class method), 238  
 update\_sel() (deepmd.descriptor.DescriptLocFrame class method), 242  
 update\_sel() (deepmd.descriptor.DescriptSeAMask class method), 262  
 update\_sel() (deepmd.descriptor.DescriptSeAtten class method), 267  
 update\_sel() (deepmd.descriptor.hybrid.DescriptHybrid class method), 290  
 update\_sel() (deepmd.descriptor.loc\_frame.DescriptLocFrame class method), 294  
 update\_sel() (deepmd.descriptor.se.DescriptSe class method), 296  
 update\_sel() (deepmd.descriptor.se\_a\_mask.DescriptSeAMask class method), 316  
 update\_sel() (deepmd.descriptor.se\_atten.DescriptSeAtten class method), 321  
 update\_sel() (deepmd.model.frozen.FrozenModel class method), 454  
 update\_sel() (deepmd.model.linear.LinearModel class method), 458  
 update\_sel() (deepmd.model.model.Model class method), 458

method), 463

update\_sel() (deepmd.model.model.StandardModel class method), 465

update\_sel() (deepmd.model.multi.MultiModel class method), 469

update\_sel() (deepmd.model.MultiModel class method), 444

update\_sel() (deepmd.model.pairtab.PairTabModel class method), 471

update\_sel() (deepmd.model.pairwise\_dprc.PairwiseDPRC class method), 474

use\_aparam\_as\_mask:  
model[standard]/fitting\_net[ener]/use\_aparam\_as\_mask (Argument), 113

use\_spin:  
model/spin/use\_spin (Argument), 92

use\_srtab:  
model/use\_srtab (Argument), 89

## V

valid\_on\_the\_fly() (deepmd.train.trainer.DPTrainer method), 509

validation\_data:  
training/validation\_data (Argument), 126

validation\_data\_args() (in module deepmd\_utils.utils.argcheck), 583

value() (deepmd.utils.learning\_rate.LearningRateExp method), 536

value() (deepmd.utils.LearningRateExp method), 517

variable\_summaries() (in module deepmd\_utils.network), 539

VariantABCMeta (class in deepmd\_utils.plugin), 546

VariantABCMeta (class in deepmd\_utils.plugin), 599

VariantMeta (class in deepmd\_utils.plugin), 546

VariantMeta (class in deepmd\_utils.plugin), 600

version:  
nvnmmd/version (Argument), 130

virtual\_len:  
model/spin/virtual\_len (Argument), 92

## W

weight\_file:  
nvnmmd/weight\_file (Argument), 130

weighted\_average() (in module deepmd\_utils.utils.weight\_avg), 552

weighted\_average() (in module deepmd\_utils.utils.weight\_avg), 601

weights:  
model[linear\_ener]/weights (Argument), 118

WFCModel (class in deepmd.model), 445

WFCModel (class in deepmd.model.tensor), 479

world\_size (deepmd.train.run\_options.RunOptions attribute), 508

Wrap (class in deepmd.nvnmmd.entrypoints), 485

Wrap (class in deepmd.nvnmmd.entrypoints.wrap), 491

wrap() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 491

wrap() (in module deepmd.nvnmmd.entrypoints.wrap), 492

wrap\_bias() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap\_bias() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 491

wrap\_dscp() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap\_dscp() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 491

wrap\_fitn() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap\_fitn() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 492

wrap\_head() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap\_head() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 492

wrap\_lut() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap\_lut() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 492

wrap\_map() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap\_map() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 492

wrap\_weight() (deepmd.nvnmmd.entrypoints.Wrap method), 486

wrap\_weight() (deepmd.nvnmmd.entrypoints.wrap.Wrap method), 492

write\_model\_devi\_out() (in module deepmd.infer.model\_devi), 417