
DPGEN2

DeepModeling

Apr 19, 2024

USER GUIDE

1 Guide on dpgen2 commands	3
1.1 Submit a workflow	3
1.2 Check the convergence of a workflow	3
1.3 Watch the progress of a workflow	3
1.4 Show the keys of steps	4
1.5 Resubmit a workflow	4
2 Command line interface	5
2.1 Named Arguments	5
2.2 Valid subcommands	5
2.3 Sub-commands	5
2.3.1 submit	5
2.3.2 resubmit	6
2.3.3 showkey	6
2.3.4 status	7
2.3.5 download	7
2.3.6 watch	8
2.3.7 gui	8
2.3.8 terminate	9
2.3.9 stop	9
2.3.10 suspend	9
2.3.11 delete	9
2.3.12 retry	10
2.3.13 resume	10
2.3.14 restart	10
3 Guide on writing input scripts for dpgen2 commands	11
3.1 Preliminaries	11
3.2 The input script for all dpgen2 commands	11
3.3 The input script for submit and resubmit	11
3.3.1 Inputs	12
3.3.2 Training	12
3.3.3 Exploration	12
3.3.4 FP	14
3.3.5 Configuration of dflow step	14
4 Arguments of the submit script	17
4.1 Task group definition	67
4.1.1 LAMMPS task group	67
4.1.2 CALYPSO task group	72

5 Developers' guide	77
5.1 The concurrent learning algorithm	77
5.2 Overview of the DPGEN2 Implementation	78
5.3 The DPGEN2 workflow	78
5.3.1 Inside the block operator	79
5.3.2 The exploration strategy	79
5.4 How to contribute	80
6 Operators	81
6.1 The super-OP PrepRunDPTrain	81
6.2 The OP RunDPTrain	84
7 Exploration	87
7.1 Stage scheduler	87
7.2 Exploration task groups	88
7.3 Configuration selector	89
8 DPGEN2 API	91
8.1 dpgen2 package	91
8.1.1 Subpackages	91
8.1.2 Submodules	177
8.1.3 dpgen2.constants module	177
Python Module Index	179
Index	181

DPGEN2 is the 2nd generation of the Deep Potential GENerator.

Important: The project DeePMD-kit is licensed under [GNU LGPLv3.0](#).

GUIDE ON DPGEN2 COMMANDS

One may use dpgen2 through command line interface. A full documentation of the cli is found [here](#)

1.1 Submit a workflow

The dpgen2 workflow can be submitted via the `submit` command

```
dpgen2 submit input.json
```

where `input.json` is the input script. A guide of writing the script is found [here](#). When a workflow is submitted, a ID (WFID) of the workflow will be printed for later reference.

1.2 Check the convergence of a workflow

The convergence of stages of the workflow can be checked by the `status` command. It prints the indexes of the finished stages, iterations, and the accurate, candidate and failed ratio of explored configurations of each iteration.

```
$ dpgen2 status input.json WFID
#   stage  id_stg.    iter.      accu.      cand.      fail.
# Stage    0  -----
#          0      0      0  0.8333  0.1667  0.0000
#          0      1      1  0.7593  0.2407  0.0000
#          0      2      2  0.7778  0.2222  0.0000
#          0      3      3  1.0000  0.0000  0.0000
# Stage    0 converged YES reached max numb iterations NO
# All stages converged
```

1.3 Watch the progress of a workflow

The progress of a workflow can be watched on-the-fly

```
$ dpgen2 watch input.json WFID
INFO:root:steps iter-000000--prep-run-train----- finished
INFO:root:steps iter-000000--prep-run-explore----- finished
INFO:root:steps iter-000000--prep-run-fp----- finished
INFO:root:steps iter-000000--collect-data----- finished
```

(continues on next page)

(continued from previous page)

```
INFO:root:steps iter-000001--prep-run-train----- finished
INFO:root:steps iter-000001--prep-run-explore----- finished
...
```

The artifacts can be downloaded on-the-fly with -d flag. Note that the existing files are automatically skipped if one sets dflow_config["archive_mode"] = None.

1.4 Show the keys of steps

Each dpigen2 step is assigned a unique key. The keys of the finished steps can be checked with showkey command

```
0 : iter-000000--prep-train
1 -> 4 : iter-000000--run-train-0000 -> iter-000000--run-train-0003
5 : iter-000000--prep-lmp
6 -> 14 : iter-000000--run-lmp-000000 -> iter-000000--run-lmp-000008
15 : iter-000000--select-confs
16 : iter-000000--prep-fp
17 -> 20 : iter-000000--run-fp-000000 -> iter-000000--run-fp-000003
21 : iter-000000--collect-data
22 : iter-000000--scheduler
23 : iter-000000--id
24 : iter-000001--prep-train
25 -> 28 : iter-000001--run-train-0000 -> iter-000001--run-train-0003
29 : iter-000001--prep-lmp
30 -> 38 : iter-000001--run-lmp-000000 -> iter-000001--run-lmp-000008
39 : iter-000001--select-confs
40 : iter-000001--prep-fp
41 -> 44 : iter-000001--run-fp-000000 -> iter-000001--run-fp-000003
45 : iter-000001--collect-data
46 : iter-000001--scheduler
47 : iter-000001--id
```

1.5 Resubmit a workflow

If a workflow stopped abnormally, one may submit a new workflow with some steps of the old workflow reused.

```
dpigen2 resubmit input.json WFID --reuse 0-41
```

The steps of workflow WDID 0-41 (0<=id<41, note that 41 is not included) will be reused in the new workflow. The indexes of the steps are printed by dpigen2 showkey. In the example, all the steps before the iter-000001--run-fp-000000 will be used in the new workflow.

CHAPTER
TWO

COMMAND LINE INTERFACE

DPGEN2: concurrent learning workflow generating the machine learning potential energy models.

```
usage: dpgen2 [-h] [-v]
               {submit,resubmit,showkey,status,download,watch,gui,terminate,stop,suspend,
              ↵delete,retry,resume,restart}
               ...
```

2.1 Named Arguments

-v, --version show program's version number and exit

2.2 Valid subcommands

command Possible choices: submit, resubmit, showkey, status, download, watch, gui, terminate, stop, suspend, delete, retry, resume, restart

2.3 Sub-commands

2.3.1 submit

Submit DPGEN2 workflow

```
dpgen2 submit [-h] CONFIG
```

Positional Arguments

CONFIG the config file in json format defining the workflow.

2.3.2 resubmit

Submit DPGEN2 workflow resuing steps from an existing workflow

```
dpgen2 resubmit [-h] [-l] [-u REUSE [REUSE ...]] [-k] [-f] CONFIG ID
```

Positional Arguments

CONFIG the config file in json format defining the workflow.

ID the ID of the existing workflow.

Named Arguments

-l, --list list the Steps of the existing workflow.

Default: False

-u, --reuse specify which Steps to reuse.

-k, --keep-schedule if set then keep schedule of the old workflow. otherwise use the schedule defined in the input file

Default: False

-f, --fold if set then super OPs are folded to be reused in the new workflow

Default: False

2.3.3 showkey

Print the keys of the successful DPGEN2 steps

```
dpgen2 showkey [-h] CONFIG ID
```

Positional Arguments

CONFIG the config file in json format.

ID the ID of the existing workflow.

2.3.4 status

Print the status (stage, iteration, convergence) of the DPGEN2 workflow

```
dpgen2 status [-h] CONFIG ID
```

Positional Arguments

CONFIG	the config file in json format.
ID	the ID of the existing workflow.

2.3.5 download

Typically there are three ways of using the command

1. list all supported steps and their input/output artifacts \$ dpgen2 download CONFIG ID -l
2. download all the input/output of all the steps. \$ dpgen2 download CONFIG ID
3. download specified input/output artifacts of certain steps. For example \$ dpgen2 download CONFIG ID -i 0-8 8 9 -d prep-run-train/input/init_data prep-run-explore/output/trajs

The command will download the init_data of prep-run-train's input and trajs of the prep-run-explore's output from iterations 0 to 9 (by -i 0-8 8 9). The supported step and the names of input/output can be checked by the -l flag.

```
dpgen2 download [-h] [-l] [-k KEYS [KEYS ...]]  
                  [-i ITERATIONS [ITERATIONS ...]]  
                  [-d STEP_DEFINITIONS [STEP_DEFINITIONS ...]] [-p PREFIX] [-n]  
CONFIG ID
```

Positional Arguments

CONFIG	the config file in json format.
ID	the ID of the existing workflow.

Named Arguments

-l, --list-supported	list all supported steps artifacts Default: False
-k, --keys	the keys of the downloaded steps. If not provided download all artifacts
-i, --iterations	the iterations to be downloaded, support ranging expression as 0-10.
-d, --step-definitions	the definition for downloading step artifacts
-p, --prefix	the prefix of the path storing the download artifacts
-n, --no-check-point	if specified, download regardless whether check points exist. Default: True

2.3.6 watch

Watch a DPGEN2 workflow

```
dpgen2 watch [-h] [-k KEYS [KEYS ...]] [-f FREQUENCY] [-d] [-p PREFIX] [-n]
              CONFIG ID
```

Positional Arguments

CONFIG	the config file in json format.
ID	the ID of the existing workflow.

Named Arguments

-k, --keys	the subkey to watch. For example, ‘prep-run-train’ ‘prep-run-explore’ Default: ['prep-run-train', 'prep-run-explore', 'prep-run-fp', 'collect-data']
-f, --frequency	the frequency of workflow status query. In unit of second Default: 600.0
-d, --download	whether to download artifacts of a step when it finishes Default: False
-p, --prefix	the prefix of the path storing the download artifacts
-n, --no-check-point	if specified, download regardless whether check points exist. Default: True

2.3.7 gui

Serve DP-GUI.

```
dpgen2 gui [-h] [-p PORT] [--bind_all]
```

Named Arguments

-p, --port	The port to serve DP-GUI on. Default: 6042
--bind_all	Serve on all public interfaces. This will expose your DP-GUI instance to the network on both IPv4 and IPv6 (where available). Default: False

2.3.8 terminate

Terminate a DPGEN2 workflow.

```
dpigen2 terminate [-h] CONFIG ID
```

Positional Arguments

- | | |
|---------------|---------------------------------|
| CONFIG | the config file in json format. |
| ID | the ID of the workflow. |

2.3.9 stop

Stop a DPGEN2 workflow.

```
dpigen2 stop [-h] CONFIG ID
```

Positional Arguments

- | | |
|---------------|---------------------------------|
| CONFIG | the config file in json format. |
| ID | the ID of the workflow. |

2.3.10 suspend

Suspend a DPGEN2 workflow.

```
dpigen2 suspend [-h] CONFIG ID
```

Positional Arguments

- | | |
|---------------|---------------------------------|
| CONFIG | the config file in json format. |
| ID | the ID of the workflow. |

2.3.11 delete

Delete a DPGEN2 workflow.

```
dpigen2 delete [-h] CONFIG ID
```

Positional Arguments

CONFIG	the config file in json format.
ID	the ID of the workflow.

2.3.12 retry

Retry a DPGEN2 workflow.

```
dpigen2 retry [-h] CONFIG ID
```

Positional Arguments

CONFIG	the config file in json format.
ID	the ID of the workflow.

2.3.13 resume

Resume a DPGEN2 workflow.

```
dpigen2 resume [-h] CONFIG ID
```

Positional Arguments

CONFIG	the config file in json format.
ID	the ID of the workflow.

2.3.14 restart

restart a DPGEN2 workflow (for debug mode only).

```
dpigen2 restart [-h] CONFIG ID
```

Positional Arguments

CONFIG	the config file in json format.
ID	the ID of the workflow.

GUIDE ON WRITING INPUT SCRIPTS FOR DPGEN2 COMMANDS

3.1 Preliminaries

The reader of this doc is assumed to be familiar with the concurrent learning algorithm that the dpgen2 implements. If not, one may check [this paper](#).

3.2 The input script for all dpgen2 commands

For all the dpgen2 commands, one need to provide dflow2 global configurations. For example,

```
"dflow_config" : {  
    "host" : "http://address.of.the.host:port"  
},  
"dflow_s3_config" : {  
    "endpoint" : "address.of.the.s3.sever:port"  
},
```

The dpgen simply pass all keys of "dflow_config" to `dflow.config` and all keys of "dflow_s3_config" to `dflow.s3_config`.

3.3 The input script for submit and resubmit

The full documentation of the `submit` and `resubmit` script can be [found here](#). This documentation provides a fast guide on how to write the input script.

In the input script of `dpgen2 submit` and `dpgen2 resubmit`, one needs to provide the definition of the workflow and how they are executed in the input script. One may find an example input script in the dpgen2 Al-Mg alloy example.

The definition of the workflow can be provided by the following sections:

3.3.1 Inputs

This section provides the inputs to start a dpgen2 workflow. An example for the Al-Mg alloy

```
"inputs": {  
    "type_map": ["Al", "Mg"],  
    "mass_map": [27, 24],  
    "init_data_sys": [  
        "path/to/init/data/system/0",  
        "path/to/init/data/system/1"  
    ],  
}
```

The key "`init_data_sys`" provides the initial training data to kick-off the training of deep potential (DP) models.

3.3.2 Training

This section defines how a model is trained.

```
"train" : {  
    "type" : "dp",  
    "numb_models" : 4,  
    "config" : {},  
    "template_script" : "/path/to/the/template/input.json",  
    "_comment" : "all"  
}
```

The "`type` : `"dp"`" tell the traning method is "`dp`", i.e. calling DeePMD-kit to train DP models. The "`config`" key defines the training configs, see [the full documentation](#). The "`template_script`" provides the template training script in json format.

3.3.3 Exploration

This section defines how the configuration space is explored.

```
"explore" : {  
    "type" : "lmp",  
    "config" : {  
        "command": "lmp -var restart 0"  
    },  
    "convergence": {  
        "type" : "fixed-levels",  
        "conv_accuracy" : 0.9,  
        "level_f_lo": 0.05,  
        "level_f_hi": 0.50,  
        "_comment" : "all"  
    },  
    "max_numb_iter" : 5,  
    "fatal_at_max" : false,  
    "configurations": [  
        {  
            "type": "alloy",  
    }  
]
```

(continues on next page)

(continued from previous page)

```

    "lattice" : ["fcc", 4.57],
    "replicate" : [2, 2, 2],
    "numb_confs" : 30,
    "concentration" : [[1.0, 0.0], [0.5, 0.5], [0.0, 1.0]]
  },
  {
    "type" : "file",
    "prefix": "/file/prefix",
    "files" : ["relpath/to/confs/*"],
    "fmt" : "deepmd/npy"
  }
],
"stages": [
  [
    {
      "_comment" : "stage 0, task group 0",
      "type" : "lmp-md",
      "ensemble": "nvt", "nsteps": 50, "temps": [50, 100], "trj_freq": 10,
      "conf_idx": [0], "n_sample" : 3
    },
    {
      "_comment" : "stage 0, task group 1",
      "type" : "lmp-template",
      "lmp" : "template.lammps", "plm" : "template.plumed",
      "trj_freq" : 10, "revisions" : {"V_NSTEPS" : [40], "V_TEMP" : [150, ↵
      200]}, "conf_idx": [0], "n_sample" : 3
    }
  ],
  [
    {
      "_comment" : "stage 1, task group 0",
      "type" : "lmp-md",
      "ensemble": "npt", "nsteps": 50, "press": [1e0], "temps": [50, 100, ↵
      200], "trj_freq": 10,
      "conf_idx": [1], "n_sample" : 3
    }
  ]
]
}

```

The `"type" : "lmp"` means that configurations are explored by LAMMPS DPMD runs. The `"config"` key defines the lmp configs. The `"configurations"` provides the initial configurations (coordinates of atoms and the simulation cell) of the DPMD simulations. It is a list. The elements of the list are dicts that defines how the configurations are generated

- Automatic alloy configuration generator. See [the detailed doc](#) for the allowed keys.
- Configurations load from files. See [the detailed doc](#) for the allowed keys.

The `"stages"` defines the exploration stages. It is of type `list[list[dict]]`. The outer list enumerate the exploration stages, the inner list enumerate the task groups of the stage. Each dict defines a stage. See [the full documentation of the task group](#) for writting task groups.

The `"n_sample"` tells the number of configiruations randomly sampled from the set picked by `"conf_idx"` from

"*configurations*" for each exploration task. All configurations has the equal possibility to be sampled. The default value of "n_sample" is null, in this case all picked configurations are sampled. In the example, we have 3 samples for stage 0 task group 0 and 2 thermodynamic states (NVT, T=50 and 100K), then the task group has $3 \times 2 = 6$ NVT DPMD tasks.

3.3.4 FP

This section defines the first-principle (FP) calculation .

```
"fp" : {
    "type": "vasp",
    "task_max": 2,
    "run_config": {
        "command": "source /opt/intel/oneapi/setvars.sh && mpirun -n 16 vasp_std"
    },
    "inputs_config": {
        "pp_files": {"Al" : "vasp/POTCAR.Al", "Mg" : "vasp/POTCAR.Mg"},
        "kspacing": 0.32,
        "incar": "vasp/INCAR"
    }
}
```

The "*type* : "vasp" means that first-principles are VASP calculations. The "*run_config*" key defines the configs for running VASP tasks. The "*task_max*" key defines the maximal number of vasp calculations in each dpgen2 iteration. The "*pp_files*", "*kspacing*" and "*incar*" keys provides the pseudopotential files, spacing for kspace sampling and the template incar file, respectively.

3.3.5 Configuration of dflow step

The execution units of the dpgen2 are the dflow Steps. How each step is executed is defined by the "*step_configs*".

```
"step_configs": {
    "prep_train_config" : {
        "_comment" : "content omitted"
    },
    "run_train_config" : {
        "_comment" : "content omitted"
    },
    "prep_explore_config" : {
        "_comment" : "content omitted"
    },
    "run_explore_config" : {
        "_comment" : "content omitted"
    },
    "prep_fp_config" : {
        "_comment" : "content omitted"
    },
    "run_fp_config" : {
        "_comment" : "content omitted"
    },
    "select_confs_config" : {
        "_comment" : "content omitted"
    }
}
```

(continues on next page)

(continued from previous page)

```

},
"collect_data_config" : {
    "_comment" : "content omitted"
},
"cl_step_config" : {
    "_comment" : "content omitted"
},
"_comment" : "all"
},

```

The configs for prepare training, run training, prepare exploration, run exploration, prepare fp, run fp, select configurations, collect data and concurrent learning steps are given correspondingly.

Any of the config in the "*step_configs*" can be omitted. If so, the configs of the step is set to the default step configs, which is provided by the following section, for example,

```

"default_step_config" : {
    "template_config" : {
        "image" : "dpgen2:x.x.x"
    }
},

```

The way of writing the "*default_step_config*" is the same as any step config in the "*step_configs*".

CHAPTER
FOUR

ARGUMENTS OF THE SUBMIT SCRIPT

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* `dpgen2 gui`. All parameters below can be set in DP-GUI. By clicking “SAVE JSON”, one can download the input file.

dflow_config:

type: dict | NoneType, optional, default: None

argument path: `dflow_config`

The configuration passed to dflow

dflow_s3_config:

type: dict | NoneType, optional, default: None

argument path: `dflow_s3_config`

The S3 configuration passed to dflow

default_step_config:

type: dict, optional, default: {}

argument path: `default_step_config`

The default step configuration.

template_config:

type: dict, optional, default: {'image': 'dptechnology/dpgen2:latest'}

argument path: `default_step_config/template_config`

The configs passed to the PythonOPTemplate.

image:

type: str, optional, default: `dptechnology/dpgen2:latest`

argument path: `default_step_config/template_config/image`

The image to run the step.

timeout:

type: NoneType | int, optional, default: None

argument path: `default_step_config/template_config/timeout`

The time limit of the OP. Unit is second.

```
retry_on_transient_error:
    type: NoneType | int, optional, default: None
    argument path: default_step_config/template_config/
        retry_on_transient_error
    The number of retry times if a TransientError is raised.

timeout_as_transient_error:
    type: bool, optional, default: False
    argument path: default_step_config/template_config/
        timeout_as_transient_error
    Treat the timeout as TransientError.

envs:
    type: dict | NoneType, optional, default: None
    argument path: default_step_config/template_config/envs
    The environmental variables.

template_slice_config:
    type: dict, optional
    argument path: default_step_config/template_slice_config
    The configs passed to the Slices.

group_size:
    type: NoneType | int, optional, default: None
    argument path:
        default_step_config/template_slice_config/group_size
    The number of tasks running on a single node. It is efficient for a large number
    of short tasks.

pool_size:
    type: NoneType | int, optional, default: None
    argument path:
        default_step_config/template_slice_config/pool_size
    The number of tasks running at the same time on one node.

continue_on_failed:
    type: bool, optional, default: False
    argument path: default_step_config/continue_on_failed
    If continue the the step is failed (FatalError, TransientError, A certain number of retrial
    is reached...).

continue_on_num_success:
    type: NoneType | int, optional, default: None
    argument path: default_step_config/continue_on_num_success
    Only in the sliced OP case. Continue the workflow if a certain number of the sliced
    jobs are successful.
```

continue_on_success_ratio:

type: `NoneType | float`, optional, default: `None`
argument path: `default_step_config/continue_on_success_ratio`

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: `NoneType | int`, optional, default: `None`
argument path: `default_step_config/parallelism`

The parallelism for the step

executor:

type: `dict | NoneType`, optional, default: `None`
argument path: `default_step_config/executor`

The executor of the step.

Depending on the value of `type`, different sub args are accepted.

type:

type: `str` (flag key)
argument path: `default_step_config/executor/type`
possible choices: `dispatcher`

The type of the executor.

When `type` is set to `dispatcher`:

bohrium_config:

type: `dict | NoneType`, optional, default: `None`
argument path: `bohrium_config`

Configurations for the Bohrium platform.

username:

type: `str`
argument path: `bohrium_config/username`

The username of the Bohrium platform

password:

type: `str`
argument path: `bohrium_config/password`

The password of the Bohrium platform

project_id:

type: `int`
argument path: `bohrium_config/project_id`

The project ID of the Bohrium platform

host:

type: str, optional, default: `https://workflows.deepmodeling.com`
argument path: `bohrium_config/host`

The host name of the Bohrium platform. Will overwrite `dflow_config['host']`

k8s_api_server:

type: str, optional, default: `https://workflows.deepmodeling.com`
argument path: `bohrium_config/k8s_api_server`

The k8s server of the Bohrium platform. Will overwrite `dflow_config['k8s_api_server']`

repo_key:

type: str, optional, default: `oss-bohrium`
argument path: `bohrium_config/repo_key`

The repo key of the Bohrium platform. Will overwrite `dflow_s3_config['repo_key']`

storage_client:

type: str, optional, default: `dflow.plugins.bohrium.TiefblueClient`
argument path: `bohrium_config/storage_client`

The storage client of the Bohrium platform. Will overwrite `dflow_s3_config['storage_client']`

step_configs:

type: dict, optional, default: {}
argument path: `step_configs`

Configurations for executing dflow steps

prep_train_config:

type: dict, optional, default: {'template_config': {'image': 'dptechology/dpgen2:latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None}, 'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'parallelism': None, 'executor': None}
argument path: `step_configs/prep_train_config`

Configuration for prepare train

template_config:

type: dict, optional, default: {'image': 'dptechology/dpgen2:latest'}
argument path:
`step_configs/prep_train_config/template_config`

The configs passed to the PythonOPTemplate.

image:

type: str, optional, default: `dptechology/dpgen2:latest`

argument path: step_configs/prep_train_config/template_config/image
The image to run the step.

timeout:
type: NoneType | int, optional, default: None
argument path: step_configs/prep_train_config/template_config/timeout
The time limit of the OP. Unit is second.

retry_on_transient_error:
type: NoneType | int, optional, default: None
argument path: step_configs/prep_train_config/template_config/retry_on_transient_error
The number of retry times if a TransientError is raised.

timeout_as_transient_error:
type: bool, optional, default: False
argument path: step_configs/prep_train_config/template_config/timeout_as_transient_error
Treat the timeout as TransientError.

envs:
type: dict | NoneType, optional, default: None
argument path: step_configs/prep_train_config/template_config/envs
The environmental variables.

template_slice_config:
type: dict, optional
argument path:
step_configs/prep_train_config/template_slice_config
The configs passed to the Slices.

group_size:
type: NoneType | int, optional, default: None
argument path: step_configs/prep_train_config/template_slice_config/group_size
The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:
type: NoneType | int, optional, default: None
argument path: step_configs/prep_train_config/template_slice_config/pool_size
The number of tasks running at the same time on one node.

continue_on_failed:
type: bool, optional, default: False

argument path:
step_configs/prep_train_config/continue_on_failed

If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).

continue_on_num_success:

type: `NoneType | int`, optional, default: `None`
argument path: **step_configs/prep_train_config/continue_on_num_success**

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: `NoneType | float`, optional, default: `None`
argument path: **step_configs/prep_train_config/continue_on_success_ratio**

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: `NoneType | int`, optional, default: `None`
argument path: **step_configs/prep_train_config/parallelism**

The parallelism for the step

executor:

type: `dict | NoneType`, optional, default: `None`
argument path: **step_configs/prep_train_config/executor**

The executor of the step.

Depending on the value of *type*, different sub args are accepted.

type:

type: `str` (flag key)
argument path:
step_configs/prep_train_config/executor/type
possible choices: *dispatcher*

The type of the executor.

When *type* is set to *dispatcher*:

run_train_config:

type: `dict`, optional, default: `{'template_config': {'image': 'dptechnology/dpigen2:latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None}, 'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'parallelism': None, 'executor': None}`

argument path: `step_configs/run_train_config`
 Configuration for run train

template_config:

type: dict, optional, default: {'image': 'dptechology/dngen2:latest'}
 argument path:
`step_configs/run_train_config/template_config`

The configs passed to the PythonOPTemplate.

image:

type: str, optional, default:
`dptechology/dngen2:latest`
 argument path: `step_configs/run_train_config/template_config/image`

The image to run the step.

timeout:

type: NoneType | int, optional, default: None
 argument path: `step_configs/run_train_config/template_config/timeout`

The time limit of the OP. Unit is second.

retry_on_transient_error:

type: NoneType | int, optional, default: None
 argument path: `step_configs/run_train_config/template_config/retry_on_transient_error`

The number of retry times if a TransientError is raised.

timeout_as_transient_error:

type: bool, optional, default: False
 argument path: `step_configs/run_train_config/template_config/timeout_as_transient_error`

Treat the timeout as TransientError.

envs:

type: dict | NoneType, optional, default: None
 argument path: `step_configs/run_train_config/template_config/envs`

The environmental variables.

template_slice_config:

type: dict, optional
 argument path: `step_configs/run_train_config/template_slice_config`

The configs passed to the Slices.

group_size:

type: NoneType | int, optional, default: None

argument path: `step_configs/run_train_config/template_slice_config/group_size`

The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:

type: `NoneType | int`, optional, default: `None`

argument path: `step_configs/run_train_config/template_slice_config/pool_size`

The number of tasks running at the same time on one node.

continue_on_failed:

type: `bool`, optional, default: `False`

argument path:

`step_configs/run_train_config/continue_on_failed`

If continue the the step is failed (`FatalError`, `TransientError`, A certain number of retrial is reached...).

continue_on_num_success:

type: `NoneType | int`, optional, default: `None`

argument path: `step_configs/run_train_config/continue_on_num_success`

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: `NoneType | float`, optional, default: `None`

argument path: `step_configs/run_train_config/continue_on_success_ratio`

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: `NoneType | int`, optional, default: `None`

argument path:

`step_configs/run_train_config/parallelism`

The parallelism for the step

executor:

type: `dict | NoneType`, optional, default: `None`

argument path:

`step_configs/run_train_config/executor`

The executor of the step.

Depending on the value of `type`, different sub args are accepted.

```

type:
    type: str (flag key)
    argument path: step_configs/run_train_config/
    executor/type
    possible choices: dispatcher
    The type of the executor.

When type is set to dispatcher:

prep_explore_config:

    type: dict, optional, default: {'template_config': {'image':
        'dptechnology/dpgen2:latest', 'timeout': None,
        'retry_on_transient_error': None,
        'timeout_as_transient_error': False, 'envs': None},
        'continue_on_failed': False, 'continue_on_num_success':
        None, 'continue_on_success_ratio': None, 'parallelism':
        None, 'executor': None}
    argument path: step_configs/prep_explore_config
    Configuration for prepare exploration

template_config:

    type: dict, optional, default: {'image':
        'dptechnology/dpgen2:latest'}
    argument path:
        step_configs/prep_explore_config/template_config
    The configs passed to the PythonOPTemplate.

image:
    type: str, optional, default:
        dptechnology/dpgen2:latest
    argument path: step_configs/
        prep_explore_config/template_config/image
    The image to run the step.

timeout:
    type: NoneType | int, optional, default: None
    argument path:
        step_configs/prep_explore_config/
        template_config/timeout
    The time limit of the OP. Unit is second.

retry_on_transient_error:
    type: NoneType | int, optional, default: None
    argument path:
        step_configs/prep_explore_config/
        template_config/retry_on_transient_error
    The number of retry times if a TransientError is raised.

timeout_as_transient_error:
    type: bool, optional, default: False

```

```
argument path:  
step_configs/prep_explore_config/  
template_config/timeout_as_transient_error  
Treat the timeout as TransientError.  
envs:  
type: dict | NoneType, optional, default: None  
argument path: step_configs/  
prep_explore_config/template_config/envs  
The environmental variables.  
template_slice_config:  
type: dict, optional  
argument path: step_configs/prep_explore_config/  
template_slice_config  
The configs passed to the Slices.  
group_size:  
type: NoneType | int, optional, default: None  
argument path:  
step_configs/prep_explore_config/  
template_slice_config/group_size  
The number of tasks running on a single node. It is efficient for a large number of short tasks.  
pool_size:  
type: NoneType | int, optional, default: None  
argument path:  
step_configs/prep_explore_config/  
template_slice_config/pool_size  
The number of tasks running at the same time on one node.  
continue_on_failed:  
type: bool, optional, default: False  
argument path: step_configs/prep_explore_config/  
continue_on_failed  
If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).  
continue_on_num_success:  
type: NoneType | int, optional, default: None  
argument path: step_configs/prep_explore_config/  
continue_on_num_success  
Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.  
continue_on_success_ratio:  
type: NoneType | float, optional, default: None
```

argument path: `step_configs/prep_explore_config/continue_on_success_ratio`

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: `NoneType | int`, optional, default: `None`

argument path:

`step_configs/prep_explore_config/parallelism`

The parallelism for the step

executor:

type: `dict | NoneType`, optional, default: `None`

argument path:

`step_configs/prep_explore_config/executor`

The executor of the step.

Depending on the value of `type`, different sub args are accepted.

type:

type: `str` (flag key)

argument path: `step_configs/prep_explore_config/executor/type`

possible choices: `dispatcher`

The type of the executor.

When `type` is set to `dispatcher`:

run_explore_config:

type: `dict`, optional, default: `{'template_config': {'image': 'dptechology/dpgen2:latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None}, 'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'parallelism': None, 'executor': None}`

argument path: `step_configs/run_explore_config`

Configuration for run exploration

template_config:

type: `dict`, optional, default: `{'image': 'dptechology/dpgen2:latest'}`

argument path:

`step_configs/run_explore_config/template_config`

The configs passed to the PythonOPTemplate.

image:

type: `str`, optional, default:

`dptechology/dpgen2:latest`

argument path: `step_configs/run_explore_config/template_config/image`
The image to run the step.

timeout:
type: `NoneType | int`, optional, default: `None`
argument path: `step_configs/run_explore_config/template_config/timeout`
The time limit of the OP. Unit is second.

retry_on_transient_error:
type: `NoneType | int`, optional, default: `None`
argument path:
`step_configs/run_explore_config/template_config/retry_on_transient_error`
The number of retry times if a TransientError is raised.

timeout_as_transient_error:
type: `bool`, optional, default: `False`
argument path:
`step_configs/run_explore_config/template_config/timeout_as_transient_error`
Treat the timeout as TransientError.

envs:
type: `dict | NoneType`, optional, default: `None`
argument path: `step_configs/run_explore_config/template_config/envs`
The environmental variables.

template_slice_config:
type: `dict`, optional
argument path: `step_configs/run_explore_config/template_slice_config`
The configs passed to the Slices.

group_size:
type: `NoneType | int`, optional, default: `None`
argument path:
`step_configs/run_explore_config/template_slice_config/group_size`
The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:
type: `NoneType | int`, optional, default: `None`
argument path:
`step_configs/run_explore_config/template_slice_config/pool_size`
The number of tasks running at the same time on one node.

continue_on_failed:

type: bool, optional, default: False
 argument path: step_configs/run_explore_config/
 continue_on_failed

If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).

continue_on_num_success:

type: NoneType | int, optional, default: None
 argument path: step_configs/run_explore_config/
 continue_on_num_success

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: NoneType | float, optional, default: None
 argument path: step_configs/run_explore_config/
 continue_on_success_ratio

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: NoneType | int, optional, default: None
 argument path:
 step_configs/run_explore_config/parallelism

The parallelism for the step

executor:

type: dict | NoneType, optional, default: None
 argument path:
 step_configs/run_explore_config/executor

The executor of the step.

Depending on the value of *type*, different sub args are accepted.

type:

type: str (flag key)
 argument path: step_configs/
 run_explore_config/executor/type
 possible choices: *dispatcher*

The type of the executor.

When *type* is set to *dispatcher*:

prep_fp_config:

type: dict, optional, default: {'template_config': {'image': 'dptechology/dpgen2:latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None},

```
'continue_on_failed': False, 'continue_on_num_success':  
None, 'continue_on_success_ratio': None, 'parallelism':  
None, 'executor': None}  
argument path: step_configs/prep_fp_config  
Configuration for prepare fp  
template_config:  
type: dict, optional, default: {'image':  
'dptechology/dpgen2:latest'}  
argument path:  
step_configs/prep_fp_config/template_config  
The configs passed to the PythonOPTemplate.  
image:  
type: str, optional, default:  
dptechology/dpgen2:latest  
argument path: step_configs/prep_fp_config/  
template_config/image  
The image to run the step.  
timeout:  
type: NoneType | int, optional, default: None  
argument path: step_configs/prep_fp_config/  
template_config/timeout  
The time limit of the OP. Unit is second.  
retry_on_transient_error:  
type: NoneType | int, optional, default: None  
argument path: step_configs/prep_fp_config/  
template_config/retry_on_transient_error  
The number of retry times if a TransientError is raised.  
timeout_as_transient_error:  
type: bool, optional, default: False  
argument path: step_configs/prep_fp_config/  
template_config/timeout_as_transient_error  
Treat the timeout as TransientError.  
envs:  
type: dict | NoneType, optional, default: None  
argument path: step_configs/prep_fp_config/  
template_config/envs  
The environmental variables.  
template_slice_config:  
type: dict, optional  
argument path:  
step_configs/prep_fp_config/template_slice_config  
The configs passed to the Slices.
```

group_size:

type: NoneType | int, optional, default: None
 argument path: step_configs/prep_fp_config/
 template_slice_config/group_size

The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:

type: NoneType | int, optional, default: None
 argument path: step_configs/prep_fp_config/
 template_slice_config/pool_size

The number of tasks running at the same time on one node.

continue_on_failed:

type: bool, optional, default: False
 argument path:
 step_configs/prep_fp_config/continue_on_failed

If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).

continue_on_num_success:

type: NoneType | int, optional, default: None
 argument path: step_configs/prep_fp_config/
 continue_on_num_success

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: NoneType | float, optional, default: None
 argument path: step_configs/prep_fp_config/
 continue_on_success_ratio

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: NoneType | int, optional, default: None
 argument path:
 step_configs/prep_fp_config/parallelism

The parallelism for the step

executor:

type: dict | NoneType, optional, default: None
 argument path: step_configs/prep_fp_config/executor

The executor of the step.

Depending on the value of *type*, different sub args are accepted.

type:
type: str (flag key)
argument path:
step_configs/prep_fp_config/executor/type
possible choices: *dispatcher*

The type of the executor.

When `type` is set to *dispatcher*:

run_fp_config:

type: dict, optional, default: {'template_config': {'image': 'dptechology/dpge2/latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None}, 'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'parallelism': None, 'executor': None}
argument path: step_configs/run_fp_config

Configuration for run fp

template_config:

type: dict, optional, default: {'image': 'dptechology/dpge2/latest'}
argument path:
step_configs/run_fp_config/template_config

The configs passed to the PythonOPTemplate.

image:

type: str, optional, default:
dptechology/dpge2/latest
argument path: step_configs/run_fp_config/
template_config/image

The image to run the step.

timeout:

type: NoneType | int, optional, default: None
argument path: step_configs/run_fp_config/
template_config/timeout

The time limit of the OP. Unit is second.

retry_on_transient_error:

type: NoneType | int, optional, default: None
argument path: step_configs/run_fp_config/
template_config/retry_on_transient_error

The number of retry times if a TransientError is raised.

timeout_as_transient_error:

type: bool, optional, default: False
argument path: step_configs/run_fp_config/
template_config/timeout_as_transient_error

Treat the timeout as TransientError.

envs:

type: dict | NoneType, optional, default: None
argument path: step_configs/run_fp_config/
template_config/envs

The environmental variables.

template_slice_config:

type: dict, optional
argument path:
step_configs/run_fp_config/template_slice_config

The configs passed to the Slices.

group_size:

type: NoneType | int, optional, default: None
argument path: step_configs/run_fp_config/
template_slice_config/group_size

The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:

type: NoneType | int, optional, default: None
argument path: step_configs/run_fp_config/
template_slice_config/pool_size

The number of tasks running at the same time on one node.

continue_on_failed:

type: bool, optional, default: False
argument path:
step_configs/run_fp_config/continue_on_failed

If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).

continue_on_num_success:

type: NoneType | int, optional, default: None
argument path: step_configs/run_fp_config/
continue_on_num_success

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: NoneType | float, optional, default: None
argument path: step_configs/run_fp_config/
continue_on_success_ratio

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: `NoneType` | `int`, optional, default: `None`
argument path:
`step_configs/run_fp_config/parallelism`

The parallelism for the step

executor:

type: `dict` | `NoneType`, optional, default: `None`
argument path: `step_configs/run_fp_config/executor`
The executor of the step.

Depending on the value of `type`, different sub args are accepted.

type:

type: `str` (flag key)
argument path:
`step_configs/run_fp_config/executor/type`
possible choices: `dispatcher`

The type of the executor.

When `type` is set to `dispatcher`:

select_confs_config:

type: `dict`, optional, default: `{'template_config': {'image': 'dptechnology/dpigen2:latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None}, 'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'parallelism': None, 'executor': None}`
argument path: `step_configs/select_confs_config`

Configuration for the select confs

template_config:

type: `dict`, optional, default: `{'image': 'dptechnology/dpigen2:latest'}`
argument path:
`step_configs/select_confs_config/template_config`

The configs passed to the PythonOPTemplate.

image:

type: `str`, optional, default:
`dptechnology/dpigen2:latest`
argument path: `step_configs/select_confs_config/template_config/image`

The image to run the step.

timeout:

type: `NoneType` | `int`, optional, default: `None`

argument path:
step_configs/select_confs_config/
template_config/timeout

The time limit of the OP. Unit is second.

retry_on_transient_error:

type: NoneType | int, optional, default: None
argument path:
step_configs/select_confs_config/
template_config/retry_on_transient_error

The number of retry times if a TransientError is raised.

timeout_as_transient_error:

type: bool, optional, default: False
argument path:
step_configs/select_confs_config/
template_config/timeout_as_transient_error

Treat the timeout as TransientError.

envs:

type: dict | NoneType, optional, default: None
argument path: **step_configs/**
select_confs_config/template_config/envs

The environmental variables.

template_slice_config:

type: dict, optional
argument path: **step_configs/select_confs_config/**
template_slice_config

The configs passed to the Slices.

group_size:

type: NoneType | int, optional, default: None
argument path:
step_configs/select_confs_config/
template_slice_config/group_size

The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:

type: NoneType | int, optional, default: None
argument path:
step_configs/select_confs_config/
template_slice_config/pool_size

The number of tasks running at the same time on one node.

continue_on_failed:

type: bool, optional, default: False

argument path: `step_configs/select_confs_config/continue_on_failed`

If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).

continue_on_num_success:

type: `NoneType | int`, optional, default: `None`

argument path: `step_configs/select_confs_config/continue_on_num_success`

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: `NoneType | float`, optional, default: `None`

argument path: `step_configs/select_confs_config/continue_on_success_ratio`

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: `NoneType | int`, optional, default: `None`

argument path:

`step_configs/select_confs_config/parallelism`

The parallelism for the step

executor:

type: `dict | NoneType`, optional, default: `None`

argument path:

`step_configs/select_confs_config/executor`

The executor of the step.

Depending on the value of `type`, different sub args are accepted.

type:

type: `str` (flag key)

argument path: `step_configs/`

`select_confs_config/executor/type`

possible choices: `dispatcher`

The type of the executor.

When `type` is set to `dispatcher`:

collect_data_config:

```
type: dict, optional, default: {'template_config': {'image': 'dptechology/dpgen2:latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None}, 'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'parallelism': None, 'executor': None}
```

argument path: `step_configs/collect_data_config`
 Configuration for the collect data

template_config:

type: dict, optional, default: {'image':
 'dptechology/dpgen2:latest'}

argument path:
`step_configs/collect_data_config/template_config`

The configs passed to the PythonOPTemplate.

image:

type: str, optional, default:
 dptechology/dpgen2:latest

argument path: `step_configs/collect_data_config/template_config/image`

The image to run the step.

timeout:

type: NoneType | int, optional, default: None
 argument path:
`step_configs/collect_data_config/template_config/timeout`

The time limit of the OP. Unit is second.

retry_on_transient_error:

type: NoneType | int, optional, default: None
 argument path:
`step_configs/collect_data_config/template_config/retry_on_transient_error`

The number of retry times if a TransientError is raised.

timeout_as_transient_error:

type: bool, optional, default: False
 argument path:
`step_configs/collect_data_config/template_config/timeout_as_transient_error`

Treat the timeout as TransientError.

envs:

type: dict | NoneType, optional, default: None
 argument path: `step_configs/collect_data_config/template_config/envs`

The environmental variables.

template_slice_config:

type: dict, optional
 argument path: `step_configs/collect_data_config/template_slice_config`

The configs passed to the Slices.

group_size:

type: NoneType | int, optional, default: None
argument path:
`step_configs/collect_data_config/`
`template_slice_config/group_size`

The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:

type: NoneType | int, optional, default: None
argument path:
`step_configs/collect_data_config/`
`template_slice_config/pool_size`

The number of tasks running at the same time on one node.

continue_on_failed:

type: bool, optional, default: False
argument path: `step_configs/collect_data_config/`
`continue_on_failed`

If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).

continue_on_num_success:

type: NoneType | int, optional, default: None
argument path: `step_configs/collect_data_config/`
`continue_on_num_success`

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: NoneType | float, optional, default: None
argument path: `step_configs/collect_data_config/`
`continue_on_success_ratio`

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: NoneType | int, optional, default: None
argument path:
`step_configs/collect_data_config/parallelism`

The parallelism for the step

executor:

type: dict | NoneType, optional, default: None
argument path:
`step_configs/collect_data_config/executor`

The executor of the step.

Depending on the value of *type*, different sub args are accepted.

type:

type: str (flag key)
 argument path: step_configs/
 collect_data_config/executor/type
 possible choices: *dispatcher*

The type of the executor.

When *type* is set to *dispatcher*:

cl_step_config:

type: dict, optional, default: {'template_config': {'image': 'dptechology/dpge2:latest', 'timeout': None, 'retry_on_transient_error': None, 'timeout_as_transient_error': False, 'envs': None}, 'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'parallelism': None, 'executor': None}
 argument path: step_configs/cl_step_config

Configuration for the concurrent learning step

template_config:

type: dict, optional, default: {'image': 'dptechology/dpge2:latest'}
 argument path:
 step_configs/cl_step_config/template_config

The configs passed to the PythonOPTemplate.

image:

type: str, optional, default:
 dptechology/dpge2:latest
 argument path: step_configs/cl_step_config/
 template_config/image

The image to run the step.

timeout:

type: NoneType | int, optional, default: None
 argument path: step_configs/cl_step_config/
 template_config/timeout

The time limit of the OP. Unit is second.

retry_on_transient_error:

type: NoneType | int, optional, default: None
 argument path: step_configs/cl_step_config/
 template_config/retry_on_transient_error

The number of retry times if a TransientError is raised.

timeout_as_transient_error:

type: bool, optional, default: False

argument path: `step_configs/cl_step_config/template_config/timeout_as_transient_error`

Treat the timeout as TransientError.

envs:

type: `dict | NoneType`, optional, default: `None`
argument path: `step_configs/cl_step_config/template_config/envs`

The environmental variables.

template_slice_config:

type: `dict`, optional

argument path:
`step_configs/cl_step_config/template_slice_config`

The configs passed to the Slices.

group_size:

type: `NoneType | int`, optional, default: `None`
argument path: `step_configs/cl_step_config/template_slice_config/group_size`

The number of tasks running on a single node. It is efficient for a large number of short tasks.

pool_size:

type: `NoneType | int`, optional, default: `None`
argument path: `step_configs/cl_step_config/template_slice_config/pool_size`

The number of tasks running at the same time on one node.

continue_on_failed:

type: `bool`, optional, default: `False`

argument path:
`step_configs/cl_step_config/continue_on_failed`

If continue the the step is failed (FatalError, TransientError, A certain number of retrial is reached...).

continue_on_num_success:

type: `NoneType | int`, optional, default: `None`

argument path: `step_configs/cl_step_config/continue_on_num_success`

Only in the sliced OP case. Continue the workflow if a certain number of the sliced jobs are successful.

continue_on_success_ratio:

type: `NoneType | float`, optional, default: `None`

argument path: `step_configs/cl_step_config/continue_on_success_ratio`

Only in the sliced OP case. Continue the workflow if a certain ratio of the sliced jobs are successful.

parallelism:

type: `NoneType | int`, optional, default: `None`
 argument path:
`step_configs/cl_step_config/parallelism`

The parallelism for the step

executor:

type: `dict | NoneType`, optional, default: `None`
 argument path: `step_configs/cl_step_config/executor`
 The executor of the step.

Depending on the value of `type`, different sub args are accepted.

type:

type: `str` (flag key)
 argument path:
`step_configs/cl_step_config/executor/type`
 possible choices: `dispatcher`

The type of the executor.

When `type` is set to `dispatcher`:

upload_python_packages:

type: `typing.List[str] | str | NoneType`, optional, default: `None`, alias:
`upload_python_package`
 argument path: `upload_python_packages`
 Upload python package, for debug purpose

inputs:

type: `dict`
 argument path: `inputs`
 The input parameter and artifacts for dpgen2

type_map:

type: `typing.List[str]`
 argument path: `inputs/type_map`
 The type map. e.g. `["Al", "Mg"]`. Al and Mg will have type 0 and 1, respectively.

mass_map:

type: `typing.List[float]`
 argument path: `inputs/mass_map`
 The mass map. e.g. `[27., 24.]`. Al and Mg will be set with mass 27. and 24. amu, respectively.

init_data_prefix:

type: `str | NoneType`, optional, default: `None`

argument path: inputs/init_data_prefix

The prefix of initial data systems

mixed_type:

type: bool, optional, default: False

argument path: inputs/mixed_type

Use *deepmd/npy/mixed* format for storing training data.

do_finetune:

type: bool, optional, default: False

argument path: inputs/do_finetune

Finetune the pretrained model before the first iteration. If it is set to True, then an additional step, finetune-step, which is based on a branch of “PrepRunDPTrain,” will be added before the dpgen_step. In the finetune-step, the internal flag finetune_mode is set to “finetune,” which means SuperOP “PrepRunDPTrain” is now used as the “Finetune.” In this step, we finetune the pretrained model in the train step and modify the template after training. After that, in the normal dpgen-step, the flag do_finetune is set as “train-init,” which means we use *-init-frz-model* to train based on models from the previous iteration. The “do_finetune” flag is set to False by default, while the internal flag finetune_mode is set to “no,” which means anything related to finetuning will not be done.

init_data_sys:

type: typing.List[str] | str | NoneType, optional, default: None

argument path: inputs/init_data_sys

The initial data systems

multitask:

type: bool, optional, default: False

argument path: inputs/multitask

Do multitask training

head:

type: str | NoneType, optional, default: None

argument path: inputs/head

Head to use in the multitask training

multi_init_data:

type: dict | NoneType, optional, default: None

argument path: inputs/multi_init_data

The initial data for multitask, it should be a dict, whose keys are task names and each value is a dict containing fields *prefix* and *sys* for initial data of each task

valid_data_prefix:

type: str | NoneType, optional, default: None

argument path: inputs/valid_data_prefix

The prefix of validation data systems

valid_data_sys:

type: typing.List[str] | str | NoneType, optional, default: None
 argument path: inputs/valid_data_sys

The validation data systems

train:

type: dict
 argument path: train

The configuration for training

Depending on the value of *type*, different sub args are accepted.

type:

type: str (flag key)
 argument path: train/type
 possible choices: *dp*, *dp-dist*

the type of the training

When *type* is set to *dp*:

config:

type: dict, optional, default: {'impl': 'tensorflow',
 'init_model_policy': 'no', 'init_model_old_ratio': 0.9,
 'init_model_numb_steps': 400000, 'init_model_start_lr':
 0.0001, 'init_model_start_pref_e': 0.1,
 'init_model_start_pref_f': 100, 'init_model_start_pref_v':
 0.0, 'init_model_with_finetune': False, 'finetune_args':
 '', 'multitask': False, 'head': None,
 'multi_init_data_idx': None}
 argument path: train[dp]/config

Number of models trained for evaluating the model deviation

impl:

type: str, optional, default: tensorflow, alias: *backend*
 argument path: train[dp]/config/impl

The implementation/backend of DP. It can be ‘tensorflow’ or ‘pytorch’. ‘tensorflow’ for default.

init_model_policy:

type: str, optional, default: no
 argument path: train[dp]/config/init_model_policy

The policy of init-model training. It can be

- ‘no’: No init-model training. Training from scratch.
- ‘yes’: Do init-model training.
- ‘old_data_larger_than:XXX’: Do init-model if the training data size of the previous model is larger than XXX. XXX is an int number.

init_model_old_ratio:

type: float, optional, default: 0.9
argument path: train[dp]/config/init_model_old_ratio

The frequency ratio of old data over new data

init_model_numb_steps:

type: int, optional, default: 400000, alias:
init_model_stop_batch
argument path:
train[dp]/config/init_model_numb_steps

The number of training steps when init-model

init_model_start_lr:

type: float, optional, default: 0.0001
argument path: train[dp]/config/init_model_start_lr

The start learning rate when init-model

init_model_start_pref_e:

type: float, optional, default: 0.1
argument path:
train[dp]/config/init_model_start_pref_e

The start energy prefactor in loss when init-model

init_model_start_pref_f:

type: float, optional, default: 100
argument path:
train[dp]/config/init_model_start_pref_f

The start force prefactor in loss when init-model

init_model_start_pref_v:

type: float, optional, default: 0.0
argument path:
train[dp]/config/init_model_start_pref_v

The start virial prefactor in loss when init-model

init_model_with_finetune:

type: bool, optional, default: False
argument path:
train[dp]/config/init_model_with_finetune

Use finetune for init model

finetune_args:

type: str, optional, default: (empty string)
argument path: train[dp]/config/finetune_args
Extra arguments for finetuning

multitask:

type: bool, optional, default: False
 argument path: train[dp]/config/multitask
 Do multitask training

head:

type: str | NoneType, optional, default: None
 argument path: train[dp]/config/head
 Head to use in the multitask training

multi_init_data_idx:

type: dict | NoneType, optional, default: None
 argument path: train[dp]/config/multi_init_data_idx
 A dict mapping from task name to list of indices in the init data

numb_models:

type: int, optional, default: 4
 argument path: train[dp]/numb_models
 Number of models trained for evaluating the model deviation

template_script:

type: typing.List[str] | str
 argument path: train[dp]/template_script
 File names of the template training script. It can be a *List[str]*, the length of which is the same as *numb_models*. Each template script in the list is used to train a model. Can be a *str*, the models share the same template training script.

init_models_paths:

type: typing.List[str] | NoneType, optional, default: None, alias:
`training_iter0_model_path`
 argument path: train[dp]/init_models_paths
 the paths to initial models

When type is set to dp-dist:

config:

type: dict, optional, default: {'impl': 'tensorflow',
 'init_model_policy': 'no', 'init_model_old_ratio': 0.9,
 'init_model_numb_steps': 400000, 'init_model_start_lr':
 0.0001, 'init_model_start_pref_e': 0.1,
 'init_model_start_pref_f': 100, 'init_model_start_pref_v':
 0.0, 'init_model_with_finetune': False, 'finetune_args':
 '', 'multitask': False, 'head': None,
 'multi_init_data_idx': None}
 argument path: train[dp-dist]/config
 Configuration of training

impl:

type: str, optional, default: tensorflow, alias: *backend*
argument path: train[dp-dist]/config/impl

The implementation/backend of DP. It can be ‘tensorflow’ or ‘pytorch’. ‘tensorflow’ for default.

init_model_policy:

type: str, optional, default: no
argument path:
train[dp-dist]/config/init_model_policy

The policy of init-model training. It can be

- ‘no’: No init-model training. Training from scratch.
- ‘yes’: Do init-model training.
- ‘old_data_larger_than:XXX’: Do init-model if the training data size of the previous model is larger than XXX. XXX is an int number.

init_model_old_ratio:

type: float, optional, default: 0.9
argument path:
train[dp-dist]/config/init_model_old_ratio

The frequency ratio of old data over new data

init_model_numb_steps:

type: int, optional, default: 400000, alias:
init_model_stop_batch
argument path:
train[dp-dist]/config/init_model_numb_steps

The number of training steps when init-model

init_model_start_lr:

type: float, optional, default: 0.0001
argument path:
train[dp-dist]/config/init_model_start_lr

The start learning rate when init-model

init_model_start_pref_e:

type: float, optional, default: 0.1
argument path:
train[dp-dist]/config/init_model_start_pref_e

The start energy prefactor in loss when init-model

init_model_start_pref_f:

type: float, optional, default: 100

argument path:
`train[dp-dist]/config/init_model_start_pref_f`

The start force prefactor in loss when init-model

init_model_start_pref_v:

type: float, optional, default: 0.0

argument path:
`train[dp-dist]/config/init_model_start_pref_v`

The start virial prefactor in loss when init-model

init_model_with_finetune:

type: bool, optional, default: False

argument path:
`train[dp-dist]/config/init_model_with_finetune`

Use finetune for init model

finetune_args:

type: str, optional, default: (empty string)

argument path: `train[dp-dist]/config/finetune_args`

Extra arguments for finetuning

multitask:

type: bool, optional, default: False

argument path: `train[dp-dist]/config/multitask`

Do multitask training

head:

type: str | NoneType, optional, default: None

argument path: `train[dp-dist]/config/head`

Head to use in the multitask training

multi_init_data_idx:

type: dict | NoneType, optional, default: None

argument path:
`train[dp-dist]/config/multi_init_data_idx`

A dict mapping from task name to list of indices in the init data

template_script:

type: typing.List[str] | str

argument path: `train[dp-dist]/template_script`

File names of the template training script. It can be a *List[str]*, the length of which is the same as *numb_models*. Each template script in the list is used to train a model. Can be a *str*, the models share the same template training script.

student_model_path:

type: str, optional

argument path: `train[dp-dist]/student_model_path`

The path of student model

explore:

type: dict

argument path: `explore`

The configuration for exploration

Depending on the value of `type`, different sub args are accepted.

type:

type: str (flag key)

argument path: `explore/type`

possible choices: `lmp, calypso`

The type of the exploration

When `type` is set to `lmp`:

The exploration by LAMMPS simulations

config:

type: dict, optional, default: { 'command': 'lmp',
'teacher_model_path': None, 'shuffle_models': False,
'head': None}

argument path: `explore[lmp]/config`

Configuration of lmp exploration

command:

type: str, optional, default: `lmp`

argument path: `explore[lmp]/config/command`

The command of LAMMPS

teacher_model_path:

type: str | BinaryFileInput | NoneType, optional, default:
None

argument path:

`explore[lmp]/config/teacher_model_path`

The teacher model in *Knowledge Distillation*

shuffle_models:

type: bool, optional, default: False

argument path: `explore[lmp]/config/shuffle_models`

Randomly pick a model from the group of models to drive the exploration MD simulation

head:

type: str | NoneType, optional, default: None

argument path: `explore[lmp]/config/head`

Select a head from multitask

max_numb_iter:

type: int, optional, default: 10
 argument path: explore[lmp]/max_numb_iter
 Maximum number of iterations per stage

fatal_at_max:

type: bool, optional, default: True
 argument path: explore[lmp]/fatal_at_max
 Fatal when the number of iteration per stage reaches the *max_numb_iter*

output_nopbc:

type: bool, optional, default: False
 argument path: explore[lmp]/output_nopbc
 Remove pbc of the output configurations

convergence:

type: dict
 argument path: explore[lmp]/convergence
 The method of convergence check.

Depending on the value of *type*, different sub args are accepted.

type:

type: str (flag key)
 argument path: explore[lmp]/convergence/type
 possible choices: *fixed-levels*,
fixed-levels-max-select, *adaptive-lower*
 the type of the condidate selection and convergence check method.

When *type* is set to *fixed-levels*:

The configurations with force model deviation between *level_f_lo*, *level_f_hi* or virial model deviation between *level_v_lo* and *level_v_hi* are treated as candidates (The virial model deviation check is optional). The configurations will be randomly sampled from candidates for FP calculations. If the ratio of accurate (below *level_f_lo* and *level_v_lo*) is higher then *conv_accuracy*, the stage is treated as converged.

level_f_lo:

type: float
 argument path:
 explore[lmp]/convergence[fixed-levels]/level_f_lo

The lower trust level of force model deviation

level_f_hi:

type: float
 argument path:
 explore[lmp]/convergence[fixed-levels]/level_f_hi

The higher trust level of force model deviation

level_v_lo:

type: `NoneType | float`, optional, default: `None`
argument path:
`explore[lmp]/convergence[fixed-levels]/level_v_lo`

The lower trust level of virial model deviation

level_v_hi:

type: `NoneType | float`, optional, default: `None`
argument path:
`explore[lmp]/convergence[fixed-levels]/level_v_hi`

The higher trust level of virial model deviation

conv_accuracy:

type: `float`, optional, default: `0.9`
argument path: `explore[lmp]/convergence[fixed-levels]/conv_accuracy`

If the ratio of accurate frames is larger than this value, the stage is converged

When `type` is set to `fixed-levels-max-select`:

The configurations with force model deviation between `level_f_lo`, `level_f_hi` or virial model deviation between `level_v_lo` and `level_v_hi` are treated as candidates (The virial model deviation check is optional). The configurations with maximal model deviation in the candidates are sent for FP calculations. If the ratio of accurate (below `level_f_lo` and `level_v_lo`) is higher than `conv_accuracy`, the stage is treated as converged.

level_f_lo:

type: `float`
argument path: `explore[lmp]/convergence[fixed-levels-max-select]/level_f_lo`

The lower trust level of force model deviation

level_f_hi:

type: `float`
argument path: `explore[lmp]/convergence[fixed-levels-max-select]/level_f_hi`

The higher trust level of force model deviation

level_v_lo:

type: `NoneType | float`, optional, default: `None`
argument path: `explore[lmp]/convergence[fixed-levels-max-select]/level_v_lo`

The lower trust level of virial model deviation

level_v_hi:

type: `NoneType | float`, optional, default: `None`

argument path: `explore[lmp]/convergence[fixed-levels-max-select]/level_v_hi`

The higher trust level of virial model deviation

conv_accuracy:

type: `float`, optional, default: `0.9`

argument path: `explore[lmp]/convergence[fixed-levels-max-select]/conv_accuracy`

If the ratio of accurate frames is larger than this value, the stage is converged

When `type` is set to `adaptive-lower`:

The method of adaptive adjust the lower trust levels. In each step of iterations, a number (set by `numb_candi_f` or `numb_candi_v`) or a ratio (set by `rate_candi_f` or `rate_candi_v`) of configurations with a model deviation lower than the higher trust level (`level_f_hi`, `level_v_hi`) are treated as candidates. The lowest model deviation of the candidates are treated as the lower trust level. If the lower trust level does not change significant (controlled by `conv_tolerance`) in `n_checked_steps`, the stage is treated as converged.

level_f_hi:

type: `float`, optional, default: `0.5`

argument path: `explore[lmp]/convergence[adaptive-lower]/level_f_hi`

The higher trust level of force model deviation

numb_candi_f:

type: `int`, optional, default: `200`

argument path: `explore[lmp]/convergence[adaptive-lower]/numb_candi_f`

The number of force frames that has a model deviation lower than `level_f_hi` treated as candidate.

rate_candi_f:

type: `float`, optional, default: `0.01`

argument path: `explore[lmp]/convergence[adaptive-lower]/rate_candi_f`

The ratio of force frames that has a model deviation lower than `level_f_hi` treated as candidate.

level_v_hi:

type: `NoneType | float`, optional, default: `None`

argument path: `explore[lmp]/convergence[adaptive-lower]/level_v_hi`

The higher trust level of virial model deviation

numb_candi_v:

type: int, optional, default: 0
argument path: explore[lmp]/
convergence[adaptive-lower]/numb_candi_v

The number of virial frames that has a model deviation lower than
level_v_hi treated as candidate.

rate_candi_v:

type: float, optional, default: 0.0
argument path: explore[lmp]/
convergence[adaptive-lower]/rate_candi_v

The ratio of virial frames that has a model deviation lower than
level_v_hi treated as candidate.

n_checked_steps:

type: int, optional, default: 2
argument path: explore[lmp]/
convergence[adaptive-lower]/n_checked_steps

The number of steps to check the convergence.

conv_tolerance:

type: float, optional, default: 0.05
argument path: explore[lmp]/
convergence[adaptive-lower]/conv_tolerance

The convergence tolerance.

candi_sel_prob:

type: str, optional, default: uniform
argument path: explore[lmp]/
convergence[adaptive-lower]/candi_sel_prob

The method for selecting candidates. It can be ‘uniform’: all candidates are of the same probability. ‘inv_pop_f’ or ‘inv_pop_f:nhist’: the probability is inversely proportional to the population of a histogram between *leven_f_lo* and *level_f_hi*. The number of bins in the histogram is set by *nhist*, which should be an integer. The default is 10.

configurations:

type: list, alias: *configuration*
argument path: explore[lmp]/configurations

A list of initial configurations.

This argument takes a list with each element containing the following:

Depending on the value of *type*, different sub args are accepted.

type:

type: str (flag key)
argument path: explore[lmp]/configurations/type

possible choices: `alloy`, `file`

the type of the initial configuration generator.

When `type` is set to `alloy`:

Generate alloys with *a certain lattice or user provided structure*, the elements randomly occupying the lattice with *user provided probability*.

numb_confs:

type: `int`, optional, default: 1

argument path:

`explore[lmp]/configurations[alloy]/numb_confs`

The number of configurations to generate

lattice:

type: `list` | `tuple`

argument path:

`explore[lmp]/configurations[alloy]/lattice`

The lattice. Should be a list providing [“lattice_type”, lattice_const], or a list providing [“/path/to/dpdata/system”, “fmt”]. The two styles are distinguished by the type of the second element. Currently “lattice_type” can be “bcc”, “fcc”, “hcp”, “sc” or “diamond”.

replicate:

type: `list` | `NoneType`, optional, default: `None`

argument path:

`explore[lmp]/configurations[alloy]/replicate`

The number of replicates in each direction

concentration:

type: `list` | `NoneType`, optional, default: `None`

argument path:

`explore[lmp]/configurations[alloy]/concentration`

The concentration of each element. *List[List[float]]* or *List[float]* or *None*. If *List[float]*, the concentrations of each element. The length of the list should be the same as the `type_map`. If *List[List[float]]*, a list of concentrations (*List[float]*) is randomly picked from the List. If *None*, the elements are assumed to be of equal concentration.

cell_pert_frac:

type: `float`, optional, default: `0.0`

argument path:

`explore[lmp]/configurations[alloy]/cell_pert_frac`

The fraction of cell perturbation

atom_pert_dist:

type: `float`, optional, default: `0.0`

argument path:
explore[lmp]/configurations[alloy]/atom_pert_dist

The distance of atomic position perturbation

When `type` is set to `file`:

Generate alloys from user provided file(s). The file(s) are assumed to be load by `dodata`.

files:

type: str | list

argument path:

explore[lmp]/configurations[file]/files

The paths to the configuration files. wildcards are supported.

prefix:

type: str | NoneType, optional, default: None

argument path:

explore[lmp]/configurations[file]/prefix

The prefix of file paths.

fmt:

type: str, optional, default: auto

argument path: explore[lmp]/configurations[file]/fmt

The format (dodata accepted formats) of the files.

remove_pbc:

type: bool, optional, default: False

argument path:

explore[lmp]/configurations[file]/remove_pbc

The remove the pbc of the data. shift the coords to the center of box so it can be used with lammps.

stages:

type: typing.List[typing.List[dict]]

argument path: explore[lmp]/stages

The definition of exploration stages of type `List[List[ExplorationTaskGroup]]`. The outer list provides the enumeration of the exploration stages. Then each stage is defined by a list of exploration task groups. Each task group is described in [the task group definition](#)

When `type` is set to `calypso`:

The exploration by CALYPSO structure prediction

config:

```
type: dict, optional, default: {'command': 'lmp',
'teacher_model_path': None, 'shuffle_models': False,
'head': None}
```

argument path: `explore[calypso]/config`

Configuration of lmp exploration

command:

type: `str`, optional, default: `lmp`

argument path: `explore[calypso]/config/command`

The command of LAMMPS

teacher_model_path:

type: `str | BinaryFileInput | NoneType`, optional, default: `None`

argument path:

`explore[calypso]/config/teacher_model_path`

The teacher model in *Knowledge Distillation*

shuffle_models:

type: `bool`, optional, default: `False`

argument path:

`explore[calypso]/config/shuffle_models`

Randomly pick a model from the group of models to drive the exploration MD simulation

head:

type: `str | NoneType`, optional, default: `None`

argument path: `explore[calypso]/config/head`

Select a head from multitask

max_numb_iter:

type: `int`, optional, default: `10`

argument path: `explore[calypso]/max_numb_iter`

Maximum number of iterations per stage

fatal_at_max:

type: `bool`, optional, default: `True`

argument path: `explore[calypso]/fatal_at_max`

Fatal when the number of iteration per stage reaches the `max_numb_iter`

output_nopbc:

type: `bool`, optional, default: `False`

argument path: `explore[calypso]/output_nopbc`

Remove pbc of the output configurations

convergence:

type: `dict`

argument path: `explore[calypso]/convergence`

The method of convergence check.

Depending on the value of `type`, different sub args are accepted.

type:

type: `str` (flag key)

argument path: `explore[calypso]/convergence/type`

possible choices: `fixed-levels`,

`fixed-levels-max-select`, `adaptive-lower`

the type of the candidate selection and convergence check method.

When `type` is set to `fixed-levels`:

The configurations with force model deviation between `level_f_lo`, `level_f_hi` or virial model deviation between `level_v_lo` and `level_v_hi` are treated as candidates (The virial model deviation check is optional). The configurations will be randomly sampled from candidates for FP calculations. If the ratio of accurate (below `level_f_lo` and `level_v_lo`) is higher than `conv_accuracy`, the stage is treated as converged.

level_f_lo:

type: `float`

argument path: `explore[calypso]/convergence[fixed-levels]/level_f_lo`

The lower trust level of force model deviation

level_f_hi:

type: `float`

argument path: `explore[calypso]/convergence[fixed-levels]/level_f_hi`

The higher trust level of force model deviation

level_v_lo:

type: `NoneType | float`, optional, default: `None`

argument path: `explore[calypso]/convergence[fixed-levels]/level_v_lo`

The lower trust level of virial model deviation

level_v_hi:

type: `NoneType | float`, optional, default: `None`

argument path: `explore[calypso]/convergence[fixed-levels]/level_v_hi`

The higher trust level of virial model deviation

conv_accuracy:

type: `float`, optional, default: `0.9`

argument path: `explore[calypso]/convergence[fixed-levels]/conv_accuracy`

If the ratio of accurate frames is larger than this value, the stage is converged

When `type` is set to `fixed-levels-max-select`:

The configurations with force model deviation between `level_f_lo`, `level_f_hi` or virial model deviation between `level_v_lo` and `level_v_hi` are treated as candidates (The virial model deviation check is optional). The configurations with maximal model deviation in the candidates are sent for FP calculations. If the ratio of accurate (below `level_f_lo` and `level_v_lo`) is higher than `conv_accuracy`, the stage is treated as converged.

`level_f_lo`:

type: float
argument path: `explore[calypso]/convergence[fixed-levels-max-select]/level_f_lo`

The lower trust level of force model deviation

`level_f_hi`:

type: float
argument path: `explore[calypso]/convergence[fixed-levels-max-select]/level_f_hi`

The higher trust level of force model deviation

`level_v_lo`:

type: NoneType | float, optional, default: None
argument path: `explore[calypso]/convergence[fixed-levels-max-select]/level_v_lo`

The lower trust level of virial model deviation

`level_v_hi`:

type: NoneType | float, optional, default: None
argument path: `explore[calypso]/convergence[fixed-levels-max-select]/level_v_hi`

The higher trust level of virial model deviation

`conv_accuracy`:

type: float, optional, default: 0.9
argument path: `explore[calypso]/convergence[fixed-levels-max-select]/conv_accuracy`

If the ratio of accurate frames is larger than this value, the stage is converged

When `type` is set to `adaptive-lower`:

The method of adaptive adjust the lower trust levels. In each step of iterations, a number (set by `numb_candi_f` or `numb_candi_v`) or a ratio (set by `rate_candi_f` or `rate_candi_v`) of configurations with a model deviation lower than the higher trust level (`level_f_hi`, `level_v_hi`) are treated as candidates. The lowest model deviation of the candidates are treated as the lower trust level. If the lower trust level

does not change significant (controlled by *conv_tolerance*) in *n_checked_steps*, the stage is treated as converged.

level_f_hi:

type: float, optional, default: 0.5
argument path: explore[calypso]/convergence[adaptive-lower]/level_f_hi

The higher trust level of force model deviation

numb_candi_f:

type: int, optional, default: 200
argument path: explore[calypso]/convergence[adaptive-lower]/numb_candi_f

The number of force frames that has a model deviation lower than *level_f_hi* treated as candidate.

rate_candi_f:

type: float, optional, default: 0.01
argument path: explore[calypso]/convergence[adaptive-lower]/rate_candi_f

The ratio of force frames that has a model deviation lower than *level_f_hi* treated as candidate.

level_v_hi:

type: NoneType | float, optional, default: None
argument path: explore[calypso]/convergence[adaptive-lower]/level_v_hi

The higher trust level of virial model deviation

numb_candi_v:

type: int, optional, default: 0
argument path: explore[calypso]/convergence[adaptive-lower]/numb_candi_v

The number of virial frames that has a model deviation lower than *level_v_hi* treated as candidate.

rate_candi_v:

type: float, optional, default: 0.0
argument path: explore[calypso]/convergence[adaptive-lower]/rate_candi_v

The ratio of virial frames that has a model deviation lower than *level_v_hi* treated as candidate.

n_checked_steps:

type: int, optional, default: 2
argument path: explore[calypso]/convergence[adaptive-lower]/n_checked_steps

The number of steps to check the convergence.

conv_tolerance:

type: float, optional, default: 0.05
 argument path: explore[calypso]/convergence[adaptive-lower]/conv_tolerance

The convergence tolerance.

candi_sel_prob:

type: str, optional, default: uniform
 argument path: explore[calypso]/convergence[adaptive-lower]/candi_sel_prob

The method for selecting candidates. It can be ‘uniform’: all candidates are of the same probability. ‘inv_pop_f’ or ‘inv_pop_f:nhist’: the probability is inversely proportional to the population of a histogram between `leven_f_lo` and `level_f_hi`. The number of bins in the histogram is set by `nhist`, which should be an integer. The default is 10.

configurations:

type: list, alias: *configuration*
 argument path: explore[calypso]/configurations

A list of initial configurations.

This argument takes a list with each element containing the following:

Depending on the value of *type*, different sub args are accepted.

type:

type: str (flag key)
 argument path: explore[calypso]/configurations/type
 possible choices: `alloy`, `file`
 the type of the initial configuration generator.

When `type` is set to `alloy`:

Generate alloys with *a certain lattice or user provided structure*, the elements randomly occupying the lattice with *user provided probability*.

numb_confs:

type: int, optional, default: 1
 argument path:
`explore[calypso]/configurations[alloy]/numb_confs`

The number of configurations to generate

lattice:

type: list | tuple
 argument path:
`explore[calypso]/configurations[alloy]/lattice`

The lattice. Should be a list providing [“lattice_type”, `lattice_const`], or a list providing [“/path/to/dpdata/system”, “fmt”]

]. The two styles are distinguished by the type of the second element. Currently “lattice_type” can be “bcc”, “fcc”, “hcp”, “sc” or “diamond”.

replicate:

type: list | NoneType, optional, default: None

argument path:

explore[calypso]/configurations[alloy]/replicate

The number of replicates in each direction

concentration:

type: list | NoneType, optional, default: None

argument path: explore[calypso]/

configurations[alloy]/concentration

The concentration of each element. *List[List[float]]* or *List[float]* or *None*. If *List[float]*, the concentrations of each element. The length of the list should be the same as the *type_map*. If *List[List[float]]*, a list of concentrations (*List[float]*) is randomly picked from the List. If *None*, the elements are assumed to be of equal concentration.

cell_pert_frac:

type: float, optional, default: 0.0

argument path: explore[calypso]/

configurations[alloy]/cell_pert_frac

The fraction of cell perturbation

atom_pert_dist:

type: float, optional, default: 0.0

argument path: explore[calypso]/

configurations[alloy]/atom_pert_dist

The distance of atomic position perturbation

When *type* is set to *file*:

Generate alloys from user provided file(s). The file(s) are assume to be load by *dodata*.

files:

type: str | list

argument path:

explore[calypso]/configurations[file]/files

The paths to the configuration files. wildcards are supported.

prefix:

type: str | NoneType, optional, default: None

argument path:

explore[calypso]/configurations[file]/prefix

The prefix of file paths.

fmt:

type: str, optional, default: auto
 argument path:
`explore[calypso]/configurations[file]/fmt`
 The format (dpdata accepted formats) of the files.

remove_pbc:

type: bool, optional, default: False
 argument path:
`explore[calypso]/configurations[file]/remove_pbc`
 The remove the pbc of the data. shift the coords to the center of box so it can be used with lammps.

stages:

type: typing.List[typing.List[dict]]
 argument path: `explore[calypso]/stages`

The definition of exploration stages of type *List[List[ExplorationTaskGroup]]*. The outer list provides the enumeration of the exploration stages. Then each stage is defined by a list of exploration task groups. Each task group is described in *the task group definition*

fp:

type: dict
 argument path: fp

The configuration for FP

Depending on the value of *type*, different sub args are accepted.

type:

type: str (flag key)
 argument path: fp/type
 possible choices: `vasp, gaussian, deepmd, fpop_abacus`
 the type of the fp

When *type* is set to `vasp`:

inputs_config:

type: dict
 argument path: fp[vasp]/inputs_config
 Configuration for preparing vasp inputs

incar:

type: str
 argument path: fp[vasp]/inputs_config/incar
 The path to the template incar file

pp_files:

type: dict
argument path: fp[vasp]/inputs_config/pp_files
The pseudopotential files set by a dict, e.g. {"Al" : "path/to/the/al/pp/file", "Mg" : "path/to/the/mg/pp/file"}

kspacing:

type: float
argument path: fp[vasp]/inputs_config/kspacing
The spacing of k-point sampling. *ksapcing* will overwrite the incar template

kgamma:

type: bool, optional, default: True
argument path: fp[vasp]/inputs_config/kgamma
If the k-mesh includes the gamma point. *kgamma* will overwrite the incar template

run_config:

type: dict
argument path: fp[vasp]/run_config
Configuration for running vaspx tasks

command:

type: str, optional, default: vaspx
argument path: fp[vasp]/run_config/command
The command of VASP

out:

type: str, optional, default: data
argument path: fp[vasp]/run_config/out
The output dir name of labeled data. In *deepmd/npy* format provided by *dodata*.

log:

type: str, optional, default: fp.log
argument path: fp[vasp]/run_config/log
The log file name of VASP

task_max:

type: int, optional, default: 10
argument path: fp[vasp]/task_max
Maximum number of vaspx tasks for each iteration

When type is set to gaussian:

inputs_config:

type: dict

argument path: fp[gaussian]/inputs_config

Configuration for preparing vasp inputs

keywords:

type: str | list

argument path: fp[gaussian]/inputs_config/keywords

Gaussian keywords, e.g. force b3lyp/6-31g**. If a list, run multiple steps.

multiplicity:

type: str | int, optional, default: auto

argument path:

fp[gaussian]/inputs_config/multiplicity

spin multiplicity state. It can be a number. If auto, multiplicity will be detected automatically, with the following rules:

fragment_guesses=True multiplicity will +1 for each radical, and +2 for each oxygen molecule

fragment_guesses=False multiplicity will be 1 or 2, but +2 for each oxygen molecule.

charge:

type: int, optional, default: 0

argument path: fp[gaussian]/inputs_config/charge

molecule charge. Only used when charge is not provided by the system

basis_set:

type: str, optional

argument path: fp[gaussian]/inputs_config/basis_set

custom basis set

keywords_high_multiplicity:

type: str, optional

argument path: fp[gaussian]/inputs_config/keywords_high_multiplicity

keywords for points with multiple radicals. multiplicity should be auto. If not set, fallback to normal keywords

fragment_guesses:

type: bool, optional, default: False

argument path:

fp[gaussian]/inputs_config/fragment_guesses

initial guess generated from fragment guesses. If True, multiplicity should be auto

nproc:

type: int, optional, default: 1
argument path: fp[gaussian]/inputs_config/nproc
Number of CPUs to use

run_config:

type: dict
argument path: fp[gaussian]/run_config
Configuration for running vasp tasks

command:

type: str, optional, default: g16
argument path: fp[gaussian]/run_config/command
The command of Gaussian

out:

type: str, optional, default: data
argument path: fp[gaussian]/run_config/out
The output dir name of labeled data. In *deepmd/npy* format provided by *dodata*.

task_max:

type: int, optional, default: 10
argument path: fp[gaussian]/task_max
Maximum number of vasp tasks for each iteration

When type is set to deepmd:

inputs_config:

type: dict
argument path: fp[deepmd]/inputs_config
Configuration for preparing vasp inputs

run_config:

type: dict
argument path: fp[deepmd]/run_config
Configuration for running vasp tasks

teacher_model_path:

type: str | BinaryFileInput
argument path:
fp[deepmd]/run_config/teacher_model_path
The path of teacher model, which can be loaded by
deepmd.infer.DeepPot

out:

type: str, optional, default: data
 argument path: fp[deepmd]/run_config/out

The output dir name of labeled data. In *deepmd/npy* format provided by *dodata*.

log:

type: str, optional, default: fp.log
 argument path: fp[deepmd]/run_config/log

The log file name of dp

task_max:

type: int, optional, default: 10
 argument path: fp[deepmd]/task_max

Maximum number of vasp tasks for each iteration

When type is set to fpop_abacus:

inputs_config:

type: dict
 argument path: fp[fpop_abacus]/inputs_config

Configuration for preparing vasp inputs

input_file:

type: str
 argument path:
 fp[fpop_abacus]/inputs_config/input_file

A template INPUT file.

pp_files:

type: dict
 argument path:
 fp[fpop_abacus]/inputs_config/pp_files

The pseudopotential files for the elements. For example: {"H": "/path/to/H.upf", "O": "/path/to/O.upf"}.

element_mass:

type: dict | NoneType, optional, default: None
 argument path:
 fp[fpop_abacus]/inputs_config/element_mass

Specify the mass of some elements. For example: {"H": 1.0079, "O": 15.9994}.

kpt_file:

type: str | NoneType, optional, default: None

argument path:
fp[fpop_abacus]/inputs_config/kpt_file

The KPT file, by default None.

orb_files:

type: dict | NoneType, optional, default: None

argument path:

fp[fpop_abacus]/inputs_config/orb_files

The numerical orbital fields for the elements, by default None. For example: {"H": "/path/to/H.orb", "O": "/path/to/O.orb"}.

deepks_descriptor:

type: str | NoneType, optional, default: None

argument path:

fp[fpop_abacus]/inputs_config/deepks_descriptor

The deepks descriptor file, by default None.

deepks_model:

type: str | NoneType, optional, default: None

argument path:

fp[fpop_abacus]/inputs_config/deepks_model

The deepks model file, by default None.

run_config:

type: dict

argument path: fp[fpop_abacus]/run_config

Configuration for running vasp tasks

command:

type: str, optional, default: abacus

argument path: fp[fpop_abacus]/run_config/command

The command of abacus

out:

type: str, optional, default: data

argument path: fp[fpop_abacus]/run_config/out

The output dir name of labeled data. In *deepmd/npy* format provided by *dodata*.

task_max:

type: int, optional, default: 10

argument path: fp[fpop_abacus]/task_max

Maximum number of vasp tasks for each iteration

name:

type: str, optional, default: dpgen

argument path: `name`
The workflow name, ‘dpgen’ for default

4.1 Task group definition

4.1.1 LAMMPS task group

`task_group:`

type: `dict`
argument path: `task_group`

Depending on the value of `type`, different sub args are accepted.

`type:`

type: `str` (flag key)
argument path: `task_group/type`
possible choices: `lmp-md`, `lmp-template`, `customized-lmp-template`
the type of the task group

When `type` is set to `lmp-md` (or its alias `lmp-npt`):

Lammps MD tasks. DPGEN will generate the lammps input script

`conf_idx:`

type: `list`, alias: `sys_idx`
argument path: `task_group[lmp-md]/conf_idx`

The configurations of `configurations[conf_idx]` will be used to generate the initial configurations of the tasks. This key provides the index of selected item in the `configurations` array.

`n_sample:`

type: `NoneType` | `int`, optional, default: `None`
argument path: `task_group[lmp-md]/n_sample`

Number of configurations. If this number is smaller than the number of configurations in `configurations[conf_idx]`, then `n_sample` configurations are randomly sampled from `configurations[conf_idx]`, otherwise all configurations in `configurations[conf_idx]` will be used. If not provided, all configurations in `configurations[conf_idx]` will be used.

`temps:`

type: `list`, alias: `Ts`
argument path: `task_group[lmp-md]/temps`
A list of temperatures in K. Also used to initialize the temperature

`press:`

type: `list`, optional, alias: `Ps`
argument path: `task_group[lmp-md]/press`
A list of pressures in bar.

ens:

type: `str`, optional, default: `nve`, alias: *ensemble*
argument path: `task_group[lmp-md]/ens`

The ensemble. Allowed options are ‘nve’, ‘nvt’, ‘npt’, ‘npt-a’, ‘npt-t’. ‘npt-a’ stands for anisotropic box sampling and ‘npt-t’ stands for triclinic box sampling.

dt:

type: `float`, optional, default: `0.001`
argument path: `task_group[lmp-md]/dt`

The time step

nsteps:

type: `int`, optional, default: `100`
argument path: `task_group[lmp-md]/nsteps`
The number of steps

trj_freq:

type: `int`, optional, default: `10`, aliases: *t_freq*, *trj_freq*, *traj_freq*
argument path: `task_group[lmp-md]/trj_freq`
The number of steps

tau_t:

type: `float`, optional, default: `0.05`
argument path: `task_group[lmp-md]/tau_t`
The time scale of thermostat

tau_p:

type: `float`, optional, default: `0.5`
argument path: `task_group[lmp-md]/tau_p`
The time scale of barostat

pka_e:

type: `NoneType` | `float`, optional, default: `None`
argument path: `task_group[lmp-md]/pka_e`
The energy of primary knock-on atom

neidelay:

type: `NoneType` | `int`, optional, default: `None`
argument path: `task_group[lmp-md]/neidelay`
The delay of updating the neighbor list

no_pbc:

type: `bool`, optional, default: `False`
argument path: `task_group[lmp-md]/no_pbc`
Not using the periodic boundary condition

use_clusters:

type: bool, optional, default: False
 argument path: task_group[lmp-md]/use_clusters
 Calculate atomic model deviation

relative_f_epsilon:

type: NoneType | float, optional, default: None
 argument path: task_group[lmp-md]/relative_f_epsilon
 Calculate relative force model deviation

relative_v_epsilon:

type: NoneType | float, optional, default: None
 argument path: task_group[lmp-md]/relative_v_epsilon
 Calculate relative virial model deviation

When type is set to lmp-template:

Lammps MD tasks defined by templates. User provide lammps (and plumed) template for lammps tasks. The variables in templates are revised by the revisions key. Notice that the lines for pair style, dump and plumed are reserved for the revision of dpgen2, and the users should not write these lines by themselves. Rather, users notify dpgen2 the position of the line for *pair_style* by writing ‘pair_style deepmd’, the line for *dump* by writing ‘dump dpgen_dump’. If plumed is used, the line for fix *plumed* should be written exactly as ‘fix dpgen_plm’.

conf_idx:

type: list, alias: sys_idx
 argument path: task_group[lmp-template]/conf_idx

The configurations of *configurations[conf_idx]* will be used to generate the initial configurations of the tasks. This key provides the index of selected item in the *configurations* array.

n_sample:

type: NoneType | int, optional, default: None
 argument path: task_group[lmp-template]/n_sample

Number of configurations. If this number is smaller than the number of configurations in *configurations[conf_idx]*, then *n_sample* configurations are randomly sampled from *configurations[conf_idx]*, otherwise all configurations in *configurations[conf_idx]* will be used. If not provided, all configurations in *configurations[conf_idx]* will be used.

lmp_template_fname:

type: str, aliases: lmp_template, lmp
 argument path: task_group[lmp-template]/lmp_template_fname
 The file name of lammps input template

plm_template_fname:

type: str | NoneType, optional, default: None, aliases: plm_template, plm

argument path: `task_group[lmp-template]/plm_template_fname`

The file name of plumed input template

revisions:

type: dict, optional, default: {}

argument path: `task_group[lmp-template]/revisions`

traj_freq:

type: int, optional, default: 10, aliases: `t_freq`, `trj_freq`, `trj_freq`

argument path: `task_group[lmp-template]/traj_freq`

The frequency of dumping configurations and thermodynamic states

When `type` is set to `customized-lmp-template`:

Lammps MD tasks defined by user customized shell commands and templates. User provided shell script generates a series of folders, and each folder contains a lammps template task group.

conf_idx:

type: list, alias: `sys_idx`

argument path: `task_group[customized-lmp-template]/conf_idx`

The configurations of `configurations[conf_idx]` will be used to generate the initial configurations of the tasks. This key provides the index of selected item in the `configurations` array.

n_sample:

type: NoneType | int, optional, default: None

argument path: `task_group[customized-lmp-template]/n_sample`

Number of configurations. If this number is smaller than the number of configurations in `configurations[conf_idx]`, then `n_sample` configurations are randomly sampled from `configurations[conf_idx]`, otherwise all configurations in `configurations[conf_idx]` will be used. If not provided, all configurations in `configurations[conf_idx]` will be used.

custom_shell_commands:

type: list

argument path:

`task_group[customized-lmp-template]/custom_shell_commands`

Customized shell commands to be run for each configuration. The commands require `input_lmp_conf_name` as input conf file, `input_lmp_tmpl_name` and `input_plm_tmpl_name` as templates, and `input_extra_files` as extra input files. By running the commands a series folders in pattern `output_dir_pattern` are supposed to be generated, and each folder is supposed to contain a configuration file `output_lmp_conf_name`, a lammps template file `output_lmp_tmpl_name` and a plumed template file `output_plm_tmpl_name`.

revisions:

type: dict, optional, default: {}

argument path: `task_group[customized-lmp-template]/revisions`

The revisions. Should be a dict providing the key - list of desired values pair. Key is the word to be replaced in the templates, and it may appear in both the lammps and plumed input templates. All values in the value list will be enumerated.

traj_freq:

type: int, optional, default: 10, aliases: *t_freq*, *trj_freq*, *trj_freq*
 argument path: task_group[customized-lmp-template]/traj_freq

The frequency of dumping configurations and thermodynamic states

input_lmp_conf_name:

type: str, optional, default: conf.lmp
 argument path:
 task_group[customized-lmp-template]/input_lmp_conf_name

Input conf file name for the shell commands.

input_lmp_tmpl_name:

type: str, optional, default: in.lammps, aliases: *lmp_template*, *lmp*
 argument path:
 task_group[customized-lmp-template]/input_lmp_tmpl_name

The file name of lammps input template

input_plm_tmpl_name:

type: str | NoneType, optional, default: None, aliases: *plm_template*, *plm*
 argument path:
 task_group[customized-lmp-template]/input_plm_tmpl_name

The file name of plumed input template

input_extra_files:

type: list, optional, default: []
 argument path:
 task_group[customized-lmp-template]/input_extra_files

Extra files that may be needed to execute the shell commands

output_dir_pattern:

type: str | list, optional, default: *
 argument path:
 task_group[customized-lmp-template]/output_dir_pattern

Pattern of resultant folders generated by the shell commands.

output_lmp_conf_name:

type: str, optional, default: conf.lmp
 argument path:
 task_group[customized-lmp-template]/output_lmp_conf_name

Generated conf file name.

output_lmp_tmpl_name:

type: str, optional, default: in.lammps

argument path:
task_group[customized-lmp-template]/output_lmp_tmpl_name
Generated lmp input file name.

output_plm_tmpl_name:

type: str, optional, default: input.plumed
argument path:
task_group[customized-plm-template]/output_plm_tmpl_name
Generated plm input file name.

4.1.2 CALYPSO task group

task_group:

type: dict
argument path: task_group
CALYPSO structure prediction tasks. DPGEN will generate the calypso input script

numb_of_species:

type: int
argument path: task_group/numb_of_species
number of species.

name_of_atoms:

type: list
argument path: task_group/name_of_atoms
name of atoms.

atomic_number:

type: list
argument path: task_group/atomic_number
atomic number of each element.

numb_of_atoms:

type: list
argument path: task_group/numb_of_atoms
number of each atom.

distance_of_ions:

type: list
argument path: task_group/distance_of_ions
the distance matrix between different elements.

pop_size:

type: int, optional, default: 30
argument path: task_group/pop_size
the number of structures would be generated in each step.

max_step:

type: `int`, optional, default: 5
 argument path: `task_group/max_step`
 the max iteration number of CALYPSO loop.

system_name:

type: `str`, optional, default: CALYPSO
 argument path: `task_group/system_name`
 system name.

numb_of_formula:

type: `list`, optional, default: [1, 1]
 argument path: `task_group/numb_of_formula`
 the formula range of simulation cell.

pressure:

type: `float`, optional, default: 0.001
 argument path: `task_group/pressure`
 the aim pressure (in Kbar) when using MLP to optimize structures.

fmax:

type: `float`, optional, default: 0.01
 argument path: `task_group/fmax`
 the converge criterion. The force on all individual atoms should be less than *fmax*.

volume:

type: `float`, optional, default: 0
 argument path: `task_group/volume`
 the volume of simulation cell in one formula.

ialgo:

type: `int`, optional, default: 2
 argument path: `task_group/ialgo`
 the evolution algorithm of CALYPSO. 1: global pso, 2: local pso, 3: sabc.

pso_ratio:

type: `float`, optional, default: 0.6
 argument path: `task_group/pso_ratio`
 the ratio of structures generated by evolution algorithm in one step.

icode:

type: `int`, optional, default: 15
 argument path: `task_group/icode`
 the software of structure optimization. 1: VASP, 15: DP.

numb_of_lbest:

type: int, optional, default: 4
argument path: task_group/numb_of_lbest
the number of evolution direction when using LPSO.

numb_of_local_optim:

type: int, optional, default: 3
argument path: task_group/numb_of_local_optim
the number of making structure optimization when using dft.

command:

type: str, optional, default: sh submit.sh
argument path: task_group/command
the command of running structure optimization.

max_time:

type: int, optional, default: 9000
argument path: task_group/max_time
the max time (in second) of structure optimization.

pick_up:

type: bool, optional, default: False
argument path: task_group/pick_up
whether to continue the calculation.

pick_step:

type: int, optional, default: 0
argument path: task_group/pick_step
from which step to continue the calculation.

parallel:

type: bool, optional, default: False
argument path: task_group/parallel
whether to run calypso in parallel.

split:

type: bool, optional, default: True
argument path: task_group/split
specify generating structures and structure optimizations. in dpge2, split must be True.

spec_space_group:

type: list, optional, default: [2, 230]
argument path: task_group/spec_space_group
the range of spacegroup.

vsc:

type: bool, optional, default: False
argument path: task_group/vsc
whether to run calypso in variational stoichiometry way.

ctrl_range:

type: list, optional, default: [[1, 10]]
argument path: task_group/ctrl_range
the atom range of each atoms.

max_numb_atoms:

type: int, optional, default: 100
argument path: task_group/max_numb_atoms
the max number of atoms.

DEVELOPERS' GUIDE

- *The concurrent learning algorithm*
- *Overview of the DPGEN2 implementation*
- *The DPGEN2 workflow*
- *How to contribute*

5.1 The concurrent learning algorithm

DPGEN2 implements the concurrent learning algorithm named DP-GEN, described in [this paper](#). It is noted that other types of workflows, like active learning, should be easily implemented within the infrastructure of DPGEN2.

The DP-GEN algorithm is iterative. In each iteration, four steps are consecutively executed: training, exploration, selection, and labeling.

1. **Training.** A set of DP models are trained with the same dataset and the same hyperparameters. The only difference is the random seed initializing the model parameters.
2. **Exploration.** One of the DP models is used to explore the configuration space. The strategy of exploration highly depends on the purpose of the application case of the model. The simulation technique for exploration can be molecular dynamics, Monte Carlo, structure search/optimization, enhanced sampling, or any combination of them. Current DPGEN2 only supports exploration based on molecular simulation platform [LAMMPS](#).
3. **Selection.** Not all the explored configurations are labeled, rather, the model prediction errors on the configurations are estimated by the *model deviation*, which is defined as the standard deviation in predictions of the set of the models. The critical configurations with large and not-that-large errors are selected for labeling. The configurations with very large errors are not selected because the large error is usually caused by non-physical configurations, e.g. overlapping atoms.
4. **Labeling.** The selected configurations are labeled with energy, forces, and virial calculated by a method of first-principles accuracy. The usually used method is the [density functional theory](#) implemented in [VASP](#), [Quantum Espresso](#), [CP2K](#), and etc.. The labeled data are finally added to the training dataset to start the next iteration.

In each iteration, the quality of the model is improved by selecting and labeling more critical data and adding them to the training dataset. The DP-GEN iteration is converged when no more critical data can be selected.

5.2 Overview of the DPGEN2 Implementation

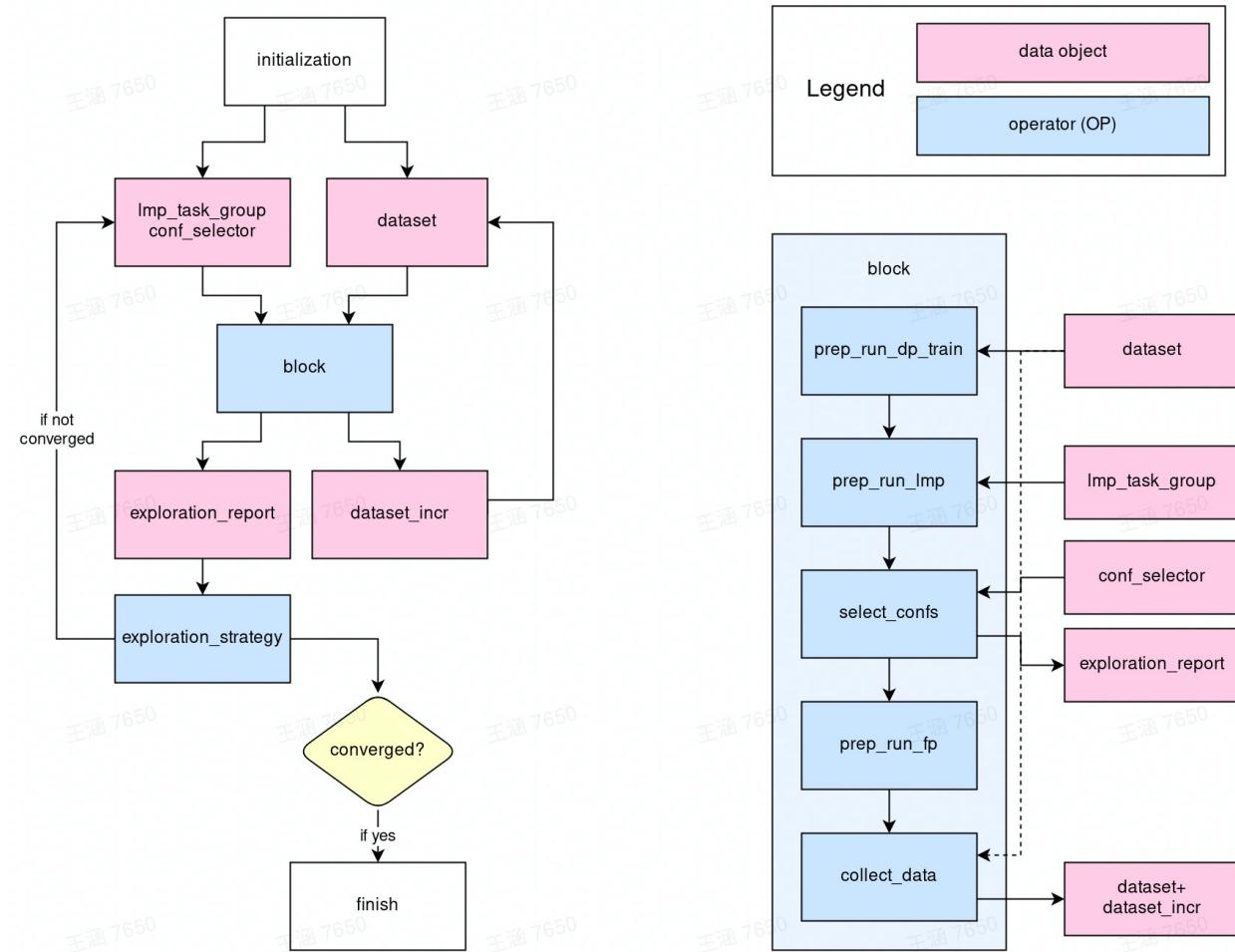
The implementation DPGEN2 is based on the workflow platform [dflow](#), which is a python wrapper of the [Argo Workflows](#), an open-source container-native workflow engine on [Kubernetes](#).

The DP-GEN algorithm is conceptually modeled as a computational graph. The implementation is then considered as two lines: the operators and the workflow.

1. **Operators.** Operators are implemented in Python v3. The operators should be implemented and tested *without* the workflow.
2. **Workflow.** Workflow is implemented on [dflow](#). Ideally, the workflow is implemented and tested with all operators mocked.

5.3 The DPGEN2 workflow

The workflow of DPGEN2 is illustrated in the following figure



In the center is the **block** operator, which is a super-OP (an OP composed by several OPs) for one DP-GEN iteration, i.e. the super-OP of the training, exploration, selection, and labeling steps. The inputs of the **block** OP are **Imp_task_group**, **conf_selector** and **dataset**.

- **Imp_task_group:** definition of a group of LAMMPS tasks that explore the configuration space.

- `conf_selector`: defines the rule by which the configurations are selected for labeling.
- `dataset`: the training dataset.

The outputs of the `block` OP are

- `exploration_report`: a report recording the result of the exploration. For example, how many configurations are accurate enough and how many are selected as candidates for labeling.
- `dataset_incr`: the increment of the training dataset.

The `dataset_incr` is added to the training dataset.

The `exploration_report` is passed to the `exploration_strategy` OP. The `exploration_strategy` implements the strategy of exploration. It reads the `exploration_report` generated by each iteration (`block`), then tells if the iteration is converged. If not, it generates a group of LAMMPS tasks (`lmp_task_group`) and the criteria of selecting configurations (`conf_selector`). The `lmp_task_group` and `conf_selector` are then used by `block` of the next iteration. The iteration closes.

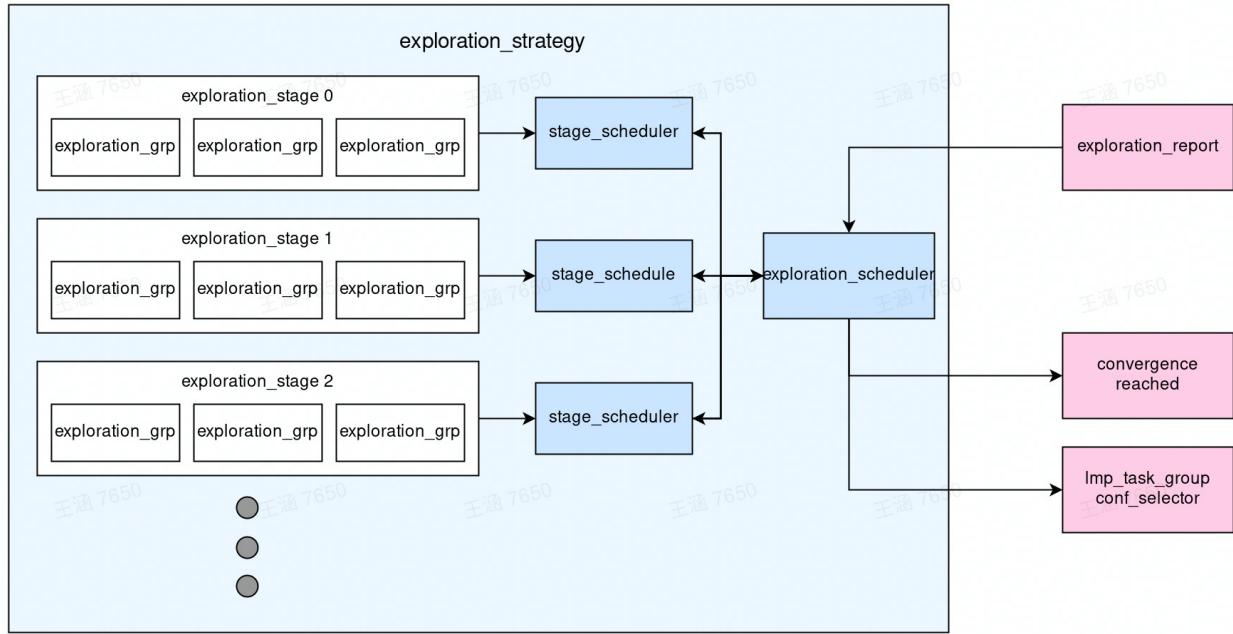
5.3.1 Inside the block operator

The inside of the super-OP `block` is displayed on the right-hand side of the figure. It contains the following steps to finish one DPGEN iteration

- `prep_run_dp_train`: prepares training tasks of DP models and runs them.
- `prep_run_lmp`: prepares the LAMMPS exploration tasks and runs them.
- `select_confs`: selects configurations for labeling from the explored configurations.
- `prep_run_fp`: prepares and runs first-principles tasks.
- `collect_data`: collects the `dataset_incr` and adds it to the `dataset`.

5.3.2 The exploration strategy

The exploration strategy defines how the configuration space is explored by the concurrent learning algorithm. The design of the exploration strategy is graphically illustrated in the following figure. The exploration is composed of stages. Only the DP-GEN exploration is converged at one stage (no configuration with a large error is explored), the exploration goes to the next iteration. The whole procedure is controlled by `exploration_scheduler`. Each stage has its schedule, which talks to the `exploration_scheduler` to generate the schedule for the DP-GEN algorithm.



Some concepts are explained below:

- **Exploration group.** A group of LAMMPS tasks shares similar settings. For example, a group of NPT MD simulations in a certain thermodynamic space.
- **Exploration stage.** The `exploration_stage` contains a list of exploration groups. It contains all information needed to define the `Imp_task_group` used by the `block` in the DP-GEN iteration.
- **Stage scheduler.** It guarantees the convergence of the DP-GEN algorithm in each `exploration_stage`. If the exploration is not converged, the `stage_scheduler` generates `Imp_task_group` and `conf_selector` from the `exploration_stage` for the next iteration (probably with a different initial condition, i.e. different initial configurations and randomly generated initial velocity).
- **Exploration scheduler.** The scheduler for the DP-GEN algorithm. When DP-GEN is converged in one of the stages, it goes to the next stage until all planned stages are used.

5.4 How to contribute

Anyone interested in the DPGEN2 project may contribute OPs, workflows, and exploration strategies.

- To contribute OPs, one may check the [guide on writing operators](#)
- To contribute workflows, one may take the DP-GEN workflow as an example. It is implemented in `dp-gen2/flow/dpgen_loop.py` and tested with all operators mocked in `test/test_dpgen_loop.py`
- To contribute the exploration strategy, one may check the [guide on writing exploration strategies](#)

OPERATORS

There are two types of OPs in DPGEN2

- *OP*. An execution unit the the workflow. It can be roughly viewed as a piece of Python script taking some input and gives some outputs. An OP cannot be used in the `dflow` until it is embedded in a super-OP.
- *Super-OP*. An execution unite that is composed by one or more OP and/or super-OPs.

Technically, OP is a Python class derived from `dflow.python.OP`. It serves as the `PythonOPTemplate` of `dflow.Step`.

The super-OP is a Python class derived from `dflow.Steps`. It contains `dflow.Steps` as building blocks, and can be used as OP template to generate a `dflow.Step`. The explanation of the concepts `dflow.Step` and `dflow.Steps`, one may refer to the [manual of dflow](#).

6.1 The super-OP PrepRunDPTrain

In the following we will take the `PrepRunDPTrain` super-OP as an example to illustrate how to write OPs in DPGEN2.

`PrepRunDPTrain` is a super-OP that prepares several DeePMD-kit training tasks, and submit all of them. This super-OP is composed by two `dflow.Steps` building from `dflow.python.OPs` `PrepDPTrain` and `RunDPTrain`.

```
from dflow import (
    Step,
    Steps,
)
from dflow.python import(
    PythonOPTemplate,
    OP,
    Slices,
)

class PrepRunDPTrain(Steps):
    def __init__(
        self,
        name : str,
        prep_train_op : OP,
        run_train_op : OP,
        prep_train_image : str = "dflow:v1.0",
        run_train_image : str = "dflow:v1.0",
    ):
        ...
        ...
```

(continues on next page)

(continued from previous page)

```

self = _prep_run_dp_train(
    self,
    self.step_keys,
    prep_train_op,
    run_train_op,
    prep_train_image = prep_train_image,
    run_train_image = run_train_image,
)

```

The construction of the PrepRunDPTrain takes prepare-training OP and run-training OP and their docker images as input, and implemented in internal method `_prep_run_dp_train`.

```

def _prep_run_dp_train(
    train_steps,
    step_keys,
    prep_train_op : OP = PrepDPTrain,
    run_train_op : OP = RunDPTTrain,
    prep_train_image : str = "dflow:v1.0",
    run_train_image : str = "dflow:v1.0",
):
    prep_train = Step(
        ...
        template=PythonOPTemplate(
            prep_train_op,
            image=prep_train_image,
        ...
    ),
    ...
)
    train_steps.add(prep_train)

    run_train = Step(
        ...
        template=PythonOPTemplate(
            run_train_op,
            image=run_train_image,
        ...
    ),
    ...
)
    train_steps.add(run_train)

    train_steps.outputs.artifacts["scripts"]._from = run_train.outputs.artifacts["script"]
    train_steps.outputs.artifacts["models"]._from = run_train.outputs.artifacts["model"]
    train_steps.outputs.artifacts["logs"]._from = run_train.outputs.artifacts["log"]
    train_steps.outputs.artifacts["lcurves"]._from = run_train.outputs.artifacts["lcurve"]
)
    return train_steps

```

In `_prep_run_dp_train`, two instances of `dflow.Step`, i.e. `prep_train` and `run_train`, generated from `prep_train_op` and `run_train_op`, respectively, are added to `train_steps`. Both of `prep_train_op` and

`run_train_op` are OPs (python classes derived from `dflow.python.OPs`) that will be illustrated later. `train_steps` is an instance of `dflow.Steps`. The outputs of the second OP `run_train` are assigned to the outputs of the `train_steps`.

The `prep_train` prepares a list of paths, each of which contains all necessary files to start a DeePMD-kit training tasks.

The `run_train` slices the list of paths, and assign each item in the list to a DeePMD-kit task. The task is executed by `run_train_op`. This is a very nice feature of `dflow`, because the developer only needs to implement how one DeePMD-kit task is executed, and then all the items in the task list will be executed `in parallel`. See the following code to see how it works

```
run_train = Step(
    'run-train',
    template=PythonOPTemplate(
        run_train_op,
        image=run_train_image,
        slices = Slices(
            "int('{{item}}')",
            input_parameter = ["task_name"],
            input_artifact = ["task_path", "init_model"],
            output_artifact = ["model", "lcurve", "log", "script"],
        ),
    ),
    parameters={
        "config" : train_steps.inputs.parameters["train_config"],
        "task_name" : prep_train.outputs.parameters["task_names"],
    },
    artifacts={
        'task_path' : prep_train.outputs.artifacts['task_paths'],
        'init_model' : train_steps.inputs.artifacts['init_models'],
        'init_data': train_steps.inputs.artifacts['init_data'],
        'iter_data': train_steps.inputs.artifacts['iter_data'],
    },
    with_sequence=argo_sequence(argo_len(prep_train.outputs.parameters["task_names"
    ↵"])), format=train_index_pattern),
    key = step_keys['run-train'],
)
```

The input parameter "`task_names`" and artifacts "`task_paths`" and "`init_model`" are sliced and supplied to each DeePMD-kit task. The output artifacts of the tasks ("`model`", "`lcurve`", "`log`" and "`script`") are stacked in the same order as the input lists. These lists are assigned as the outputs of `train_steps` by

```
train_steps.outputs.artifacts["scripts"]._from = run_train.outputs.artifacts["script"
↪"]
train_steps.outputs.artifacts["models"]._from = run_train.outputs.artifacts["model"]
train_steps.outputs.artifacts["logs"]._from = run_train.outputs.artifacts["log"]
train_steps.outputs.artifacts["lcurves"]._from = run_train.outputs.artifacts["lcurve"
↪"]
```

6.2 The OP RunDPTTrain

We will take RunDPTTrain as an example to illustrate how to implement an OP in DPGEN2. The source code of this OP is found [here](#)

Firstly of all, an OP should be implemented as a derived class of `dflow.python.OP`.

The `dflow.python.OP` requires static type define for the input and output variables, i.e. the signatures of an OP. The input and output signatures of the `dflow.python.OP` are given by classmethods `get_input_sign` and `get_output_sign`.

```
from dflow.python import (
    OP,
    OPIO,
    OPIOSign,
    Artifact,
)
class RunDPTTrain(OP):
    @classmethod
    def get_input_sign(cls):
        return OPIOSign({
            "config" : dict,
            "task_name" : str,
            "task_path" : Artifact(Path),
            "init_model" : Artifact(Path),
            "init_data" : Artifact(List[Path]),
            "iter_data" : Artifact(List[Path]),
        })

    @classmethod
    def get_output_sign(cls):
        return OPIOSign({
            "script" : Artifact(Path),
            "model" : Artifact(Path),
            "lcurve" : Artifact(Path),
            "log" : Artifact(Path),
        })
```

All items not defined as `Artifact` are treated as parameters of the OP. The concept of parameter and artifact are explained in the [dflow document](#). To be short, the artifacts can be `pathlib.Path` or a list of `pathlib.Path`. The artifacts are passed by the file system. Other data structures are treated as parameters, they are passed as variables encoded in `str`. Therefore, a large amount of information should be stored in artifacts, otherwise they can be considered as parameters.

The operation of the OP is implemented in method `execute`, and are run in docker containers. Again taking the `execute` method of `RunDPTTrain` as an example

```
@OP.exec_sign_check
def execute(
    self,
    ip : OPIO,
) -> OPIO:
    ...
    task_name = ip['task_name']
```

(continues on next page)

(continued from previous page)

```

task_path = ip['task_path']
init_model = ip['init_model']
init_data = ip['init_data']
iter_data = ip['iter_data']

...
work_dir = Path(task_name)
...
# here copy all files in task_path to work_dir
...
with set_directory(work_dir):
    fplog = open('train.log', 'w')
    def clean_before_quit():
        fplog.close()
    # train model
    command = ['dp', 'train', train_script_name]
    ret, out, err = run_command(command)
    if ret != 0:
        clean_before_quit()
        raise FatalError('dp train failed')
    fplog.write(out)
    # freeze model
    ret, out, err = run_command(['dp', 'freeze', '-o', 'frozen_model.pb'])
    if ret != 0:
        clean_before_quit()
        raise FatalError('dp freeze failed')
    fplog.write(out)
    clean_before_quit()

return OPIO({
    "script" : work_dir / train_script_name,
    "model" : work_dir / "frozen_model.pb",
    "lcurve" : work_dir / "lcurve.out",
    "log" : work_dir / "train.log",
})

```

The inputs and outputs variables are recorded in data structure `dflow.python.OPIO`, which is initialized by a Python dict. The keys in the input/output dict, and the types of the input/output variables will be checked against their signatures by decorator `OP.exec_sign_check`. If any key or type does not match, an exception will be raised.

It is noted that all input artifacts of the OP are read-only, therefore, the first step of the `RunDPTrain.execute` is to copy all necessary input files from the directory `task_path` prepared by `PrepDPTrain` to the working directory `work_dir`.

`with_directory` method creates the `work_dir` and switches to the directory before the execution, and then exits the directory when the task finishes or an error is raised.

In what follows, the training and model frozen bash commands are executed consecutively. The return code is checked and a `FatalError` is raised if a non-zero code is detected.

Finally the trained model file, input script, learning curve file and the log file are recorded in a `dflow.python.OPIO` and returned.

EXPLORATION

DPGEN2 allows developers to contribute exploration strategies. The exploration strategy defines how the configuration space is explored by molecular simulations in each DPGEN iteration. Notice that we are not restricted to molecular dynamics, any molecular simulation is, in principle, allowed. For example, Monte Carlo, enhanced sampling, structure optimization, and so on.

An *exploration strategy* takes the history of exploration as input, and gives back DPGEN the exploration tasks (we call it **task group**) and the rule to select configurations from the trajectories generated by the tasks (we call it **configuration selector**).

One can contribute from three aspects:

- *The stage scheduler*
- *The exploration task groups*
- *Configuration selector*

7.1 Stage scheduler

The stage scheduler takes an exploration report passed from the exploration scheduler as input, and tells the exploration scheduler if the exploration in the stage is converged, if not, returns a group of exploration tasks and a configuration selector that are used in the next DPGEN iteration.

Detailed explanation of the concepts are found [here](#).

All the stage schedulers are derived from the abstract base class `StageScheduler`. The only interface to be implemented is `StageScheduler.plan_next_iteration`. One may check the doc string for the explanation of the interface.

```
class StageScheduler(ABC):
    """
    The scheduler for an exploration stage.
    """

    @abstractmethod
    def plan_next_iteration(
        self,
        hist_reports : List[ExplorationReport],
        report : ExplorationReport,
        confs : List[Path],
    ) -> Tuple[bool, ExplorationTaskGroup, ConfSelector] :
        """
```

(continues on next page)

(continued from previous page)

Make the plan for the next iteration of the stage.

It checks the report of the current and all historical iterations of the stage, and tells if the iterations are converged.

If not converged, it will plan the next iteration for the stage.

Parameters

hist_reports: List[ExplorationReport]

The historical exploration report of the stage. If this is the first iteration of the stage, this list is empty.

report : ExplorationReport

The exploration report of this iteration.

confs: List[Path]

A list of configurations generated during the exploration. May be used to generate new configurations for the next iteration.

Returns

converged: bool

If the stage converged.

task: ExplorationTaskGroup

A `ExplorationTaskGroup` defining the exploration of the next iteration.

Should be `None` if the stage is converged.

conf_selector: ConfSelector

The configuration selector for the next iteration. Should be `None` if the

stage is converged.

.....

One may check more details on the *exploratin task group* and the configuration selector.

7.2 Exploration task groups

DPGEN2 defines a python class `ExplorationTask` to manage all necessary files needed to run a exploration task. It can be used as the example provided in the doc string.

```
class ExplorationTask():
    """Define the files needed by an exploration task.
```

Examples

```
>>> # this example dumps all files needed by the task.
>>> files = exploration_task.files()
... for file_name, file_content in files.items():
...     with open(file_name, 'w') as fp:
...         fp.write(file_content)
```

.....

A collection of the exploration tasks is called exploration task group. All tasks groups are derived from the base class

`ExplorationTaskGroup`. The exploration task group can be viewed as a list of `ExplorationTasks`, one may get the list by using property `ExplorationTaskGroup.task_list`. One may add tasks, or `ExplorationTaskGroup` to the group by methods `ExplorationTaskGroup.add_task` and `ExplorationTaskGroup.add_group`, respectively.

```
class ExplorationTaskGroup(Sequence):
    @property
    def task_list(self) -> List[ExplorationTask]:
        """Get the `list` of `ExplorationTask`"""
        ...

    def add_task(self, task: ExplorationTask):
        """Add one task to the group."""
        ...

    def add_group(
            self,
            group : 'ExplorationTaskGroup',
    ):
        """Add another group to the group."""
        ...
```

An example of generating a group of NPT MD simulations may illustrate how to implement the `ExplorationTaskGroups`.

7.3 Configuration selector

The configuration selectors are derived from the abstract base class `ConfSelector`

```
class ConfSelector(ABC):
    """Select configurations from trajectory and model deviation files.
    """

    @abstractmethod
    def select (
            self,
            trajs : List[Path],
            model_devis : List[Path],
            traj_fmt : str = 'deepmd/npy',
            type_map : List[str] = None,
    ) -> Tuple[List[ Path ], ExplorationReport]:
```

The abstractmethod to implement is `ConfSelector.select`. `trajs` and `model_devis` are lists of files that recording the simulations trajectories and model deviations respectively. `traj_fmt` and `type_map` are parameters that may be needed for loading the trajectories by `dodata`.

The `ConfSelector.select` returns a `Path`, each of which can be treated as a `dodata.MultiSystems`, and a `ExplorationReport`.

An example of selecting configurations from LAMMPS trajectories may illustrate how to implement the `ConfSelectors`.

DPGEN2 API

8.1 dpgen2 package

8.1.1 Subpackages

`dpgen2.conf` package

Submodules

`dpgen2.conf.alloy_conf` module

```
class dpgen2.conf.alloy_conf.AlloyConf(lattice: System | Tuple[str, float], type_map: List[str], replicate: List[int] | Tuple[int] | int | None = None)
```

Bases: `object`

Parameters

`lattice Union[dpdata.System, Tuple[str, float]]`

Lattice of the alloy confs. can be `dodata.System`: lattice in `dodata.System` `Tuple[str, float]`: pair of lattice type and lattice constant. lattice type can be “bcc”, “fcc”, “hcp”, “sc” or “diamond”

`replicate Union[List[int], Tuple[int], int]`

replicate of the lattice

`type_map List[str]`

The type map

Methods

```
generate_file_content(numb_confs[, ...])
```

Parameters

```
generate_systems(numb_confs[, ...])
```

Parameters

```
generate_file_content(numb_confs, concentration: List[List[float]] | List[float] | None = None,  
cell_pert_frac: float = 0.0, atom_pert_dist: float = 0.0, fmt: str = 'lammps/lmp')  
→ List[str]
```

Parameters**numb_confs int**

Number of configurations to generate

concentration List[List[float]] or List[float] or None

If *List[float]*, the concentrations of each element. The length of the list should be the same as the *type_map*. If *List[List[float]]*, a list of concentrations (*List[float]*) is randomly picked from the List. If *None*, the elements are assumed to be of equal concentration.

cell_pert_frac float

fraction of cell perturbation

atom_pert_dist float

the atom perturbation distance (unit angstrom).

fmt str

the format of the returned conf strings. Should be one of the formats supported by *dodata*

Returns**conf_list List[str]**

A list of file content of configurations.

generate_systems(*numb_confs*, *concentration*: *List[List[float]]* | *List[float]* | *None* = *None*, *cell_pert_frac*: *float* = 0.0, *atom_pert_dist*: *float* = 0.0) → *List[System]*

Parameters**numb_confs int**

Number of configurations to generate

concentration List[List[float]] or List[float] or None

If *List[float]*, the concentrations of each element. The length of the list should be the same as the *type_map*. If *List[List[float]]*, a list of concentrations (*List[float]*) is randomly picked from the List. If *None*, the elements are assumed to be of equal concentration.

cell_pert_frac float

fraction of cell perturbation

atom_pert_dist float

the atom perturbation distance (unit angstrom).

Returns**conf_list List[dodata.System]**

A list of generated confs in *dodata.System*.

class *dpgen2.conf.alloy_conf.AlloyConfGenerator*(*numb_confs*, *lattice*: *System* | *Tuple[str, float]*,
replicate: *List[int]* | *Tuple[int]* | *int* | *None* = *None*,
concentration: *List[List[float]]* | *List[float]* | *None* = *None*, *cell_pert_frac*: *float* = 0.0, *atom_pert_dist*: *float* = 0.0)

Bases: *ConfGenerator*

Parameters**numb_confs int**

Number of configurations to generate

lattice Union[*dodata.System*, *Tuple[str, float]*]

Lattice of the alloy confs. can be *dodata.System*: lattice in *dodata.System Tuple[str, float]*: pair of lattice type and lattice constant. lattice type can be “bcc”, “fcc”, “hcp”, “sc” or “diamond”

replicate Union[*List[int]*, *Tuple[int, int]*, *int*]

replicate of the lattice

concentration *List[List[float]]* or *List[float]* or *None*

If *List[float]*, the concentrations of each element. The length of the list should be the same as the *type_map*. If *List[List[float]]*, a list of concentrations (*List[float]*) is randomly picked from the List. If *None*, the elements are assumed to be of equal concentration.

cell_pert_frac *float*

fraction of cell perturbation

atom_pert_dist *float*

the atom perturbation distance (unit angstrom).

Methods

<code>generate(type_map)</code>	Method of generating configurations.
<code>get_file_content(type_map[, fmt])</code>	Get the file content of configurations
<code>normalize_config([data, strict])</code>	Normalized the argument.

args
doc

static args() → *List[Argument]*

static doc() → *str*

generate(*type_map*) → *MultiSystems*

Method of generating configurations.

Parameters**type_map**

[*List[str]*] The type map.

Returns**confs: *dodata.MultiSystems***

The returned configurations in *dodata.MultiSystems* format

`dpgen2.conf.alloy_conf.gen_doc(*, make_anchor=True, make_link=True, **kwargs)`

`dpgen2.conf.alloy_conf.generate_alloy_conf_args()`

`dpgen2.conf.alloy_conf.generate_alloy_conf_file_content(lattice: System | Tuple[str, float], type_map: List[str], numb_confs, replicate: List[int] | Tuple[int] | int | None = None, concentration: List[List[float]] | List[float] | None = None, cell_pert_frac: float = 0.0, atom_pert_dist: float = 0.0, fmt: str = 'lammps/lmp')`

```
dpgen2.conf.alloy_conf.normalize(data)
```

dpgen2.conf.conf_generator module

```
class dpgen2.conf.conf_generator.ConfGenerator  
Bases: ABC
```

Methods

<code>generate(type_map)</code>	Method of generating configurations.
<code>get_file_content(type_map[, fmt])</code>	Get the file content of configurations
<code>normalize_config([data, strict])</code>	Normalized the argument.

args

```
abstract static args() → List[Argument]
```

```
abstract generate(type_map) → MultiSystems
```

Method of generating configurations.

Parameters

`type_map`

[List[str]] The type map.

Returns

`conf: dpdata.MultiSystems`

The returned configurations in *dppdata.MultiSystems* format

```
get_file_content(type_map, fmt='lammps/lmp') → List[str]
```

Get the file content of configurations

Parameters

`type_map`

[List[str]] The type map.

Returns

`conf_list: List[str]`

A list of file content of configurations.

```
classmethod normalize_config(data: Dict = {}, strict: bool = True) → Dict
```

Normalized the argument.

Parameters

`data`

[Dict] The input dict of arguments.

`strict`

[bool] Strictly check the arguments.

Returns

data: Dict

The normalized arguments.

dpgen2.conf.file_conf module

```
class dpgen2.conf.file_conf.FileConfGenerator(files: str | List[str], fmt: str = 'auto', prefix: str | None = None, remove_pbc: bool | None = False)
```

Bases: *ConfGenerator*

Methods

<code>generate(type_map)</code>	Method of generating configurations.
<code>get_file_content(type_map[, fmt])</code>	Get the file content of configurations
<code>normalize_config([data, strict])</code>	Normalized the argument.

<code>args</code>
<code>doc</code>
<code>generate_mixed</code>
<code>generate_std</code>

static args() → List[Argument]

static doc() → str

generate(type_map) → MultiSystems

Method of generating configurations.

Parameters**type_map**

[List[str]] The type map.

Returns**conf: dpdata.MultiSystems**

The returned configurations in *dpdata.MultiSystems* format

generate_mixed(type_map) → MultiSystems

generate_std(type_map) → MultiSystems

dpgen2.conf.unit_cells module

```
class dpgen2.conf.unit_cells.BCC
```

Bases: *object*

Methods

gen_box
numb_atoms
poscar_unit

```
gen_box()  
numb_atoms()  
poscar_unit(latt)  
  
class dpgen2.conf.unit_cells.DIAMOND  
Bases: object
```

Methods

gen_box
numb_atoms
poscar_unit

```
gen_box()  
numb_atoms()  
poscar_unit(latt)  
  
class dpgen2.conf.unit_cells.FCC  
Bases: object
```

Methods

gen_box
numb_atoms
poscar_unit

```
gen_box()  
numb_atoms()  
poscar_unit(latt)  
  
class dpgen2.conf.unit_cells.HCP  
Bases: object
```

Methods

gen_box
numb_atoms
poscar_unit

```
gen_box()
numb_atoms()
poscar_unit(latt)
class dpigen2.conf.unit_cells.SC
Bases: object
```

Methods

gen_box
numb_atoms
poscar_unit

```
gen_box()
numb_atoms()
poscar_unit(latt)
dpigen2.conf.unit_cells.generate_unit_cell(crystal: str, latt: float = 1.0) → System
```

dpigen2.entrypoint package

Submodules

dpigen2.entrypoint.args module

```
dpigen2.entrypoint.args.bohrium_conf_args()
dpigen2.entrypoint.args.default_step_config_args()
dpigen2.entrypoint.args.dflow_conf_args()
dpigen2.entrypoint.args.dp_dist_train_args()
dpigen2.entrypoint.args.dp_train_args()
dpigen2.entrypoint.args.dpgen_step_config_args(default_config)
dpigen2.entrypoint.args.fp_args(inputs, run)
dpigen2.entrypoint.args.gen_doc(*, make_anchor=True, make_link=True, **kwargs)
```

```
dpgen2.entrypoint.args.input_args()  
dpgen2.entrypoint.args.lmp_args()  
dpgen2.entrypoint.args.make_link(content, ref_key)  
dpgen2.entrypoint.args.normalize(data)  
dpgen2.entrypoint.args.submit_args(default_step_config={'continue_on_failed': False,  
    'continue_on_num_success': None, 'continue_on_success_ratio': None,  
    'executor': None, 'parallelism': None, 'template_config': {'envs': None,  
        'image': 'dptechnology/dpgen2:latest', 'retry_on_transient_error':  
            None, 'timeout': None, 'timeout_as_transient_error': False}})  
dpgen2.entrypoint.args.variant_conf()  
dpgen2.entrypoint.args.variant_conv()  
dpgen2.entrypoint.args.variant_explore()  
dpgen2.entrypoint.args.variant_fp()  
dpgen2.entrypoint.args.variant_train()
```

dpgen2.entrypoint.common module

```
dpgen2.entrypoint.common.expand_idx(in_list) → List[int]  
dpgen2.entrypoint.common.expand_sys_str(root_dir: str | Path) → List[str]  
dpgen2.entrypoint.common.global_config_workflow(wf_config)
```

dpgen2.entrypoint.download module

```
dpgen2.entrypoint.download.download(workflow_id, wf_config: Dict | None = {}, wf_keys: List | None =  
    None, prefix: str | None = None, chk_pnt: bool = False)  
dpgen2.entrypoint.download.download_by_def(workflow_id, wf_config: Dict = {}, iterations: List[int] |  
    None = None, step_defs: List[str] | None = None, prefix: str  
    | None = None, chk_pnt: bool = False)
```

dpgen2.entrypoint.gui module

DP-GUI entrypoint.

```
dpgen2.entrypoint.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

dpgen2.entrypoint.main module

dpgen2.entrypoint.main.main()
dpgen2.entrypoint.main.main_parser() → `ArgumentParser`
 DPGEN2 commandline options argument parser.
Returns
argparse.ArgumentParser
 the argument parser

Notes

This function is used by documentation.

dpgen2.entrypoint.main.parse_args(args: List[str] | None = None)
 DPGEN2 commandline options argument parsing.
Parameters

args
 [List[str]] list of command line arguments, main purpose is testing default option None
 takes arguments from sys.argv

dpgen2.entrypoint.showkey module

dpgen2.entrypoint.showkey.showkey(wf_id, wf_config)

dpgen2.entrypoint.status module

dpgen2.entrypoint.status.status(workflow_id, wf_config: Dict | None = {})

dpgen2.entrypoint.submit module

dpgen2.entrypoint.submit.copy_scheduler_plans(scheduler_new, scheduler_old)
dpgen2.entrypoint.submit.fold_keys(all_step_keys)
dpgen2.entrypoint.submit.get_kspacing_kgamma_from_incar(fname)
dpgen2.entrypoint.submit.get_resubmit_keys(wf)
dpgen2.entrypoint.submit.get_scheduler_ids(reuse_step)
dpgen2.entrypoint.submit.get_superop(key)

```
dpgen2.entrypoint.submit.make_calypso_naive_exploration_scheduler(config)
```

```
dpigen2.entrypoint.submit.make_concurrent_learning_op(train_style: str = 'dp', explore_style: str =
    'lmp', fp_style: str = 'vasp', prep_train_config:
    dict = {'continue_on_failed': False,
    'continue_on_num_success': None,
    'continue_on_success_ratio': None, 'executor':
    None, 'parallelism': None, 'template_config':
    {'envs': None, 'image':
    'dptechnology/dpigen2:latest',
    'retry_on_transient_error': None, 'timeout':
    None, 'timeout_as_transient_error': False}},
    run_train_config: dict = {'continue_on_failed':
    False, 'continue_on_num_success': None,
    'continue_on_success_ratio': None, 'executor':
    None, 'parallelism': None, 'template_config':
    {'envs': None, 'image':
    'dptechnology/dpigen2:latest',
    'retry_on_transient_error': None, 'timeout':
    None, 'timeout_as_transient_error': False}},
    prep_explore_config: dict =
    {'continue_on_failed': False,
    'continue_on_num_success': None,
    'continue_on_success_ratio': None, 'executor':
    None, 'parallelism': None, 'template_config':
    {'envs': None, 'image':
    'dptechnology/dpigen2:latest',
    'retry_on_transient_error': None, 'timeout':
    None, 'timeout_as_transient_error': False}},
    run_explore_config: dict =
    {'continue_on_failed': False,
    'continue_on_num_success': None,
    'continue_on_success_ratio': None, 'executor':
    None, 'parallelism': None, 'template_config':
    {'envs': None, 'image':
    'dptechnology/dpigen2:latest',
    'retry_on_transient_error': None, 'timeout':
    None, 'timeout_as_transient_error': False}},
    prep_fp_config: dict = {'continue_on_failed':
    False, 'continue_on_num_success': None,
    'continue_on_success_ratio': None, 'executor':
    None, 'parallelism': None, 'template_config':
    {'envs': None, 'image':
    'dptechnology/dpigen2:latest',
    'retry_on_transient_error': None, 'timeout':
    None, 'timeout_as_transient_error': False}},
    run_fp_config: dict = {'continue_on_failed':
    False, 'continue_on_num_success': None,
    'continue_on_success_ratio': None, 'executor':
    None, 'parallelism': None, 'template_config':
    {'envs': None, 'image':
    'dptechnology/dpigen2:latest',
    'retry_on_transient_error': None, 'timeout':
    None, 'timeout_as_transient_error': False}},
    select_confs_config: dict =
    {'continue_on_failed': False,
    'continue_on_num_success': None,
    'continue_on_success_ratio': None, 'executor':
    None, 'parallelism': None, 'template_config':
    {'envs': None, 'image':
    'dptechnology/dpigen2:latest',
    'retry_on_transient_error': None, 'timeout':
    None, 'timeout_as_transient_error': False}}
```

```
dpgen2.entrypoint.submit.make_finetune_step(config, prep_train_config, run_train_config,
                                             upload_python_packages, numb_models, template_script,
                                             train_config, init_models, init_data, iter_data,
                                             valid_data=None)

dpgen2.entrypoint.submit.make_lmp_naive_exploration_scheduler(config)

dpgen2.entrypoint.submit.make_naive_exploration_scheduler(config)

dpgen2.entrypoint.submit.make_optional_parameter(mixed_type=False, finetune_mode='no')

dpgen2.entrypoint.submit.print_list_steps(steps)

dpgen2.entrypoint.submit.resubmit_concurrent_learning(wf_config, wfid, list_steps=False,
                                                       reuse=None, replace_scheduler=False,
                                                       fold=False)

dpgen2.entrypoint.submit.submit_concurrent_learning(wf_config, reuse_step: List[ArgoStep] | None = None,
                                                    replace_scheduler: bool = False,
                                                    no_submission: bool = False)

dpgen2.entrypoint.submit.successful_step_keys(wf)

dpgen2.entrypoint.submit.update_reuse_step_scheduler(reuse_step, scheduler_new)

dpgen2.entrypoint.submit.workflow_concurrent_learning(config: Dict) → Tuple[Step, Step | None]
```

dpgen2.entrypoint.watch module

```
dpgen2.entrypoint.watch.update_finished_steps(wf, finished_keys: List[str] | None = None, download: bool | None = False, watching_keys: List[str] | None = None, prefix: str | None = None, chk_pnt: bool = False)

dpgen2.entrypoint.watch.watch(workflow_id, wf_config: Dict | None = {}, watching_keys: List | None = ['prep-run-train', 'prep-run-explore', 'prep-run-fp', 'collect-data'], frequency: float = 600.0, download: bool = False, prefix: str | None = None, chk_pnt: bool = False)
```

dpgen2.entrypoint.workflow module

```
dpgen2.entrypoint.workflow.add_subparser_workflow_subcommand(subparsers, command: str)

dpgen2.entrypoint.workflow.execute_workflow_subcommand(command: str, wfid: str, wf_config: dict | None = {})
```

dpgen2.exploration package**Subpackages****dpgen2.exploration.deviation package****Submodules****dpgen2.exploration.deviation.deviation_manager module**

```
class dpgen2.exploration.deviation.deviation_manager.DeviManager
```

Bases: ABC

A class for model deviation management.

Methods

<code>add(name, deviation)</code>	Add a model deviation into this manager.
<code>clear()</code>	Clear all data in this manager.
<code>get(name)</code>	Gat a model deviation from this manager.

```
AVG_DEVI_F = 'avg_devi_f'
```

```
AVG_DEVI_V = 'avg_devi_v'
```

```
MAX_DEVI_F = 'max_devi_f'
```

```
MAX_DEVI_V = 'max_devi_v'
```

```
MIN_DEVI_F = 'min_devi_f'
```

```
MIN_DEVI_V = 'min_devi_v'
```

```
add(name: str, deviation: ndarray) → None
```

Add a model deviation into this manager.

Parameters**name**

[str] The name of the deviation. The name is restricted to (DeviManager.MAX_DEVI_V, DeviManager.MIN_DEVI_V, DeviManager.AVG_DEVI_V, DeviManager.MAX_DEVI_F, DeviManager.MIN_DEVI_F, DeviManager.AVG_DEVI_F)

deviation

[np.ndarray] The model deviation is a one-dimensional array extracted from a trajectory file.

```
abstract clear() → None
```

Clear all data in this manager.

```
get(name: str) → List[ndarray | None]
```

Gat a model deviation from this manager.

Parameters

name

[str] The name of the deviation. The name is restricted to (DeviManager.MAX_DEVI_V, DeviManager.MIN_DEVI_V,

DeviManager.AVG_DEVI_V, DeviManager.MAX_DEVI_F, DeviManager.MIN_DEVI_F, DeviManager.AVG_DEVI_F)

dpgen2.exploration.deviation.deviation_std module**class dpgen2.exploration.deviation.deviation_std.DeviManagerStd**

Bases: *DeviManager*

The class which is responsible for model deviation management.

This is the standard implementation of DeviManager. Each deviation (e.g. max_devi_f, max_devi_v in file *model_devi.out*) is stored as a List[Optional[np.ndarray]], where np.array is a one-dimensional array. A List[np.ndarray][ii][jj] is the force model deviation of the jj-th frame of the ii-th trajectory. The model deviation can be List[None], where len(List[None]) is the number of trajectory files.

Methods

add(name, deviation)	Add a model deviation into this manager.
<i>clear()</i>	Clear all data in this manager.
get(name)	Gat a model deviation from this manager.

clear() → None

Clear all data in this manager.

dpgen2.exploration.render package**Submodules****dpgen2.exploration.render.traj_render module****class dpgen2.exploration.render.traj_render.TrajRender**

Bases: ABC

Methods

<i>get_confs(traj, id_selected[, type_map, ...])</i>	Get configurations from trajectory by selection.
<i>get_model_devi(files)</i>	Get model deviations from recording files.

abstract get_confs(*traj*: List[Path], *id_selected*: List[List[int]], *type_map*: List[str] | None = None, *conf_filters*: ConfFilters | None = None) → MultiSystems

Get configurations from trajectory by selection.

Parameters

traj

[List[Path]] Trajectory files

id_selected

[List[List[int]]] The selected frames. id_selected[ii][jj] is the jj-th selected frame from the ii-th trajectory. id_selected[ii] may be an empty list.

type_map

[List[str]] The type map.

Returns**ms: dpdata.MultiSystems**

The configurations in dpdata.MultiSystems format

abstract get_model_devi(files: List[Path]) → DeviManager

Get model deviations from recording files.

Parameters**files**

[List[Path]] The paths to the model deviation recording files

Returns**DeviManager:** The class which is responsible for model deviation management.**dpgen2.exploration.render.traj_render_lammps module****class dpgen2.exploration.render.traj_render_lammps.TrajRenderLammps(nopbc: bool = False)**Bases: *TrajRender***Methods**

<i>get_confs(trajs, id_selected[, type_map, ...])</i>	Get configurations from trajectory by selection.
<i>get_model_devi(files)</i>	Get model deviations from recording files.

get_confs(trajs: List[Path], id_selected: List[List[int]], type_map: List[str] | None = None, conf_filters: ConfFilters | None = None) → MultiSystems

Get configurations from trajectory by selection.

Parameters**traj**

[List[Path]] Trajectory files

id_selected

[List[List[int]]] The selected frames. id_selected[ii][jj] is the jj-th selected frame from the ii-th trajectory. id_selected[ii] may be an empty list.

type_map

[List[str]] The type map.

Returns**ms: dpdata.MultiSystems**

The configurations in dpdata.MultiSystems format

get_model_devi(files: *List[Path]*) → *DeviManager*

Get model deviations from recording files.

Parameters**files**

[List[Path]] The paths to the model deviation recording files

Returns**DeviManager:** The class which is responsible for model deviation management.**dpgen2.exploration.report package****Submodules****dpgen2.exploration.report.report module****class dpgen2.exploration.report.report.ExplorationReport**

Bases: ABC

Methods

<i>clear()</i>	Clear the report
<i>converged(reports)</i>	Check if the exploration is converged.
<i>get_candidate_ids([max_nframes])</i>	Get indexes of candidate configurations
<i>no_candidate()</i>	If no candidate configuration is found
<i>print(stage_idx, idx_in_stage, iter_idx)</i>	Print the report
<i>print_header()</i>	Print the header of report
<i>record(model_devi)</i>	Record the model deviations of the trajectories

abstract clear()

Clear the report

abstract converged(reports) → bool

Check if the exploration is converged.

Parameters**reports**

Historical reports

Returns**converged bool**

If the exploration is converged.

abstract get_candidate_ids(max_nframes: int | None = None) → List[List[int]]

Get indexes of candidate configurations

Parameters**max_nframes**

The maximal number of frames of candidates.

Returns

idx: List[List[int]]

The frame indices of candidate configurations. idx[ii][jj] is the frame index of the jj-th candidate of the ii-th trajectory.

no_candidate() → bool

If no candidate configuration is found

abstract print(stage_idx: int, idx_in_stage: int, iter_idx: int) → str

Print the report

abstract print_header() → str

Print the header of report

abstract record(model_devi: DeviManager)

Record the model deviations of the trajectories

Parameters**model_devi**

[DeviManager] The class which is responsible for model deviation management. Model deviations is stored as a List[Optional[np.ndarray]], where np.array is a one-dimensional array. List[np.ndarray][ii][jj] is the force model deviation of the jj-th frame of the ii-th trajectory. Model deviations can be List[None], where len(List[None]) is the number of trajectory files.

dpgen2.exploration.report.report_adaptive_lower module

```
class dpgen2.exploration.report.report_adaptive_lower.ExplorationReportAdaptiveLower(level_f_hi:
    float
    =
    0.5,
    numb_candi_f:
    int =
    200,
    rate_candi_f:
    float
    =
    0.01,
    level_v_hi:
    float |
    None
    =
    None,
    numb_candi_v:
    int =
    0,
    rate_candi_v:
    float
    =
    0.0,
    n_checked_steps:
    int =
    2,
    conv_tolerance:
    float
    =
    0.05,
    candi_sel_prob:
    str =
    'uni-
    form')
```

Bases: *ExplorationReport*

The exploration report that adapts the lower trust level.

This report will treat a fixed number of frames that has force model deviation lower than *level_f_hi*, and virial model deviation lower than *level_v_hi* as candidates.

The number of force frames is given by $\max(\text{numb_candi_f}, \text{rate_candi_f} * \text{nframes})$ The number of virial frames is given by $\max(\text{numb_candi_v}, \text{rate_candi_v} * \text{nframes})$

The lower force trust level will be set to the lowest force model deviation of the force frames. The lower virial trust level will be set to the lowest virial model deviation of the virial frames

The exploration will be treated as converged if the differences in model deviations in the neighboring steps are less than *conv_tolerance* in the last *n_checked_steps*.

Parameters

level_f_hi float

The higher trust level of force model deviation

numb_candi_f int

The number of force frames that has a model deviation lower than *level_f_hi* treated as candidate.

rate_candi_f float

The ratio of force frames that has a model deviation lower than *level_f_hi* treated as candidate.

level_v_hi float

The higher trust level of virial model deviation

numb_candi_v int

The number of virial frames that has a model deviation lower than *level_v_hi* treated as candidate.

rate_candi_v float

The ratio of virial frames that has a model deviation lower than *level_v_hi* treated as candidate.

n_checked_steps int

The number of steps to check the convergence.

conv_tolerance float

The convergence tolerance.

candi_sel_prob str

The method for selecting candidates. It can be “uniform”: all candidates are of the same probability. “inv_pop_f” or “inv_pop_f:nhist”: the probability is inversely proportional to the population of a histogram between *level_f_lo* and *level_f_hi*. The number of bins in the histogram is set by *nhist*, which should be an integer. The default is 10.

Methods

<code>clear()</code>	Clear the report
<code>converged(reports)</code>	Check if the exploration is converged.
<code>get_candidate_ids([max_nframes])</code>	Get indexes of candidate configurations
<code>no_candidate()</code>	If no candidate configuration is found
<code>print(stage_idx, idx_in_stage, iter_idx)</code>	Print the report
<code>print_header()</code>	Print the header of report
<code>record(model_devi)</code>	Record the model deviations of the trajectories

accurate_ratio**args****candidate_ratio****doc****failed_ratio**

accurate_ratio(tag=None)

static args() → List[Argument]

candidate_ratio(tag=None)

clear()

Clear the report

converged(*reports*) → bool

Check if the exploration is converged.

Parameters**reports**

Historical reports

Returns**converged** bool

If the exploration is converged.

static doc() → str**failed_ratio**(*tag=None*)**get_candidate_ids**(*max_nframes: int | None = None*) → List[List[int]]

Get indexes of candidate configurations

Parameters**max_nframes**

The maximal number of frames of candidates.

Returns**idx: List[List[int]]**

The frame indices of candidate configurations. idx[ii][jj] is the frame index of the jj-th candidate of the ii-th trajectory.

print(*stage_idx: int, idx_in_stage: int, iter_idx: int*) → str

Print the report

print_header() → str

Print the header of report

record(*model_devi: DeviManager*)

Record the model deviations of the trajectories

Parameters**model_devi**

[DeviManager] The class which is responsible for model deviation management. Model deviations is stored as a List[Optional[np.ndarray]], where np.array is a one-dimensional array. List[np.ndarray][ii][jj] is the force model deviation of the jj-th frame of the ii-th trajectory. Model deviations can be List[None], where len(List[None]) is the number of trajectory files.

dpgen2.exploration.report.report_trust_levels_base module**class dpgen2.exploration.report.report_trust_levels_base.ExplorationReportTrustLevels**(*level_f_lo, level_f_hi, level_v_lo=None, level_v_hi=None, conv_accuracy=0.9*)Bases: *ExplorationReport*

Methods

<code>clear()</code>	Clear the report
<code>converged([reports])</code>	Check if the exploration is converged.
<code>get_candidate_ids([max_nframes])</code>	Get indexes of candidate configurations
<code>no_candidate()</code>	If no candidate configuration is found
<code>print(stage_idx, idx_in_stage, iter_idx)</code>	Print the report
<code>print_header()</code>	Print the header of report
<code>record(model_devi)</code>	Record the model deviations of the trajectories

accurate_ratio
args
candidate_ratio
failed_ratio

accurate_ratio(tag=None)

static args() → List[Argument]

candidate_ratio(tag=None)

clear()

Clear the report

abstract converged(reports: List[ExplorationReport] | None = None) → bool

Check if the exploration is converged.

Parameters

reports

Historical reports

Returns

converged bool

If the exploration is converged.

failed_ratio(tag=None)

abstract get_candidate_ids(max_nframes: int | None = None) → List[List[int]]

Get indexes of candidate configurations

Parameters

max_nframes

The maximal number of frames of candidates.

Returns

idx: List[List[int]]

The frame indices of candidate configurations. idx[ii][jj] is the frame index of the jj-th candidate of the ii-th trajectory.

print(stage_idx: int, idx_in_stage: int, iter_idx: int) → str

Print the report

print_header() → str
Print the header of report

record(model_devi: DeviManager)
Record the model deviations of the trajectories

Parameters**model_devi**

[DeviManager] The class which is responsible for model deviation management. Model deviations is stored as a List[Optional[np.ndarray]], where np.array is a one-dimensional array. List[np.ndarray][ii][jj] is the force model deviation of the jj-th frame of the ii-th trajectory. Model deviations can be List[None], where len(List[None]) is the number of trajectory files.

dpgen2.exploration.report.report_trust_levels_max module

```
class dpgen2.exploration.report.report_trust_levels_max.ExplorationReportTrustLevelsMax(level_f_lo,
                                         level_f_hi,
                                         level_v_lo=None,
                                         level_v_hi=None,
                                         conv_accuracy=0.9)
```

Bases: *ExplorationReportTrustLevels*

Methods

clear()	Clear the report
converged([reports])	Check if the exploration is converged.
get_candidate_ids([max_nframes])	Get indexes of candidate configurations
no_candidate()	If no candidate configuration is found
print(stage_idx, idx_in_stage, iter_idx)	Print the report
print_header()	Print the header of report
record(model_devi)	Record the model deviations of the trajectories

accurate_ratio
args
candidate_ratio
doc
failed_ratio

converged(reports: List[ExplorationReport] | None = None) → bool

Check if the exploration is converged.

Parameters**reports**

Historical reports

Returns**converged bool**

If the exploration is converged.

```
static doc() → str
get_candidate_ids(max_nframes: int | None = None) → List[List[int]]
```

Get indexes of candidate configurations

Parameters

max_nframes

The maximal number of frames of candidates.

Returns

idx: List[List[int]]

The frame indices of candidate configurations. idx[ii][jj] is the frame index of the jj-th candidate of the ii-th trajectory.

dpgen2.exploration.report.report_trust_levels_random module

```
class dpgen2.exploration.report.report_trust_levels_random.ExplorationReportTrustLevelsRandom(level_f_lo,
level_f_hi,
level_v_lo=
level_v_hi=
conv_accuracy=
```

Bases: *ExplorationReportTrustLevels*

Methods

clear()	Clear the report
converged([reports])	Check if the exploration is converged.
get_candidate_ids([max_nframes])	Get indexes of candidate configurations
no_candidate()	If no candidate configuration is found
print(stage_idx, idx_in_stage, iter_idx)	Print the report
print_header()	Print the header of report
record(model_devi)	Record the model deviations of the trajectories

accurate_ratio
args
candidate_ratio
doc
failed_ratio

converged(reports: List[ExplorationReport] | None = None) → bool

Check if the exploration is converged.

Parameters

reports

Historical reports

Returns

converged bool

If the exploration is converged.

```
static doc() → str
```

```
get_candidate_ids(max_nframes: int | None = None) → List[List[int]]
```

Get indexes of candidate configurations

Parameters

max_nframes

The maximal number of frames of candidates.

Returns

idx: List[List[int]]

The frame indices of candidate configurations. idx[ii][jj] is the frame index of the jj-th candidate of the ii-th trajectory.

dpgen2.exploration.scheduler package

Submodules

dpgen2.exploration.scheduler.convergence_check_stage_scheduler module

```
class dpgen2.exploration.scheduler.convergence_check_stage_scheduler.ConvergenceCheckStageScheduler(stage: Stage, selector: Selector, max_nframes: int | None = None, fail_talent: bool = True)
```

Bases: *StageScheduler*

Methods

<code>complete()</code>	Tell if the stage is complete
<code>converged()</code>	Tell if the stage is converged
<code>force_complete()</code>	For complete the stage
<code>get_reports()</code>	Return all exploration reports
<code>next_iteration()</code>	Return the index of the next iteration
<code>plan_next_iteration([report, trajs])</code>	Make the plan for the next iteration of the stage.

reached_max_iteration

`complete()`

Tell if the stage is complete

Returns

converged bool

if the stage is complete

`converged()`

Tell if the stage is converged

Returns

converged bool

the convergence

`force_complete()`

For complete the stage

`get_reports()`

Return all exploration reports

Returns

reports List[ExplorationReport]

the reports

`next_iteration()`

Return the index of the next iteration

Returns

index int

the index of the next iteration

`plan_next_iteration(report: ExplorationReport | None = None, trajs: List[Path] | None = None) → Tuple[bool, BaseExplorationTaskGroup | None, ConfSelector | None]`

Make the plan for the next iteration of the stage.

It checks the report of the current and all historical iterations of the stage, and tells if the iterations are converged. If not converged, it will plan the next iteration for the stage.

Parameters

hist_reports

[List[ExplorationReport]] The historical exploration report of the stage. If this is the first iteration of the stage, this list is empty.

report

[ExplorationReport] The exploration report of this iteration.

conf

[List[Path]] A list of configurations generated during the exploration. May be used to generate new configurations for the next iteration.

Returns**stg_complete: bool**

If the stage completed. Two cases may happen: 1. converged. 2. when not fatal_at_max, not converged but reached max number of iterations.

task: ExplorationTaskGroup

A *ExplorationTaskGroup* defining the exploration of the next iteration. Should be *None* if the stage is converged.

conf_selector: ConfSelector

The configuration selector for the next iteration. Should be *None* if the stage is converged.

reached_max_iteration()**dpgen2.exploration.scheduler.scheduler module****class dpgen2.exploration.scheduler.scheduler.ExplorationScheduler**

Bases: `object`

The exploration scheduler.

Methods

<code>add_stage_scheduler(stage_scheduler)</code>	Add stage scheduler.
<code>complete()</code>	Tell if all stages are converged.
<code>force_stage_complete()</code>	Force complete the current stage
<code>get_convergence_ratio()</code>	Get the accurate, candidate and failed ratios of the iterations
<code>get_iteration()</code>	Get the index of the current iteration.
<code>get_stage()</code>	Get the index of current stage.
<code>get_stage_of_iterations()</code>	Get the stage index and the index in the stage of iterations.
<code>plan_next_iteration([report, trajs])</code>	Make the plan for the next DPGEN iteration.

`print_convergence`
`print_last_iteration`

add_stage_scheduler(stage_scheduler: StageScheduler)

Add stage scheduler.

All added schedulers can be treated as a *list* (order matters). Only one stage is converged, the iteration goes to the next iteration.

Parameters**stage_scheduler**

[StageScheduler] The added stage scheduler

complete()

Tell if all stages are converged.

force_stage_complete()

Force complete the current stage

get_convergence_ratio()

Get the accurate, candidate and failed ratios of the iterations

Returns**accu np.ndarray**

The accurate ratio. length of array the same as # iterations.

cand np.ndarray

The candidate ratio. length of array the same as # iterations.

fail np.ndarray

The failed ration. length of array the same as # iterations.

get_iteration()

Get the index of the current iteration.

Iteration index increase when *self.plan_next_iteration* returns valid *expl_task_grp* and *conf_selector* for the next iteration.

get_stage()

Get the index of current stage.

Stage index increases when the previous stage converges. Usually called after *self.plan_next_iteration*.

get_stage_of_iterations()

Get the stage index and the index in the stage of iterations.

plan_next_iteration(*report*: ExplorationReport | *None* = *None*, *trajs*: List[Path] | *None* = *None*) → Tuple[bool, ExplorationTaskGroup | *None*, ConfSelector | *None*]

Make the plan for the next DPGEN iteration.

Parameters**report**

[ExplorationReport] The exploration report of this iteration.

trajs

[List[Path]] A list of configurations generated during the exploration. May be used to generate new configurations for the next iteration.

Returns**complete: bool**

If all the DPGEN stages complete.

task: ExplorationTaskGroup

A *ExplorationTaskGroup* defining the exploration of the next iteration. Should be *None* if converged.

conf_selector: ConfSelector

The configuration selector for the next iteration. Should be *None* if converged.

print_convergence()**print_last_iteration(*print_header*=*False*)**

dpgen2.exploration.scheduler.stage_scheduler module**class dpgen2.exploration.scheduler.stage_scheduler.StageScheduler**

Bases: ABC

The scheduler for an exploration stage.

Methods

<code>complete()</code>	Tell if the stage is complete
<code>converged()</code>	Tell if the stage is converged
<code>force_complete()</code>	For complete the stage
<code>get_reports()</code>	Return all exploration reports
<code>next_iteration()</code>	Return the index of the next iteration
<code>plan_next_iteration(report, trajs)</code>	Make the plan for the next iteration of the stage.

abstract complete() → bool

Tell if the stage is complete

Returns**converged bool**

if the stage is complete

abstract converged() → bool

Tell if the stage is converged

Returns**converged bool**

the convergence

abstract force_complete()

For complete the stage

abstract get_reports() → List[ExplorationReport]

Return all exploration reports

Returns**reports List[ExplorationReport]**

the reports

abstract next_iteration() → int

Return the index of the next iteration

Returns**index int**

the index of the next iteration

abstract plan_next_iteration(report: ExplorationReport, trajs: List[Path]) → Tuple[bool, ExplorationTaskGroup, ConfSelector]

Make the plan for the next iteration of the stage.

It checks the report of the current and all historical iterations of the stage, and tells if the iterations are converged. If not converged, it will plan the next iteration for the stage.

Parameters

hist_reports

[List[ExplorationReport]] The historical exploration report of the stage. If this is the first iteration of the stage, this list is empty.

report

[ExplorationReport] The exploration report of this iteration.

confs

[List[Path]] A list of configurations generated during the exploration. May be used to generate new configurations for the next iteration.

Returns**stg_complete: bool**

If the stage completed. Two cases may happen: 1. converged. 2. when not fatal_at_max, not converged but reached max number of iterations.

task: ExplorationTaskGroup

A *ExplorationTaskGroup* defining the exploration of the next iteration. Should be *None* if the stage is converged.

conf_selector: ConfSelector

The configuration selector for the next iteration. Should be *None* if the stage is converged.

dpgen2.exploration.selector package**Submodules****dpgen2.exploration.selector.conf_filter module****class dpgen2.exploration.selector.conf_filter.ConfFilter**

Bases: ABC

Methods

<code>check(coords, cell, atom_types, nopbc)</code>	Check if the configuration is valid.
---	--------------------------------------

abstract check(coords: ndarray, cell: ndarray, atom_types: ndarray, nopbc: bool) → bool

Check if the configuration is valid.

Parameters**coords**

[numpy.array] The coordinates, numpy array of shape natoms x 3

cell

[numpy.array] The cell tensor. numpy array of shape 3 x 3

atom_types

[numpy.array] The atom types. numpy array of shape natoms

nopbc

[bool] If no periodic boundary condition.

Returns

valid

[bool] *True* if the configuration is a valid configuration, else *False*.

class dpgen2.exploration.selector.conf_filter.ConfFilters

Bases: `object`

Methods

add
check

add(*conf_filter*: ConfFilter) → ConfFilters

check(*conf*: System) → System

dpgen2.exploration.selector.conf_selector module

class dpge2.exploration.selector.conf_selector.ConfSelector

Bases: ABC

Select configurations from trajectory and model deviation files.

Methods

select

abstract select(*trajs*: List[Path], *model_devis*: List[Path], *type_map*: List[str] | None = None) → Tuple[List[Path], ExplorationReport]

dpgen2.exploration.selector.conf_selector_frame module

class dpge2.exploration.selector.conf_selector_frame.ConfSelectorFrames(*traj_render*: TrajRender, *report*: ExplorationReport, *max numb sel*: int | None = None, *conf filters*: ConfFilters | None = None)

Bases: ConfSelector

Select frames from trajectories as confs.

Parameters: trust_level: TrustLevel

The trust level

conf_filter: ConfFilters

The configuration filter

Methods

<code>select(trajs, model_devis[, type_map])</code>	Select configurations
---	-----------------------

`select(trajs: List[Path], model_devis: List[Path], type_map: List[str] | None = None) → Tuple[List[Path], ExplorationReport]`

Select configurations

Parameters

trajs

[List[Path]] A *list* of *Path* to trajectory files generated by LAMMPS

model_devis

[List[Path]] A *list* of *Path* to model deviation files generated by LAMMPS. Format: each line has 7 numbers they are used as # frame_id md_v_max md_v_min md_v_mean md_f_max md_f_min md_f_mean where *md* stands for model deviation, *v* for virial and *f* for force

type_map

[List[str]] The *type_map* of the systems

Returns

conf

[List[Path]] The selected configurations, stored in a folder in deepmd/npy format, can be parsed as dpdata.MultiSystems. The *list* only has one item.

report

[ExplorationReport] The exploration report recording the status of the exploration.

dpgen2.exploration.task package

Subpackages

dpgen2.exploration.task.calypso package

Submodules

dpgen2.exploration.task.calypso.caly_input module

```
dpgen2.exploration.task.calypso.caly_input.make_calypso_input(numb_of_species: int,  
                                                               name_of_atoms: List[str],  
                                                               atomic_number: List[int],  
                                                               numb_of_atoms: List[int],  
                                                               distance_of_ions, pop_size: int =  
                                                               30, max_step: int = 5,  
                                                               system_name: str = 'CALYPSO',  
                                                               numb_of_formula: List[int] = [1,  
                                                               1], pressure: float = 0.001, fmax:  
                                                               float = 0.01, volume: float = 0,  
                                                               ialgo: int = 2, pso_ratio: float =  
                                                               0.6, icode: int = 15,  
                                                               numb_of_lbest: int = 4,  
                                                               numb_of_local_optim: int = 4,  
                                                               command: str = 'sh submit.sh',  
                                                               max_time: int = 9000, gen_type:  
                                                               int = 1, pick_up: bool = False,  
                                                               pick_step: int = 1, parallel: bool =  
                                                               False, split: bool = True,  
                                                               spec_space_group: List[int] = [2,  
                                                               230], vsc: bool = False, ctrl_range:  
                                                               List[List[int]] = [[1, 10]],  
                                                               max_numb_atoms: int = 100,  
                                                               **kwargs)
```

dpgen2.exploration.task.lmp package

Submodules

dpgen2.exploration.task.lmp.lmp_input module

```
dpgen2.exploration.task.lmp.lmp_input.make_lmp_input(conf_file: str, ensemble: str, graphs: List[str],  
                                                       nsteps: int, dt: float, neidelay: int | None,  
                                                       trj_freq: int, mass_map: List[float], temp:  
                                                       float, tau_t: float = 0.1, pres: float | None =  
                                                       None, tau_p: float = 0.5, use_clusters: bool =  
                                                       False, relative_f_epsilon: float | None = None,  
                                                       relative_v_epsilon: float | None = None, pka_e:  
                                                       float | None = None, ele_temp_f: float | None =  
                                                       None, ele_temp_a: float | None = None, nopbc:  
                                                       bool = False, max_seed: int = 1000000,  
                                                       deepmd_version='2.0',  
                                                       trj_seperate_files=True)
```

Submodules

dpgen2.exploration.task.caly_task_group module

class dpgen2.exploration.task.caly_task_group.CalyTaskGroup

Bases: *ExplorationTaskGroup*

Attributes

task_list

Get the *list* of *ExplorationTask*

Methods

add_group(group)	Add another group to the group.
add_task(task)	Add one task to the group.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.
make_task()	Make the CALYPSO task group.
set_params(numb_of_species, name_of_atoms, ...)	Set calypso parameters

clear

make_task() → ExplorationTaskGroup

Make the CALYPSO task group.

Returns

task_grp: ExplorationTaskGroup

Return one calypso task group.

set_params(numb_of_species, name_of_atoms, atomic_number, numb_of_atoms, distance_of_ions, pop_size: *int* = 30, max_step: *int* = 5, system_name: *str* = 'CALYPSO', numb_of_formula: *List[int]* = [1, 1], pressure: *float* = 0.001, fmax: *float* = 0.01, volume: *float* = 0, ialgo: *int* = 2, pso_ratio: *float* = 0.6, icode: *int* = 15, numb_of_lbest: *int* = 4, numb_of_local_optim: *int* = 4, command: *str* = 'sh submit.sh', max_time: *int* = 9000, gen_type: *int* = 1, pick_up: *bool* = False, pick_step: *int* = 1, parallel: *bool* = False, split: *bool* = True, spec_space_group: *List[int]* = [2, 230], vsc: *bool* = True, ctrl_range: *List[List[int]]* = [[1, 10]], max_numb_atoms: *int* = 100)

Set calypso parameters

dpgen2.exploration.task.conf_sampling_task_group module

class dpgen2.exploration.task.conf_sampling_task_group.ConfSamplingTaskGroup

Bases: *ExplorationTaskGroup*

Attributes

task_list

Get the *list* of *ExplorationTask*

Methods

<code>add_group(group)</code>	Add another group to the group.
<code>add_task(task)</code>	Add one task to the group.
<code>count(value)</code>	
<code>index(value, [start, [stop]])</code>	Raises ValueError if the value is not present.
<code>make_task()</code>	Make the task group.
<code>set_conf(conf_list[, n_sample, random_sample])</code>	Set the configurations of exploration

clear

`set_conf(conf_list: List[str], n_sample: int | None = None, random_sample: bool = False)`

Set the configurations of exploration

Parameters

`conf_list str`

A list of file contents

`n_sample int`

Number of samples drawn from the conf list each time `make_task` is called. If set to `None`, `n_sample` is set to length of the `conf_list`.

`random_sample bool`

If true the confs are randomly sampled, otherwise are consecutively sampled from the `conf_list`

dpgen2.exploration.task.customized_lmp_template_task_group module

class

`dpgen2.exploration.task.customized_lmp_template_task_group.CustomizedLmpTemplateTaskGroup`

Bases: `ConfSamplingTaskGroup`

Attributes

`task_list`

Get the *list* of `ExplorationTask`

Methods

<code>add_group(group)</code>	Add another group to the group.
<code>add_task(task)</code>	Add one task to the group.
<code>count(value)</code>	
<code>index(value, [start, [stop]])</code>	Raises ValueError if the value is not present.
<code>make_task()</code>	Make the task group.
<code>set_conf(conf_list[, n_sample, random_sample])</code>	Set the configurations of exploration
<code>set_lmp(numb_models, custom_shell_commands)</code>	Set lammps task.

clear

make_task() → *CustomizedLmpTemplateTaskGroup*

Make the task group.

set_lmp(*numb_models*: *int*, *custom_shell_commands*: *List[str]*, *revisions*: *dict* = {}, *traj_freq*: *int* = 10, *input_lmp_conf_name*: *str* = 'conf.lmp', *input_lmp_tmpl_name*: *str* = 'in.lammps', *input_plm_tmpl_name*: *str* | *None* = *None*, *input_extra_files*: *List[str]* = [], *output_dir_pattern*: *str* | *List[str]* = '*', *output_lmp_conf_name*: *str* = 'conf.lmp', *output_lmp_tmpl_name*: *str* = 'in.lammps', *output_plm_tmpl_name*: *str* | *None* = *None*) → *None*

Set lammps task.

Parameters

numb_models

[int] Number of models

custom_shell_commands

[str] Customized shell commands to be run for each configuration. The commands require *input_lmp_conf_name* as input conf file, *input_lmp_tmpl_name* and *input_plm_tmpl_name* as templates, and *input_extra_files* as extra input files. By running the commands a series folders in pattern *output_dir_pattern* are supposed to be generated, and each folder is supposed to contain a configuration file *output_lmp_conf_name*, a lammps template file *output_lmp_tmpl_name* and a plumed template file *output_plm_tmpl_name*.

revisions

[dict] Revision dictionary. Provided in {key: [enumerated values]} format

traj_freq

[int] Frequency along trajectory of checking model deviation

input_lmp_conf_name

[str] Input conf file name for the shell commands.

input_lmp_tmpl_name

[str] Template file name of lammps input

input_plm_tmpl_name

[str] Template file name of the plumed input

input_extra_files

[List[str]] Extra files that may be needed to execute the shell commands

output_dir_pattern

[Union[str, List[str]]] Pattern of resultant folders generated by the shell commands.

output_lmp_conf_name

[str] Generated conf file name.

output_lmp_tmpl_name

[str] Generated lmp input file name.

output_plm_tmpl_name

[str] Generated plm input file name.

dpgen2.exploration.task.lmp_template_task_group module**class** dpgen2.exploration.task.lmp_template_task_group.**LmpTemplateTaskGroup**Bases: *ConfSamplingTaskGroup***Attributes****task_list**Get the *list* of *ExplorationTask***Methods**

add_group(group)	Add another group to the group.
add_task(task)	Add one task to the group.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.
make_task()	Make the task group.
set_conf(conf_list[, n_sample, random_sample])	Set the configurations of exploration

clear
make_cont
set_lmp

make_cont(templates: *list*, revisions: *dict*)**make_task**() → *LmpTemplateTaskGroup*

Make the task group.

set_lmp(numb_models: *int*, lmp_template_fname: *str*, plm_template_fname: *str* | *None* = *None*, revisions: *dict* = {}, traj_freq: *int* = 10) → *None*dpgen2.exploration.task.lmp_template_task_group.**find_only_one_key**(lmp_lines, key)dpgen2.exploration.task.lmp_template_task_group.**revise_by_keys**(lmp_lines, keys, values)dpgen2.exploration.task.lmp_template_task_group.**revise_lmp_input_dump**(lmp_lines, trj_freq)dpgen2.exploration.task.lmp_template_task_group.**revise_lmp_input_model**(lmp_lines,
task_model_list,
trj_freq,
deepmd_version='1')dpgen2.exploration.task.lmp_template_task_group.**revise_lmp_input_plm**(lmp_lines, in_plm,
out_plm='output.plumed')

dpigen2.exploration.task.make_task_group_from_config module

```
dpigen2.exploration.task.make_task_group_from_config.caly_normalize(data)
dpigen2.exploration.task.make_task_group_from_config.caly_task_group_args()
dpigen2.exploration.task.make_task_group_from_config.caly_task_grp_args()
dpigen2.exploration.task.make_task_group_from_config.config_strip_confidx(config)
dpigen2.exploration.task.make_task_group_from_config.customized_lmp_template_task_group_args()
dpigen2.exploration.task.make_task_group_from_config.lmp_normalize(data)
dpigen2.exploration.task.make_task_group_from_config.lmp_task_group_args()
dpigen2.exploration.task.make_task_group_from_config.lmp_template_task_group_args()
dpigen2.exploration.task.make_task_group_from_config.make_calypso_task_group_from_config(config)
dpigen2.exploration.task.make_task_group_from_config.make_lmp_task_group_from_config( numb_models,
                                         mass_map,
                                         con-
                                         fig)

dpigen2.exploration.task.make_task_group_from_config.make_task_group_from_config( numb_models,
                                         mass_map,
                                         config)

dpigen2.exploration.task.make_task_group_from_config.npt_task_group_args()
dpigen2.exploration.task.make_task_group_from_config.variant_task_group()
```

dpigen2.exploration.task.npt_task_group module

class dpigen2.exploration.task.npt_task_group.NPTTaskGroup

Bases: *ConfSamplingTaskGroup*

Attributes

task_list

Get the *list* of *ExplorationTask*

Methods

add_group(group)	Add another group to the group.
add_task(task)	Add one task to the group.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.
make_task()	Make the LAMMPS task group.
set_conf(conf_list[, n_sample, random_sample])	Set the configurations of exploration
set_md(numb_models, mass_map, temps[, ...])	Set MD parameters

clear**make_task()** → *NPTTaskGroup*

Make the LAMMPS task group.

Returns**task_grp: ExplorationTaskGroup**

The returned lammps task group. The number of tasks is $nconf \times nT \times nP$. $nconf$ is set by n_sample parameter of *set_conf*. nT and nP are lengths of the *temps* and *press* parameters of *set_md*.

```
set_md(numb_models, mass_map, temps: List[float], press: List[float] | None = None, ens: str = 'npt', dt: float = 0.001, nsteps: int = 1000, trj_freq: int = 10, tau_t: float = 0.1, tau_p: float = 0.5, pka_e: float | None = None, neidelay: int | None = None, no_pbc: bool = False, use_clusters: bool = False, relative_f_epsilon: float | None = None, relative_v_epsilon: float | None = None, ele_temp_f: float | None = None, ele_temp_a: float | None = None)
```

Set MD parameters

dpgen2.exploration.task.stage module**class dpgen2.exploration.task.stage.ExplorationStage**Bases: *object*

The exploration stage.

Methods

<i>add_task_group</i>(grp)	Add an exploration group
<i>clear()</i>	Clear all exploration group.
<i>make_task()</i>	Make the LAMMPS task group.

add_task_group(grp: *ExplorationTaskGroup*)

Add an exploration group

Parameters**grp**

[ExplorationTaskGroup] The added exploration task group

clear()

Clear all exploration group.

make_task() → *BaseExplorationTaskGroup*

Make the LAMMPS task group.

Returns**task_grp: BaseExplorationTaskGroup**

The returned lammps task group. The number of tasks is equal to the summation of task groups defined by all the exploration groups added to the stage.

dpgen2.exploration.task.task module

class dpgen2.exploration.task.task.ExplorationTask

Bases: object

Define the files needed by an exploration task.

Examples

```
>>> # this example dumps all files needed by the task.
>>> files = exploration_task.files()
... for file_name, file_content in files.items():
...     with open(file_name, 'w') as fp:
...         fp.write(file_content)
```

Methods

<code>add_file(fname, fcont)</code>	Add file to the task
<code>files()</code>	Get all files for the task.

add_file(*fname: str, fcont: str*)

Add file to the task

Parameters

fname

[str] The name of the file

fcont

[str] The content of the file.

files() → Dict

Get all files for the task.

Returns

files

[dict] The dict storing all files for the task. The file name is a key of the dict, and the file content is the corresponding value.

dpgen2.exploration.task.task_group module

class dpgen2.exploration.task.task_group.BaseExplorationTaskGroup

Bases: Sequence

A group of exploration tasks. Implemented as a *list* of ExplorationTask.

Attributes

task_list

Get the *list* of ExplorationTask

Methods

<code>add_group(group)</code>	Add another group to the group.
<code>add_task(task)</code>	Add one task to the group.
<code>count(value)</code>	
<code>index(value, [start, [stop]])</code>	Raises ValueError if the value is not present.

clear

`add_group(group: ExplorationTaskGroup)`

 Add another group to the group.

`add_task(task: ExplorationTask)`

 Add one task to the group.

`clear() → None`

`property task_list: List[ExplorationTask]`

 Get the *list* of *ExplorationTask*

`class dpogen2.exploration.task.task_group.ExplorationTaskGroup`

Bases: `ABC, BaseExplorationTaskGroup`

Attributes

`task_list`

 Get the *list* of *ExplorationTask*

Methods

<code>add_group(group)</code>	Add another group to the group.
<code>add_task(task)</code>	Add one task to the group.
<code>count(value)</code>	
<code>index(value, [start, [stop]])</code>	Raises ValueError if the value is not present.
<code>make_task()</code>	Make the task group.

clear

`abstract make_task() → ExplorationTaskGroup`

 Make the task group.

`class dpogen2.exploration.task.task_group.FooTask(conf_name='conf.lmp', conf_cont='',
 inpu_name='in.lammps', inpu_cont='')`

Bases: `ExplorationTask`

Methods

<code>add_file(fname, fcont)</code>	Add file to the task
<code>files()</code>	Get all files for the task.

class `dpgen2.exploration.task.task_group.FooTaskGroup(numb_task)`

Bases: `BaseExplorationTaskGroup`

Attributes

`task_list`

Get the *list* of *ExplorationTask*

Methods

<code>add_group(group)</code>	Add another group to the group.
<code>add_task(task)</code>	Add one task to the group.
<code>count(value)</code>	
<code>index(value, [start, [stop]])</code>	Raises ValueError if the value is not present.

clear

`property task_list`

Get the *list* of *ExplorationTask*

dpgen2.flow package

Submodules

dpgen2.flow.dpgen_loop module

class `dpgen2.flow.dpgen_loop.ConcurrentLearning(name: str, block_op: ConcurrentLearningBlock, step_config: dict = {'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'executor': None, 'parallelism': None, 'template_config': {'envs': None, 'image': 'dptechology/dpgen2:latest', 'retry_on_transient_error': None, 'timeout': None, 'timeout_as_transient_error': False}}, upload_python_packages: List[PathLike] | None = None)`

Bases: `Steps`

Attributes

init_keys
input_artifacts
input_parameters
loop_keys
output_artifacts
output_parameters

Methods

add(step)	Add a step or a list of parallel steps to the steps
-----------	---

add_slices
convert_to_argo
convert_to_graph
copy
deepcopy
from_dict
from_graph
handle_key
run

```
property init_keys
property input_artifacts
property input_parameters
property loop_keys
property output_artifacts
property output_parameters

class dpigen2.flow.dpigen_loop.ConcurrentLearningLoop(name: str, block_op: ConcurrentLearningBlock,
                                                       step_config: dict = {'continue_on_failed': False,
                                                               'continue_on_num_success': None,
                                                               'continue_on_success_ratio': None, 'executor': None,
                                                               'parallelism': None, 'template_config': {'envs': None, 'image': 'dptechology/dpigen2:latest',
                                                               'retry_on_transient_error': None, 'timeout': None,
                                                               'timeout_as_transient_error': False}},
                                                       upload_python_packages: List[PathLike] | None = None)
```

Bases: [Steps](#)**Attributes**

input_artifacts
input_parameters
keys
output_artifacts
output_parameters

Methods

add(step)	Add a step or a list of parallel steps to the steps
-----------	---

add_slices
convert_to_argo
convert_to_graph
copy
deepcopy
from_dict
from_graph
handle_key
run

```
property input_artifacts
property input_parameters
property keys
property output_artifacts
property output_parameters

class dpgen2.flow.dpgen_loop.MakeBlockId(*args, **kwargs)
    Bases: OP
```

Attributes

```
key
workflow_name
```

Methods

execute(ip)	Run the OP
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

```
execute(ip: OPIO) → OPIO
    Run the OP

@classmethod get_input_sign()
    Get the signature of the inputs

@classmethod get_output_sign()
    Get the signature of the outputs

class dpgen2.flow.dpgen_loop.SchedulerWrapper(*args, **kwargs)
```

Bases: OP

Attributes

```
key
workflow_name
```

Methods

execute(ip)	Run the OP
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs

```
convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction
```

```
execute(ip: OPIO) → OPIO
```

Run the OP

```
@classmethod get_input_sign()
```

Get the signature of the inputs

```
@classmethod get_output_sign()
```

Get the signature of the outputs

```
dpgen2.flow.dpgen_loop.make_block_optional_parameter(cl_optional_parameter)
```

dpgen2.fp package

Submodules

dpgen2.fp.abacus module

```
class dpgen2.fp.abacus.FpOpAbacusInputs(input_file: str | Path, pp_files: Dict[str, str | Path],
                                         element_mass: Dict[str, float] | None = None, kpt_file: str | Path
                                         | None = None, orb_files: Dict[str, str | Path] | None = None,
                                         deepks_descriptor: str | Path | None = None, deepks_model: str | Path
                                         | None = None)
```

Bases: AbacusInputs

Methods

<code>get_mass(element_list)</code>	Get the mass of elements.
<code>read_inputf(inputf)</code>	Read INPUT and transfer to a dict.
<code>write_deepks()</code>	Check if INPUT is a deepks job, if yes, will return the deepks descriptor file name, else will return None.
<code>write_pporb(element_list)</code>	Based on element list, write the pp/orb files, and return a list of the filename.

<code>args</code>
<code>get_deepks_descriptor</code>
<code>get_deepks_model</code>
<code>get_input</code>
<code>get_orb</code>
<code>get_pp</code>
<code>set_deepks_descriptor</code>
<code>set_deepks_model</code>
<code>set_input</code>
<code>set_mass</code>
<code>set_orb</code>
<code>set_pp</code>
<code>write_input</code>
<code>write_kpt</code>

`static args()`

```
class dpgen2.fp.abacus.PrepFpOpAbacus(*args, **kwargs)
```

Bases: OP

Attributes

<code>key</code>
<code>workflow_name</code>

Methods

<code>execute(ip)</code>	Run the OP
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

<code>convert_to_graph</code>
<code>exec_sign_check</code>
<code>from_graph</code>
<code>function</code>
<code>get_info</code>
<code>get_input_artifact_link</code>
<code>get_input_artifact_storage_key</code>
<code>get_opio_info</code>
<code>get_output_artifact_link</code>
<code>get_output_artifact_storage_key</code>
<code>register_output_artifact</code>
<code>superfunction</code>

`execute(ip: OPIO) → OPIO`

Run the OP

`classmethod get_input_sign()`

Get the signature of the inputs

`classmethod get_output_sign()`

Get the signature of the outputs

`class dpgen2.fp.abacus.RunFpOpAbacus(*args, **kwargs)`

Bases: `OP`

Attributes

<code>key</code>
<code>workflow_name</code>

Methods

<code>execute(ip)</code>	Run the OP
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

args
convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

```
static args()
execute(ip: OPIO) → OPIO
    Run the OP
classmethod get_input_sign()
    Get the signature of the inputs
classmethod get_output_sign()
    Get the signature of the outputs
```

dpgen2.fp.deepmd module

Prep and Run Gaussian tasks.

```
class dpgen2.fp.deepmd.DeepmdInputs(**kwargs: Any)
    Bases: object
```

Methods

args

```
static args() → List[Argument]
class dpgen2.fp.deepmd.PrepDeepmd(*args, **kwargs)
    Bases: PrepFp
    Attributes
```

```
key
workflow_name
```

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs
<code>prep_task(conf_frame, inputs)</code>	Define how one Deepmd task is prepared.

<code>convert_to_graph</code>
<code>exec_sign_check</code>
<code>from_graph</code>
<code>function</code>
<code>get_info</code>
<code>get_input_artifact_link</code>
<code>get_input_artifact_storage_key</code>
<code>get_opio_info</code>
<code>get_output_artifact_link</code>
<code>get_output_artifact_storage_key</code>
<code>register_output_artifact</code>
<code>superfunction</code>

`prep_task(conf_frame: System, inputs)`

Define how one Deepmd task is prepared.

Parameters

`conf_frame`

[dodata.System] One frame of configuration in the dodata format.

`inputs`

[str or dict] This parameter is useless in deepmd.

`class dpogen2.fp.deepmd.RunDeepmd(*args, **kwargs)`

Bases: `RunFp`

Attributes

`key`

`workflow_name`

Methods

<code>args()</code>	The argument definition of the <code>run_task</code> method.
<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs
<code>input_files()</code>	The mandatory input files to run a Deepmd task.
<code>normalize_config([data, strict])</code>	Normalized the argument.
<code>optional_input_files()</code>	The optional input files to run a Deepmd task.
<code>run_task(teacher_model_path, out, log)</code>	Defines how one FP task runs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

static args() → List[Argument]

The argument definition of the *run_task* method.

Returns**arguments: List[dargs.Argument]**

List of dargs.Argument defines the arguments of *run_task* method.

input_files() → List[str]

The mandatory input files to run a Deepmd task.

Returns**files: List[str]**

A list of madatory input files names.

optional_input_files() → List[str]

The optional input files to run a Deepmd task.

Returns**files: List[str]**

A list of optional input files names.

run_task(teacher_model_path: BinaryFileInput, out: str, log: str) → Tuple[str, str]

Defines how one FP task runs

Parameters**command**

[str] The command of running Deepmd task

out

[str] The name of the output data file.

Returns**out_name: str**

The file name of the output data in the dpdata.LabeledSystem format.

log_name: str

The file name of the log.

dpgen2.fp.gaussian module

Prep and Run Gaussian tasks.

class dpgen2.fp.gaussian.**GaussianInputs**(***kwargs: Any*)

Bases: *object*

Methods

args()	The arguments of the GaussianInputs class.
---------------	--

static args() → List[Argument]

The arguments of the GaussianInputs class.

class dpgen2.fp.gaussian.**PrepGaussian**(*args, **kwargs)

Bases: *PrepFp*

Attributes

key

workflow_name

Methods

execute(ip)	Execute the OP.
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs
prep_task(conf_frame, inputs)	Define how one Gaussian task is prepared.

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

prep_task(conf_frame: System, inputs: GaussianInputs)

Define how one Gaussian task is prepared.

Parameters

conf_frame

[dpdata.System] One frame of configuration in the dpdata format.

inputs

[GaussianInputs] The GaussianInputs object handles all other input files of the task.

```
class dpgen2.fp.gaussian.RunGaussian(*args, **kwargs)
```

Bases: *RunFp*

Attributes

key

workflow_name

Methods

<i>args()</i>	The argument definition of the <i>run_task</i> method.
<i>execute(ip)</i>	Execute the OP.
<i>get_input_sign()</i>	Get the signature of the inputs
<i>get_output_sign()</i>	Get the signature of the outputs
<i>input_files()</i>	The mandatory input files to run a Gaussian task.
<i>normalize_config([data, strict])</i>	Normalized the argument.
<i>optional_input_files()</i>	The optional input files to run a Gaussian task.
<i>run_task(command, out)</i>	Defines how one FP task runs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

static args() → List[Argument]

The argument definition of the *run_task* method.

Returns

arguments: List[dargs.Argument]

List of dargs.Argument defines the arguments of *run_task* method.

input_files() → List[str]

The mandatory input files to run a Gaussian task.

Returns

files: List[str]

A list of mandatory input files names.

optional_input_files() → List[str]

The optional input files to run a Gaussian task.

Returns

files: List[str]

A list of optional input files names.

run_task(command: str, out: str) → Tuple[str, str]

Defines how one FP task runs

Parameters**command**

[str] The command of running gaussian task

out

[str] The name of the output data file.

Returns**out_name: str**

The file name of the output data in the dpdata.LabeledSystem format.

log_name: str

The file name of the log.

dpgen2.fp.prep_fp module**class dpgen2.fp.prep_fp.PrepFp(*args, **kwargs)**

Bases: OP, ABC

Prepares the working directories for first-principles (FP) tasks.

A list of (same length as ip[“conf”]) working directories containing all files needed to start FP tasks will be created. The paths of the directories will be returned as op[“task_paths”]. The identities of the tasks are returned as op[“task_names”].

Attributes**key****workflow_name****Methods**

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs
<code>prep_task(conf_frame, inputs)</code>	Define how one FP task is prepared.

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

execute(ip: OPIO) → OPIO

Execute the OP.

Parameters**ip**

[dict] Input dict with components:

- *config* : (*dict*) Should have *config[‘inputs’]*, which defines the input files of the FP task.
- *conf* : (*Artifact(List[Path])*) Configurations for the FP tasks. Stored in folders as deepmd/npy format. Can be parsed as dpdata.MultiSystems.

Returns**op**

[dict] Output dict with components:

- *task_names*: (*List[str]*) The name of tasks. Will be used as the identities of the tasks. The names of different tasks are different.
- *task_paths*: (*Artifact(List[Path])*) The prepared working paths of the tasks. Contains all input files needed to start the FP. The order of the Paths should be consistent with *op[“task_names”]*

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

abstract prep_task(conf_frame: System, inputs: Any)

Define how one FP task is prepared.

Parameters**conf_frame**

[dpdata.System] One frame of configuration in the dpdata format.

inputs

[Any] The class object handles all other input files of the task. For example, pseudopotential file, k-point file and so on.

dpgen2.fp.run_fp module

```
class dpgen2.fp.run_fp.RunFp(*args, **kwargs)
```

Bases: OP, ABC

Execute a first-principles (FP) task.

A working directory named *task_name* is created. All input files are copied or symbol linked to directory *task_name*. The FP command is executed from directory *task_name*. The *op[“labeled_data”]* in “deepmd/npy” format (HF5 in the future) provided by *dodata* will be created.

Attributes

key	
workflow_name	

Methods

args()	The argument definition of the <i>run_task</i> method.
execute(ip)	Execute the OP.
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs
input_files()	The mandatory input files to run a FP task.
normalize_config([data, strict])	Normalized the argument.
optional_input_files()	The optional input files to run a FP task.
run_task(**kwargs)	Defines how one FP task runs

convert_to_graph	
exec_sign_check	
from_graph	
function	
get_info	
get_input_artifact_link	
get_input_artifact_storage_key	
get_opio_info	
get_output_artifact_link	
get_output_artifact_storage_key	
register_output_artifact	
superfunction	

abstract static args() → List[Argument]

The argument definition of the *run_task* method.

Returns

arguments: List[dargs.Argument]

List of dargs.Argument defines the arguments of *run_task* method.

execute(ip: OPIO) → OPIO

Execute the OP.

Parameters

ip

[dict] Input dict with components:

- *config*: (*dict*) The config of FP task. Should have *config*[‘run’], which defines the runtime configuration of the FP task.
- *task_name*: (*str*) The name of task.
- *task_path*: (*Artifact(Path)*) The path that contains all input files prepared by *PrepFp*.

Returns

Output dict with components:

- *log*: (*Artifact(Path)*) The log file of FP.
- *labeled_data*: (*Artifact(Path)*) The path to the labeled data in “deepmd/npy” format provided by *dodata*.

Raises**TransientError**

On the failure of FP execution.

FatalError

When mandatory files are not found.

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

abstract input_files() → List[str]

The mandatory input files to run a FP task.

Returns**files: List[str]**

A list of mandatory input files names.

classmethod normalize_config(data: Dict = {}, strict: bool = True) → Dict

Normalized the argument.

Parameters**data**

[Dict] The input dict of arguments.

strict

[bool] Strictly check the arguments.

Returns**data: Dict**

The normalized arguments.

abstract optional_input_files() → List[str]

The optional input files to run a FP task.

Returns**files: List[str]**

A list of optional input files names.

abstract `run_task(**kwargs) → Tuple[str, str]`

Defines how one FP task runs

Parameters

****kwargs**

Keyword args defined by the developer. The fp/run_config session of the input file will be passed to this function.

Returns

out_name: str

The file name of the output data. Should be in dpdata.LabeledSystem format.

log_name: str

The file name of the log.

dpgen2.fp.vasp module

class `dpgen2.fp.vasp.PrepVasp(*args, **kwargs)`

Bases: `PrepFP`

Attributes

`key`

`workflow_name`

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs
<code>prep_task(conf_frame, vasp_inputs)</code>	Define how one Vasp task is prepared.

<code>convert_to_graph</code>
<code>exec_sign_check</code>
<code>from_graph</code>
<code>function</code>
<code>get_info</code>
<code>get_input_artifact_link</code>
<code>get_input_artifact_storage_key</code>
<code>get_opio_info</code>
<code>get_output_artifact_link</code>
<code>get_output_artifact_storage_key</code>
<code>register_output_artifact</code>
<code>superfunction</code>

`prep_task(conf_frame: System, vasp_inputs: VaspInputs)`

Define how one Vasp task is prepared.

Parameters

conf_frame

[dpdata.System] One frame of configuration in the dpdata format.

vasp_inputs

[VaspInputs] The VaspInputs object handles all other input files of the task.

```
class dpgen2.fp.vasp.RunVasp(*args, **kwargs)
```

Bases: *RunFp*

Attributes

key

workflow_name

Methods

<i>args()</i>	The argument definition of the <i>run_task</i> method.
<i>execute(ip)</i>	Execute the OP.
<i>get_input_sign()</i>	Get the signature of the inputs
<i>get_output_sign()</i>	Get the signature of the outputs
<i>input_files()</i>	The mandatory input files to run a vasp task.
<i>normalize_config([data, strict])</i>	Normalized the argument.
<i>optional_input_files()</i>	The optional input files to run a vasp task.
<i>run_task(command, out, log)</i>	Defines how one FP task runs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

static args()

The argument definition of the *run_task* method.

Returns

arguments: List[dargs.Argument]

List of dargs.Argument defines the arguments of *run_task* method.

input_files() → List[str]

The mandatory input files to run a vasp task.

Returns

files: List[str]

A list of mandatory input files names.

optional_input_files() → List[str]

The optional input files to run a vasp task.

Returns

files: List[str]

A list of optional input files names.

run_task(*command: str, out: str, log: str*) → Tuple[str, str]

Defines how one FP task runs

Parameters**command**

[str] The command of running vasp task

out

[str] The name of the output data file.

log

[str] The name of the log file

Returns**out_name: str**

The file name of the output data in the dpdata.LabeledSystem format.

log_name: str

The file name of the log.

dpgen2.fp.vasp_input module

class dpgen2.fp.vasp_input.VaspInputs(*kspacing: float | List[float], incar: str, pp_files: Dict[str, str], kgamma: bool = True*)

Bases: `object`

Attributes**incar_template****potcars****Methods**

args
incar_from_file
make_kpoints
make_potcar
normalize_config
potcars_from_file

```
static args()
incar_from_file(fname: str)
property incar_template
make_kpoints(box: ndarray) → str
make_potcar(atom_names) → str
static normalize_config(data={}, strict=True)
```

```

property potcars
potcars_from_file(dict_fnames: Dict[str, str])
dpgen2.fp.vasp_input.make_kspacing_kpoints(box, kspacing, kgamma)

```

dpgen2.op package

Submodules

dpgen2.op.collect_data module

```
class dpgen2.op.collect_data.CollectData(*args, **kwargs)
```

Bases: OP

Collect labeled data and add to the iteration dataset.

After running FP tasks, the labeled data are scattered in task directories. This OP collect the labeled data in one data directory and add it to the iteration data. The data generated by this iteration will be place in *ip[“name”]* subdirectory of the iteration data directory.

Attributes

key	
workflow_name	

Methods

execute(ip)	Execute the OP.
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

```
default_optional_parameter = {'mixed_type': False}
```

```
execute(ip: OPIO) → OPIO
```

Execute the OP. This OP collect data scattered in directories given by *ip[‘labeled_data’]* in to one *dp-data.Multisystems* and store it in a directory named *name*. This directory is appended to the list *iter_data*.

Parameters

ip

[dict] Input dict with components:

- *name*: (str) The name of this iteration. The data generated by this iteration will be place in a sub-directory of *name*.
- *labeled_data*: (Artifact(List[Path])) The paths of labeled data generated by FP tasks of the current iteration.
- *iter_data*: (Artifact(List[Path])) The data paths previous iterations.

Returns**Any**

Output dict with components: - *iter_data*: (Artifact(List[Path])) The data paths of previous and the current iteration data.

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

dpgen2.op.collect_run_caly module**class dpgen2.op.collect_run_caly.CollRunCaly(*args, **kwargs)**

Bases: OP

Execute CALYPSO to generate structures in work_dir.

Changing the work directory into *task_name*. All input files have been copied or symbol linked to this directory *task_name* by *PrepCalyInput*. The CALYPSO command is exectuted from directory *task_name*. The *caly.log* and the *work_dir* will be stored in *op["log"]* and *op["work_dir"]*, respectively.

Attributes**key****workflow_name****Methods**

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

calypso_args
convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
normalize_config
register_output_artifact
superfunction

```
static calypso_args()
execute(ip: OPIO) → OPIO
```

Execute the OP.

Parameters

ip

[dict] Input dict with components:

- *config*: (*dict*) The config of calypso task to obtain the command of calypso.
- *task_name*: (*str*) The name of the task (calypso_task.{idx}).
- *input_file*: (*Path*) The input file of the task (input.dat).
- *step*: (*Path*) The step file from last calypso run
- *results*: (*Path*) The results dir from last calypso run
- *opt_results_dir*: (*Path*) The results dir contains POSCAR* CONTCAR* OUTCAR* from last calypso run
- *qhull_input*: (*Path*) qhull input file *test_qconvex.in*

Returns

Any

Output dict with components: - *poscar_dir*: (*Path*) The dir contains POSCAR*.

- *task_name*: (*str*) The name of the task (calypso_task.{idx}).
- *input_file*: (*Path*) The input file of the task (input.dat).
- *step*: (*Path*) The step file.
- *results*: (*Path*) The results dir.
- *qhull_input*: (*Path*) qhull input file.

Raises

TransientError

On the failure of CALYPSO execution. Resubmit rule should be clear.

```
classmethod get_input_sign()
```

Get the signature of the inputs

```
classmethod get_output_sign()
    Get the signature of the outputs
static normalize_config(data={})
dpigen2.op.collect_run_caly.config_args()
dpigen2.op.collect_run_caly.get_value_from_inputdat(filename)
dpigen2.op.collect_run_caly.prep_last_calypso_file(step, results, opt_results_dir, qhull_input, vsc)
```

dpigen2.op.md_settings module

```
class dpigen2.op.md_settings.MDSettings(ens: str, dt: float, nsteps: int, trj_freq: int, temps: List[float] |
    None = None, press: List[float] | None = None, tau_t: float = 0.1,
    tau_p: float = 0.5, pka_e: float | None = None, neidelay: int |
    None = None, no_pbc: bool = False, use_clusters: bool = False,
    relative_epsilon: float | None = None, relative_v_epsilon: float |
    None = None, ele_temp_f: float | None = None, ele_temp_a: float
    | None = None)
```

Bases: object

Methods

to_str

`to_str()` → str

dpigen2.op.prep_caly_dp_optim module

```
class dpigen2.op.prep_caly_dp_optim.PrepCalyDPOptim(*args, **kwargs)
```

Bases: OP

Prepare the working directories and input file according to slices information for structure optimization with DP.

`POSCAR_*`, `frozen_model.pb` or `model.ckpt.pt`, `calypso_run_opt.py` and `calypso_check_opt.py` will be copied or symlink to each optimization directory from `ip["work_path"]`, according to the group size of `ip["template_slice_config"]`. The `POSCAR_*` will be splitted into `group_size` parts and the name of each path will be returned in a `task_names` list and `task_dirs` list.

Attributes

`key`
`workflow_name`

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

<code>convert_to_graph</code>
<code>exec_sign_check</code>
<code>from_graph</code>
<code>function</code>
<code>get_info</code>
<code>get_input_artifact_link</code>
<code>get_input_artifact_storage_key</code>
<code>get_opio_info</code>
<code>get_output_artifact_link</code>
<code>get_output_artifact_storage_key</code>
<code>register_output_artifact</code>
<code>superfunction</code>

`execute(ip: OPIO) → OPIO`

Execute the OP.

Parameters

`ip`

[dict] Input dict with components: - `task_name` : (str) - `finished` : (str) - `template_slice_config` : (dict) - `poscar_dir` : (Path) - `models_dir` : (Path) - `caly_run_opt_file` : (Path) - `caly_check_opt_file` : (Path)

Returns

`op`

[dict] Output dict with components:

- `task_names`: (List[str])
- `task_dirs`: (Artifact(List[Path]))
- `caly_run_opt_file` : (Path)
- `caly_check_opt_file` : (Path)

`classmethod get_input_sign()`

Get the signature of the inputs

`classmethod get_output_sign()`

Get the signature of the outputs

dpgen2.op.prep_caly_input module

```
class dpgen2.op.prep_caly_input.PrepCalyInput(*args, **kwargs)
```

Bases: `OP`

Prepare the working directories and input file for generating structures.

A calypso input file will be generated according to the given parameters (defined by `ip[“caly_inputs”]`). The artifact will be return (`ip[input_files]`). The name of directory is `ip[“task_names”]`.

Attributes

`key`
`workflow_name`

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

`convert_to_graph`
`exec_sign_check`
`from_graph`
`function`
`get_info`
`get_input_artifact_link`
`get_input_artifact_storage_key`
`get_opio_info`
`get_output_artifact_link`
`get_output_artifact_storage_key`
`register_output_artifact`
`superfunction`

execute(`ip: OPIO`) → OPIO

Execute the OP.

Parameters

ip
[dict] Input dict with components: - `caly_task_grp` : (`BigParameter()`) Definitions for CALYPSO input file.

Returns

op
[dict] Output dict with components:

- `task_names`: (`List[str]`) The name of CALYPSO tasks. Will be used as the identities of the tasks. The names of different tasks are different.
- `input_dat_files`: (`Artifact(List[Path])`) The prepared working paths of the task containing input files (`input.dat` and `calypso_run_opt.py`) needed to generate structures by CALYPSO and make structure optimization with DP model.

- *caly_run_opt_files*: (*Artifact(List[Path])*)
- *caly_check_opt_files*: (*Artifact(List[Path])*)

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

dpgen2.op.prep_dp_train module**class dpgen2.op.prep_dp_train.PrepDPTrain(*args, **kwargs)**

Bases: [OP](#)

Prepares the working directories for DP training tasks.

A list of (*numb_models*) working directories containing all files needed to start training tasks will be created. The paths of the directories will be returned as *op[“task_paths”]*. The identities of the tasks are returned as *op[“task_names”]*.

Attributes

key
workflow_name

Methods

execute(ip)	Execute the OP.
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs

[convert_to_graph](#)
[exec_sign_check](#)
[from_graph](#)
[function](#)
[get_info](#)
[get_input_artifact_link](#)
[get_input_artifact_storage_key](#)
[get_opio_info](#)
[get_output_artifact_link](#)
[get_output_artifact_storage_key](#)
[register_output_artifact](#)
[superfunction](#)

execute(ip: OPIO) → OPIO

Execute the OP.

Parameters

ip
[dict] Input dict with components:

- *template_script*: (*str* or *List[str]*) A template of the training script. Can be a *str* or *List[str]*. In the case of *str*, all training tasks share the same training input template, the only difference is the random number used to initialize the network parameters. In the case of *List[str]*, one training task uses one template from the list. The random numbers used to initialize the network parameters are different. The length of the list should be the same as *numb_models*.
- *numb_models*: (*int*) Number of DP models to train.

Returns**op**

[dict] Output dict with components:

- *task_names*: (*List[str]*) The name of tasks. Will be used as the identities of the tasks. The names of different tasks are different.
- *task_paths*: (*Artifact(List[Path])*) The prepared working paths of the tasks. The order of the Paths should be consistent with *op[“task_names”]*

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

dpgen2.op.prep_lmp module**dpgen2.op.prep_lmp.PrepExplorationTaskGroup**

alias of *PrepLmp*

class dpgen2.op.prep_lmp.PrepLmp(*args, **kwargs)

Bases: *OP*

Prepare the working directories for LAMMPS tasks.

A list of working directories (defined by *ip[“task”]*) containing all files needed to start LAMMPS tasks will be created. The paths of the directories will be returned as *op[“task_paths”]*. The identities of the tasks are returned as *op[“task_names”]*.

Attributes

key
workflow_name

Methods

execute(ip)	Execute the OP.
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

execute(ip: OPIO) → OPIO

Execute the OP.

Parameters**ip**

[dict] Input dict with components: - *lmp_task_grp* : (*BigParameter(Path)*) Can be pickle loaded as a ExplorationTaskGroup. Definitions for LAMMPS tasks

Returns**op**

[dict] Output dict with components:

- *task_names*: (*List[str]*) The name of tasks. Will be used as the identities of the tasks. The names of different tasks are different.
- *task_paths*: (*Artifact(List[Path])*) The prepared working paths of the tasks. Contains all input files needed to start the LAMMPS simulation. The order fo the Paths should be consistent with *op[“task_names”]*

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

dpgen2.op.run_caly_dp_optim module**class dpgen2.op.run_caly_dp_optim.RunCalyDPOptim(*args, **kwargs)**

Bases: **OP**

Perform structure optimization with DP in *ip[“work_path”]*.

The *optim_results_dir* and *traj_results* will be returned as *op[“optim_results_dir”]* and *op[“traj_results”]*.

Attributes**key****workflow_name**

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

<code>convert_to_graph</code>
<code>exec_sign_check</code>
<code>from_graph</code>
<code>function</code>
<code>get_info</code>
<code>get_input_artifact_link</code>
<code>get_input_artifact_storage_key</code>
<code>get_opio_info</code>
<code>get_output_artifact_link</code>
<code>get_output_artifact_storage_key</code>
<code>register_output_artifact</code>
<code>superfunction</code>

execute(*ip*: *OPIO*) → *OPIO*

Execute the OP.

Parameters

ip

[dict] Input dict with components: - *config*: (dict) The config of calypso task to obtain the command of calypso. - *task_name* : (str) - *finished* : (str) - *cnt_num* : (int) - *task_dir* : (Path)

Returns

op

[dict] Output dict with components:

- *task_name*: (str)
- *optim_results_dir*: (List[str])
- *traj_results*: (Artifact(List[Path]))

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

dpgen2.op.run_caly_model_devi module

```
class dpgen2.op.run_caly_model_devi.RunCalyModelDevi(*args, **kwargs)
```

Bases: [OP](#)

calculate model deviation of trajectories structures.

Structure optimization will be executed in *optim_path*. The trajectory will be stored in files *op[“traj”]* and *op[“model_devi”]*, respectively.

Attributes

key	Execute the OP.
workflow_name	Get the signature of the inputs
	Get the signature of the outputs

Methods

execute(ip)	Execute the OP.
get_input_sign()	Get the signature of the inputs
get_output_sign()	Get the signature of the outputs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

execute(ip: [OPIO](#)) → OPIO

Execute the OP.

Parameters

ip

[dict] Input dict with components: - *type_map*: (*List[str]*) The type map of elements. - *task_name*: (*str*) The name of the task. - *traj_dirs*: (*Artifact(List[Path])*) The List of paths that contains trajectory files. - *models*: (*Artifact(List[Path])*) The frozen model to estimate the model deviation.

Returns

Any

Output dict with components: - *task_name*: (*str*) The name of task. - *traj*: (*Artifact(List[Path])*) The output trajectory. - *model_devi*: (*Artifact(List[Path])*) The model deviation. The order of recorded model deviations should be consistent with the order of frames in *traj*.

```
classmethod get_input_sign()
    Get the signature of the inputs
classmethod get_output_sign()
    Get the signature of the outputs
dpgen2.op.run_caly_model_devi.atoms2lmpdump(atoms, struc_idx, type_map, ignore=False)
    down triangle cell can be obtained from cell params: a, b, c, alpha, beta, gamma. cell = cellpar_to_cell([a, b, c, alpha, beta, gamma]) lx, ly, lz = cell[0][0], cell[1][1], cell[2][2] xy, xz, yz = cell[1][0], cell[2][0], cell[2][1] (lx,ly,lz) = (xhi-xlo,yhi-ylo,zhi-zlo) xlo_bound = xlo + MIN(0.0,xy,xz,xy+xz) xhi_bound = xhi + MAX(0.0,xy,xz,xy+xz) ylo_bound = ylo + MIN(0.0,yz) yhi_bound = yhi + MAX(0.0,yz) zlo_bound = zlo zhi_bound = zhi
    ref: https://docs.lammps.org/Howto\_triclinic.html
dpgen2.op.run_caly_model_devi.parse_traj(traj_file)
dpgen2.op.run_caly_model_devi.write_model_devi_out(devi: ndarray, fname: str | Path, header: str = '')
```

dpgen2.op.run_dp_train module

```
class dpgen2.op.run_dp_train.RunDPTTrain(*args, **kwargs)
```

Bases: OP

Execute a DP training task. Train and freeze a DP model.

A working directory named *task_name* is created. All input files are copied or symbol linked to directory *task_name*. The DeePMD-kit training and freezing commands are executed from directory *task_name*.

Attributes

```
key
workflow_name
```

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

convert_to_graph
decide_init_model
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
normalize_config
register_output_artifact
skip_training
superfunction
training_args
write_data_to_input_script
write_other_to_input_script

```
static decide_init_model(config, init_model, init_data, iter_data, mixed_type=False)

default_optional_parameter = {'finetune_mode': 'no', 'mixed_type': False}

execute(ip: OPIO) → OPIO
```

Execute the OP.

Parameters

ip

[dict] Input dict with components:

- *config*: (dict) The config of training task. Check *RunDPTrain.training_args* for definitions.
- *task_name*: (str) The name of training task.
- *task_path*: (Artifact(Path)) The path that contains all input files prepared by *PrepDPTrain*.
- *init_model*: (Artifact(Path)) A frozen model to initialize the training.
- *init_data*: (Artifact(List[Path])) Initial training data.
- *iter_data*: (Artifact(List[Path])) Training data generated in the DPGEN iterations.

Returns

Any

Output dict with components: - *script*: (Artifact(Path)) The training script. - *model*: (Artifact(Path)) The trained frozen model. - *lcurve*: (Artifact(Path)) The learning curve file. - *log*: (Artifact(Path)) The log file of training.

Raises

FatalError

On the failure of training or freezing. Human intervention needed.

```
classmethod get_input_sign()
    Get the signature of the inputs
classmethod get_output_sign()
    Get the signature of the outputs
static normalize_config(data={})
static skip_training(work_dir, train_dict, init_model, iter_data, finetune_mode)
static training_args()
static write_data_to_input_script(idict: dict, config, init_data: List[Path], iter_data: List[Path],
                                  auto_prob_str: str = 'prob_sys_size', major_version: str = '1',
                                  valid_data: List[Path] | None = None)
static write_other_to_input_script(idict, config, do_init_model, major_version: str = '1')
dpgen2.op.run_dp_train.config_args()
```

dpgen2.op.run_lmp module

```
class dpgeen2.op.run_lmp.RunLmp(*args, **kwargs)
```

Bases: OP

Execute a LAMMPS task.

A working directory named *task_name* is created. All input files are copied or symbol linked to directory *task_name*. The LAMMPS command is executed from directory *task_name*. The trajectory and the model deviation will be stored in files *op[“traj”]* and *op[“model_devi”]*, respectively.

Attributes

```
key
workflow_name
```

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

<code>convert_to_graph</code>
<code>exec_sign_check</code>
<code>from_graph</code>
<code>function</code>
<code>get_info</code>
<code>get_input_artifact_link</code>
<code>get_input_artifact_storage_key</code>
<code>get_opio_info</code>
<code>get_output_artifact_link</code>
<code>get_output_artifact_storage_key</code>
<code>lmp_args</code>
<code>normalize_config</code>
<code>register_output_artifact</code>
<code>superfunction</code>

execute(ip: *OPIO*) → *OPIO*

Execute the OP.

Parameters**ip**

[dict] Input dict with components:

- *config*: (*dict*) The config of lmp task. Check *RunLmp.lmp_args* for definitions.
- *task_name*: (*str*) The name of the task.
- *task_path*: (*Artifact(Path)*) The path that contains all input files prepared by *PrepLmp*.
- *models*: (*Artifact(List[Path])*) The frozen model to estimate the model deviation. The first model will be used to drive molecular dynamics simulation.

Returns**Any**

Output dict with components: - *log*: (*Artifact(Path)*) The log file of LAMMPS. - *traj*: (*Artifact(Path)*) The output trajectory. - *model_devi*: (*Artifact(Path)*) The model deviation. The order of recorded model deviations should be consistent with the order of frames in *traj*.

Raises**TransientError**

On the failure of LAMMPS execution. Handle different failure cases? e.g. loss atoms.

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

static lmp_args()**static normalize_config(data={})**

```
dpgen2.op.run_lmp.config_args()  
dpgen2.op.run_lmp.find_only_one_key(lmp_lines, key, raise_not_found=True)  
dpgen2.op.run_lmp.set_models(lmp_input_name: str, model_names: List[str])
```

dpgen2.op.select_confs module

```
class dpge2.op.select_confs.SelectConfs(*args, **kwargs)
```

Bases: OP

Select configurations from exploration trajectories for labeling.

Attributes

```
key  
workflow_name
```

Methods

<code>execute(ip)</code>	Execute the OP.
<code>get_input_sign()</code>	Get the signature of the inputs
<code>get_output_sign()</code>	Get the signature of the outputs

```
convert_to_graph  
exec_sign_check  
from_graph  
function  
get_info  
get_input_artifact_link  
get_input_artifact_storage_key  
get_opio_info  
get_output_artifact_link  
get_output_artifact_storage_key  
register_output_artifact  
superfunction  
validate_trajs
```

`execute(ip: OPIO) → OPIO`

Execute the OP.

Parameters

`ip`

[dict] Input dict with components:

- `conf_selector`: (`ConfSelector`) Configuration selector.
- `type_map`: (`List[str]`) The type map.
- `trajs`: (`Artifact(List[Path])`) The trajectories generated in the exploration.
- `model_devis`: (`Artifact(List[Path])`) The file storing the model deviation of the trajectory. The order of model deviation storage is consistent with that of

the trajectories. The order of frames of one model deviation storage is also consistent with that of the corresponding trajectory.

Returns

Any

Output dict with components: - *report*: (*ExplorationReport*) The report on the exploration. - *conf*: (*Artifact(List[Path])*) The selected configurations.

```
classmethod get_input_sign()
    Get the signature of the inputs
classmethod get_output_sign()
    Get the signature of the outputs
static validate_trajs(trajs, model_devis)
```

dpgen2.superop package

Submodules

dpgen2.superop.block module

```
class dpgen2.superop.block.ConcurrentLearningBlock(name: str, prep_run_dp_train_op: PrepRunDPTTrain, prep_run_explore_op: PrepRunLmp | PrepRunCaly, select_confs_op: Type[OP], prep_run_fp_op: PrepRunFp, collect_data_op: Type[OP], select_confs_config: dict = {'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'executor': None, 'parallelism': None, 'template_config': {'envs': None, 'image': 'dptechology/dpgen2:latest', 'retry_on_transient_error': None, 'timeout': None, 'timeout_as_transient_error': False}}, collect_data_config: dict = {'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'executor': None, 'parallelism': None, 'template_config': {'envs': None, 'image': 'dptechology/dpgen2:latest', 'retry_on_transient_error': None, 'timeout': None, 'timeout_as_transient_error': False}}, upload_python_packages: List[PathLike] | None = None)
```

Bases: Steps

Attributes

- input_artifacts
- input_parameters
- keys
- output_artifacts
- output_parameters

Methods

add(step)	Add a step or a list of parallel steps to the steps
-----------	---

add_slices
convert_to_argo
convert_to_graph
copy
deepcopy
from_dict
from_graph
handle_key
run

```
property input_artifacts  
property input_parameters  
property keys  
property output_artifacts  
property output_parameters
```

```
dpgen2.superop.block.make_collect_data_optional_parameter(block_optional_parameter)
```

```
dpgen2.superop.block.make_run_dp_train_optional_parameter(block_optional_parameter)
```

dpgen2.superop.caly_evo_step module

```
class dpgen2.superop.caly_evo_step.CalyEvoStep(name: str, collect_run_caly: Type[OP],  
                                                prep_dp_optim: Type[OP], run_dp_optim: Type[OP],  
                                                prep_config: dict = {'continue_on_failed': False,  
                                                    'continue_on_num_success': None,  
                                                    'continue_on_success_ratio': None, 'executor': None,  
                                                    'parallelism': None, 'template_config': {'envs': None,  
                                                      'image': 'dptechnology/dpgen2:latest',  
                                                      'retry_on_transient_error': None, 'timeout': None,  
                                                      'timeout_as_transient_error': False}, 'run_config': dict  
                                                    = {'continue_on_failed': False,  
                                                        'continue_on_num_success': None,  
                                                        'continue_on_success_ratio': None, 'executor': None,  
                                                        'parallelism': None, 'template_config': {'envs': None,  
                                                          'image': 'dptechnology/dpgen2:latest',  
                                                          'retry_on_transient_error': None, 'timeout': None,  
                                                          'timeout_as_transient_error': False},  
                                                        'upload_python_packages': List[PathLike] | None =  
                                                       None})
```

Bases: [Steps](#)

Attributes

input_artifacts
input_parameters
keys
output_artifacts
output_parameters

Methods

add(step)	Add a step or a list of parallel steps to the steps
-----------	---

add_slices
convert_to_argo
convert_to_graph
copy
deepcopy
from_dict
from_graph
handle_key
run

property input_artifacts
property input_parameters
property keys
property output_artifacts
property output_parameters

dpgen2.superop.prep_run_calypso module

```
class dpgen2.superop.prep_run_calypso.PrepRunCaly(name: str, prep_caly_input_op: Type[OP],
                                                 caly_evo_step_op: OPTemplate,
                                                 run_caly_model_devi_op: Type[OP], prep_config:
                                                 dict = {'continue_on_failed': False,
                                                 'continue_on_num_success': None,
                                                 'continue_on_success_ratio': None, 'executor':
                                                 None, 'parallelism': None, 'template_config':
                                                 {'envs': None, 'image':
                                                 'dptechology/dpge2:latest',
                                                 'retry_on_transient_error': None, 'timeout': None,
                                                 'timeout_as_transient_error': False}}, run_config:
                                                 dict = {'continue_on_failed': False,
                                                 'continue_on_num_success': None,
                                                 'continue_on_success_ratio': None, 'executor':
                                                 None, 'parallelism': None, 'template_config':
                                                 {'envs': None, 'image':
                                                 'dptechology/dpge2:latest',
                                                 'retry_on_transient_error': None, 'timeout': None,
                                                 'timeout_as_transient_error': False}},
                                                 upload_python_packages: List[PathLike] | None =
                                                 None)
```

Bases: `Steps`**Attributes**

- `input_artifacts`
- `input_parameters`
- `keys`
- `output_artifacts`
- `output_parameters`

Methods

<code>add(step)</code>	Add a step or a list of parallel steps to the steps
------------------------	---

<code>add_slices</code>
<code>convert_to_argo</code>
<code>convert_to_graph</code>
<code>copy</code>
<code>deepcopy</code>
<code>from_dict</code>
<code>from_graph</code>
<code>handle_key</code>
<code>run</code>

`property input_artifacts`
`property input_parameters`

```
property keys
property output_artifacts
property output_parameters
```

dpgen2.superop.prep_run_dp_train module

class dpgen2.superop.prep_run_dp_train.**ModifyTrainScript**(*args, **kwargs)

Bases: OP

Modify the training scripts to prepare them for training tasks in dpgen step.

Read the training scripts modified by finetune, and replace the original template scripts to be compatible with pre-trained models. New templates are returned as *op[“template_script”]*.

Attributes

key	
workflow_name	

Methods

<i>execute(ip)</i>	Execute the OP.
<i>get_input_sign()</i>	Get the signature of the inputs
<i>get_output_sign()</i>	Get the signature of the outputs

convert_to_graph
exec_sign_check
from_graph
function
get_info
get_input_artifact_link
get_input_artifact_storage_key
get_opio_info
get_output_artifact_link
get_output_artifact_storage_key
register_output_artifact
superfunction

execute(ip: OPIO) → OPIO

Execute the OP.

Parameters

ip
[dict] Input dict with components:

- *scripts*: (*Artifact(Path)*) Training scripts from finetune.
- *numb_models*: (*int*) Number of DP models to train.

Returns

op

[dict] Output dict with components:

- *template_script*: (*List[dict]*) One template from one finetuning task. The length of the list should be the same as *num_models*.

classmethod get_input_sign()

Get the signature of the inputs

classmethod get_output_sign()

Get the signature of the outputs

```
class dpgen2.superop.prep_run_dp_train.PrepRunDPTrain(name: str, prep_train_op: ~typing.Type[~dflow.python.op.OP], run_train_op: ~typing.Type[~dpigen2.op.run_dp_train.RunDPTrain], modify_train_script_op: ~typing.Type[~dpigen2.superop.prep_run_dp_train.ModifyTrainScript] = <class 'dpgen2.superop.prep_run_dp_train.ModifyTrainScript'>, prep_config: dict = {'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'executor': None, 'parallelism': None, 'template_config': {'envs': None, 'image': 'dptechnology/dpigen2:latest', 'retry_on_transient_error': None, 'timeout': None, 'timeout_as_transient_error': False}}, run_config: dict = {'continue_on_failed': False, 'continue_on_num_success': None, 'continue_on_success_ratio': None, 'executor': None, 'parallelism': None, 'template_config': {'envs': None, 'image': 'dptechnology/dpigen2:latest', 'retry_on_transient_error': None, 'timeout': None, 'timeout_as_transient_error': False}}, upload_python_packages: ~typing.List[~os.PathLike] | None = None, finetune: bool = False, valid_data: ~dflow.common.S3Artifact | None = None)
```

Bases: [Steps](#)

Attributes

input_artifacts
input_parameters
keys
output_artifacts
output_parameters

Methods

`add(step)` Add a step or a list of parallel steps to the steps

```
add_slices  
convert_to_argo  
convert_to_graph  
copy  
deepcopy  
from_dict  
from_graph  
handle_key  
run
```

```
property input_artifacts  
property input_parameters  
property keys  
property output_artifacts  
property output_parameters
```

dpgen2.superop.prep_run_fp module

```
class dpctl.superop.prep_run_fp.PrepRunFp(name: str, prep_op: Type[OP], run_op: Type[OP],  
    prep_config: dict = {'continue_on_failed': False,  
    'continue_on_num_success': None,  
    'continue_on_success_ratio': None, 'executor': None,  
    'parallelism': None, 'template_config': {'envs': None,  
    'image': 'dpctl/dpctl:latest',  
    'retry_on_transient_error': None, 'timeout': None,  
    'timeout_as_transient_error': False}}, run_config: dict =  
    {'continue_on_failed': False, 'continue_on_num_success':  
    None, 'continue_on_success_ratio': None, 'executor': None,  
    'parallelism': None, 'template_config': {'envs': None,  
    'image': 'dpctl/dpctl:latest',  
    'retry_on_transient_error': None, 'timeout': None,  
    'timeout_as_transient_error': False}},  
    upload_python_packages: List[PathLike] | None = None)
```

Bases: Steps

Attributes

input_artifacts
input_parameters
keys
output_artifacts
output_parameters

Methods

add(step)	Add a step or a list of parallel steps to the steps
-----------	---

add_slices
convert_to_argo
convert_to_graph
copy
deepcopy
from_dict
from_graph
handle_key
run

```
property input_artifacts  
property input_parameters  
property keys  
property output_artifacts  
property output_parameters
```

dpgen2.superop.prep_run_lmp module

```
class dpgen2.superop.prep_run_lmp.PrepRunLmp(name: str, prep_op: Type[OP], run_op: Type[OP],  
                                              prep_config: dict = {'continue_on_failed': False,  
                                              'continue_on_num_success': None,  
                                              'continue_on_success_ratio': None, 'executor': None,  
                                              'parallelism': None, 'template_config': {'envs': None,  
                                              'image': 'dptechology/dpgen2:latest',  
                                              'retry_on_transient_error': None, 'timeout': None,  
                                              'timeout_as_transient_error': False}}, run_config: dict =  
                                              {'continue_on_failed': False, 'continue_on_num_success':  
                                              None, 'continue_on_success_ratio': None, 'executor':  
                                              None, 'parallelism': None, 'template_config': {'envs':  
                                              None, 'image': 'dptechology/dpgen2:latest',  
                                              'retry_on_transient_error': None, 'timeout': None,  
                                              'timeout_as_transient_error': False}},  
                                              upload_python_packages: List[PathLike] | None = None)
```

Bases: Steps

Attributes

```
input_artifacts  
input_parameters  
keys  
output_artifacts  
output_parameters
```

Methods

add(step)	Add a step or a list of parallel steps to the steps
-----------	---

add_slices
convert_to_argo
convert_to_graph
copy
deepcopy
from_dict
from_graph
handle_key
run

```
property input_artifacts
property input_parameters
property keys
property output_artifacts
property output_parameters
```

dpgen2.utils package

Submodules

dpgen2.utils.binary_file_input module

Binary file inputs

```
class dpgen2.utils.binary_file_input.BinaryFileInput(path: str | Path, ext: str | None = None)
Bases: object
```

Methods

save_as_file

```
save_as_file(path: str | Path) → None
```

dpgen2.utils.bohrium_config module

`dpgen2.utils.bohrium_config.bohrium_config_from_dict(bohrium_config)`

dpgen2.utils.chdir module

`dpgen2.utils.chdir.chdir(path_key: str)`

Returns a decorator that can change the current working path.

Parameters

path_key
[str] key to OPIO

Examples

```
>>> class SomeOP(OP):
...     @chdir("path")
...     def execute(self, ip: OPIO):
...         do_something()
```

`dpgen2.utils.chdir.set_directory(path: Path)`

Sets the current working path within the context.

Parameters

path
[Path] The path to the cwd

Yields

None

Examples

```
>>> with set_directory("some_path"):
...     do_something()
```

dpgen2.utils.dflow_config module

`dpgen2.utils.dflow_config.dflow_config(config_data)`

set the dflow config by *config_data*

the keys starting with “**s3_**” will be treated as s3_config keys, other keys are treated as config keys.

`dpgen2.utils.dflow_config.dflow_config_lower(dflow_config)`

`dpgen2.utils.dflow_config.dflow_s3_config(config_data)`

set the s3 config by *config_data*

`dpgen2.utils.dflow_config.dflow_s3_config_lower(dflow_s3_config_data)`

`dpgen2.utils.dflow_config.workflow_config_from_dict(wf_config)`

dpgen2.utils.dflow_query module

```
dpgen2.utils.dflow_query.find_slice_ranges(keys: List[str], sliced_subkey: str)
    find range of sliced OPs that matches the pattern ‘iter-[0-9]*-{sliced_subkey}-[0-9]*’,
dpgen2.utils.dflow_query.get_allSchedulers(wf: Any, keys: List[str])
    get the output Scheduler of the all the iterations
dpgen2.utils.dflow_query.get_iteration(key: str)
dpgen2.utils.dflow_query.get_last_iteration(keys: List[str])
    get the index of the last iteration from a list of step keys.
dpgen2.utils.dflow_query.get_last_scheduler(wf: Any, keys: List[str])
    get the output Scheduler of the last successful iteration
dpgen2.utils.dflow_query.get_subkey(key: str, idx: int = -1)
dpgen2.utils.dflow_query.matched_step_key(all_keys: List[str], step_keys: List[str] | None = None)
    returns the keys in all_keys that matches any of the step_keys
dpgen2.utils.dflow_query.print_keys_in_nice_format(keys: List[str], sliced_subkey: List[str],
                                                idx_fmt_len: int = 8)
dpgen2.utils.dflow_query.sort_slice_ops(keys: List[str], sliced_subkey: List[str])
    sort the keys of the sliced ops. the keys of the sliced ops contains sliced_subkey
```

dpgen2.utils.download_dpgen2_artifacts module

```
class dpgen2.utils.download_dpgen2_artifacts.DownloadDefinition
    Bases: object
```

Methods

add_def
add_input
add_output

```
add_def(tdict, key, suffix=None)
add_input(input_key, suffix=None)
add_output(output_key, suffix=None)
```

```
dpgen2.utils.download_dpgen2_artifacts.download_dpgen2_artifacts(wf: Workflow, key: str, prefix:
                                                                str | None = None, chk_pnt:
                                                                bool = False)
```

download the artifacts of a step. the key should be of format ‘iter-xxxxxx-subkey-of-step-xxxxxx’ the input and output artifacts will be downloaded to prefix/iter-xxxxxx/key-of-step/inputs/ and prefix/iter-xxxxxx/key-of-step/outputs/

the downloaded input and output artifacts of steps are defined by *op_download_setting*

```
dpigen2.utils.download_dpigen2_artifacts.download_dpigen2_artifacts_by_def(wf: Workflow,  
                           iterations: List[int] |  
                           None = None,  
                           step_defs: List[str] |  
                           None = None, prefix:  
                           str | None = None,  
                           chk_pnt: bool =  
                           False)  
  
dpigen2.utils.download_dpigen2_artifacts.print_op_download_setting(op_download_setting='collect-  
data':  
                     <dp-  
                     gen2.utils.download_dpigen2_artifacts.DownloadL  
                     object>, 'prep-run-explore':  
                     <dp-  
                     gen2.utils.download_dpigen2_artifacts.DownloadL  
                     object>, 'prep-run-fp': <dp-  
                     gen2.utils.download_dpigen2_artifacts.DownloadL  
                     object>, 'prep-run-train': <dp-  
                     gen2.utils.download_dpigen2_artifacts.DownloadL  
                     object>))
```

dpigen2.utils.obj_artifact module

```
dpigen2.utils.obj_artifact.dump_object_to_file(obj, fname)
```

pickle dump object to a file

```
dpigen2.utils.obj_artifact.load_object_from_file(fname)
```

pickle load object from a file

dpigen2.utils.run_command module

```
dpigen2.utils.run_command.run_command(cmd: str | List[str], shell: bool = False) → Tuple[int, str, str]
```

dpigen2.utils.step_config module

```
dpigen2.utils.step_config.dispatcher_args()
```

free style dispatcher args

```
dpigen2.utils.step_config.gen_doc(*, make_anchor=True, make_link=True, **kwargs)
```

```
dpigen2.utils.step_config.init_executor(executor_dict)
```

```
dpigen2.utils.step_config.normalize(data)
```

```
dpigen2.utils.step_config.step_conf_args()
```

```
dpigen2.utils.step_config.template_conf_args()
```

```
dpigen2.utils.step_config.template_slice_conf_args()
```

```
dpigen2.utils.step_config.variant_executor()
```

8.1.2 Submodules

8.1.3 dpgen2.constants module

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

dpigen2, 91
dpigen2.conf, 91
dpigen2.conf.alloy_conf, 91
dpigen2.conf.conf_generator, 94
dpigen2.conf.file_conf, 95
dpigen2.conf.unit_cells, 95
dpigen2.constants, 177
dpigen2.entrypoint, 97
dpigen2.entrypoint.args, 97
dpigen2.entrypoint.common, 98
dpigen2.entrypoint.download, 98
dpigen2.entrypoint.gui, 98
dpigen2.entrypoint.main, 99
dpigen2.entrypoint.showkey, 99
dpigen2.entrypoint.status, 99
dpigen2.entrypoint.submit, 99
dpigen2.entrypoint.watch, 102
dpigen2.entrypoint.workflow, 102
dpigen2.exploration, 103
dpigen2.exploration.deviation, 103
dpigen2.exploration.deviation.deviation_manager, 103
dpigen2.exploration.deviation.deviation_std, 104
dpigen2.exploration.render, 104
dpigen2.exploration.render.traj_render, 104
dpigen2.exploration.render.traj_render_lammps, 105
dpigen2.exploration.report, 106
dpigen2.exploration.report.report, 106
dpigen2.exploration.report.report_adaptive_lower, 107
dpigen2.exploration.report.report_trust_levels_base, 110
dpigen2.exploration.report.report_trust_levels_max, 112
dpigen2.exploration.report.report_trust_levels_random, 113
dpigen2.exploration.scheduler, 114
dpigen2.exploration.scheduler.convergence_check_stage_scheduler, 114

dpigen2.exploration.scheduler.scheduler, 116
dpigen2.exploration.scheduler.stage_scheduler, 118
dpigen2.exploration.selector, 119
dpigen2.exploration.selector.conf_filter, 119
dpigen2.exploration.selector.conf_selector, 120
dpigen2.exploration.selector.conf_selector_frame, 120
dpigen2.exploration.task, 121
dpigen2.exploration.task.caly_task_group, 123
dpigen2.exploration.task.calypso, 121
dpigen2.exploration.task.calypso.caly_input, 121
dpigen2.exploration.task.conf_sampling_task_group, 123
dpigen2.exploration.task.customized_lmp_template_task_group, 124
dpigen2.exploration.task.lmp, 122
dpigen2.exploration.task.lmp.lmp_input, 122
dpigen2.exploration.task.lmp_template_task_group, 126
dpigen2.exploration.task.make_task_group_from_config, 127
dpigen2.exploration.task.npt_task_group, 127
dpigen2.exploration.task.stage, 128
dpigen2.exploration.task.task, 129
dpigen2.exploration.task.task_group, 129
dpigen2.flow, 131
dpigen2.flow.dpigen_loop, 131
dpigen2.fp, 135
dpigen2.fp.abacus, 135
dpigen2.fp.deepmd, 137
dpigen2.fp.gaussian, 140
dpigen2.fp.prep_fp, 142
dpigen2.fp.run_fp, 144
dpigen2.fp.vasp, 146
dpigen2.fp.vasp_input, 148
dpigen2.op, 149
dpigen2.op.collect_data, 149
dpigen2.op.collect_run_caly, 150
dpigen2.op.md_settings, 152

dpigen2.op.prep_caly_dp_optim, 152
dpigen2.op.prep_caly_input, 154
dpigen2.op.prep_dp_train, 155
dpigen2.op.prep_lmp, 156
dpigen2.op.run_caly_dp_optim, 157
dpigen2.op.run_caly_model_devi, 159
dpigen2.op.run_dp_train, 160
dpigen2.op.run_lmp, 162
dpigen2.op.select_confs, 164
dpigen2.superop, 165
dpigen2.superop.block, 165
dpigen2.superop.caly_evo_step, 166
dpigen2.superop.prep_run_calypso, 168
dpigen2.superop.prep_run_dp_train, 169
dpigen2.superop.prep_run_fp, 171
dpigen2.superop.prep_run_lmp, 172
dpigen2.utils, 173
dpigen2.utils.binary_file_input, 173
dpigen2.utils.bohrium_config, 174
dpigen2.utils.chdir, 174
dpigen2.utils.dflow_config, 174
dpigen2.utils.dflow_query, 175
dpigen2.utils.download_dpigen2_artifacts, 175
dpigen2.utils.obj_artifact, 176
dpigen2.utils.run_command, 176
dpigen2.utils.step_config, 176

INDEX

A

```

accurate_ratio()           (dp- args() (dpgen2.exploration.report.report_trust_levels_base.ExplorationRe-
                           gen2.exploration.report.report_adaptive_lower.ExplorationReportAdaptiveLower
                           method), 109                         static method), 111
accurate_ratio()           (dp- args() (dpgen2.fp.abacus.FpOpAbacusInputs
                           gen2.exploration.report.report_trust_levels_base.ExplorationReportTrustLevels
                           method), 111                         static method), 135
add() (dpgen2.exploration.deviation_manager.DeviManager) (method), 137
add() (dpgen2.exploration.selector.conf_filter.ConfFilters
      method), 103                         args() (dpgen2.fp.deepmd.DeepmdInputs
                                         static method),
add() (dpgen2.exploration.selector.conf_filter.ConfFilters
      method), 120                         args() (dpgen2.fp.gaussian.GaussianInputs
                                         static method)
add_def() (dpgen2.utils.download_dpgen2_artifacts.DownloadDefinition
          method), 175                         args() (dpgen2.fp.gaussian.RunGaussian
                                         static method), 139
add_file() (dpgen2.exploration.task.task.ExplorationTask
            method), 129                         args() (dpgen2.fp.run_fp.RunFp static method), 140
add_group() (dpgen2.exploration.task.task_group.BaseExplorationTaskGroup
             method), 130                         args() (dpgen2.fp.vasp.RunVasp static method), 141
                                         args() (dpgen2.fp.vasp_input.VaspInputs static method),
add_input() (dpgen2.utils.download_dpgen2_artifacts.DownloadDefinition
             method), 175                         atom_pert_dist: 148
                                         explore[calypso]/configurations[alloy]/atom_pert_dist
                                         (Argument), 60
add_output() (dpgen2.utils.download_dpgen2_artifacts.DownloadDefinition
              method), 175                         explore[lmp]/configurations[alloy]/atom_pert_dist
                                         (Argument), 53
add_stage_scheduler()       (dp- atomic_number:
                           gen2.exploration.scheduler.ExplorationScheduler
                           method), 116                         task_group/atomic_number (Argument), 72
                                         atoms2lmpdump() (in module dpgen2.entrypoint.workflow), 102
                                         (in module dpgen2.exploration.task.task_group), 129
                                         AVG_DEVI_F (dpgen2.exploration.deviation.deviation_manager.DeviManager
                                         attribute), 103
                                         AVG_DEVI_V (dpgen2.exploration.deviation.deviation_manager.DeviManager
                                         attribute), 103
add_subparser_workflow_subcommand() (in module
                                      dpgen2.entrypoint.workflow), 102
add_task() (dpgen2.exploration.task.task_group.BaseExplorationTaskGroup
            method), 130
add_task_group()           (dp- B
                           gen2.exploration.task.stage.ExplorationStage
                           method), 128
AlloyConf (class in dpgen2.conf.alloy_conf), 91
AlloyConfGenerator         (class     in     dp-
                           gen2.conf.alloy_conf), 92
args() (dpgen2.conf.alloy_conf.AlloyConfGenerator
        static method), 93
args() (dpgen2.conf.conf_generator.ConfGenerator
        static method), 94
args() (dpgen2.conf.file_conf.FileConfGenerator
        static method), 95
args() (dpgen2.exploration.report.report_adaptive_lower.ExplorationReportAdaptiveLower
        static method), 109

```

B

```

BaseExplorationTaskGroup   (class     in     dp-
                           gen2.exploration.task.task_group), 129
basis_set:                 BCC (class in dpgen2.conf.unit_cells), 95
                                         BinaryFileInput (class     in     dp-
                                         ReportAdaptiveLowerFileInput), 173

```

bohrium_conf_args() (in module `dp-gen2.entrypoint.args`), 97
bohrium_config (Argument)
 bohrium_config:, 19
bohrium_config/host (Argument)
 host:, 19
bohrium_config/k8s_api_server (Argument)
 k8s_api_server:, 20
bohrium_config/password (Argument)
 password:, 19
bohrium_config/project_id (Argument)
 project_id:, 19
bohrium_config/repo_key (Argument)
 repo_key:, 20
bohrium_config/storage_client (Argument)
 storage_client:, 20
bohrium_config/username (Argument)
 username:, 19
bohrium_config:
 bohrium_config (Argument), 19
bohrium_config_from_dict() (in module `dp-gen2.utils.bohrium_config`), 174

C

caly_normalize() (in module `dp-gen2.exploration.task.make_task_group_from_config`), 127
caly_task_group_args() (in module `dp-gen2.exploration.task.make_task_group_from_config`), 127
caly_task_grp_args() (in module `dp-gen2.exploration.task.make_task_group_from_config`), 127
CalyEvoStep (class in `dpgen2.superop.caly_evo_step`), 166
calypso_args() (`dpgen2.op.collect_run_caly.CollRunCaly` static method), 151
CalyTaskGroup (class in `dp-gen2.exploration.task.caly_task_group`), 123
candi_sel_prob:
 explore[calypso]/convergence[adaptive-lower]/candi_sel_prob (Argument), 59
 explore[lmp]/convergence[adaptive-lower]/candi_sel_prob (Argument), 52
candidate_ratio() (dp-
 gen2.exploration.report_adaptive_lower.ExplorationReportAdaptiveLower method), 109
candidate_ratio() (dp-
 gen2.exploration.report_report_trust_levels_base.ExplorationReportTrustLevels method), 111
cell_pert_frac:
 explore[calypso]/configurations[alloy]/cell_pert_frac (Argument), 60

explore[lmp]/configurations[alloy]/cell_pert_frac (Argument), 53
charge:
 fp[gaussian]/inputs_config/charge (Argument), 63
chdir() (in module `dpgen2.utils.chdir`), 174
check() (`dpgen2.exploration.selector.conf_filter.ConfFilter` method), 119
check() (`dpgen2.exploration.selector.conf_filter.ConfFilters` method), 120
cl_step_config:
 step_configs/cl_step_config (Argument), 39
clear() (`dpgen2.exploration.deviation.deviation_manager.DeviManager` method), 103
clear() (`dpgen2.exploration.deviation.deviation_std.DeviManagerStd` method), 104
clear() (`dpgen2.exploration.report.report.ExplorationReport` method), 106
clear() (`dpgen2.exploration.report.report_adaptive_lower.ExplorationReport` method), 109
clear() (`dpgen2.exploration.report.report_trust_levels_base.ExplorationReport` method), 111
clear() (`dpgen2.exploration.task.stage.ExplorationStage` method), 128
clear() (`dpgen2.exploration.task.task_group.BaseExplorationTaskGroup` method), 130
collect_data_config:
 step_configs/collect_data_config (Argument), 36
 CollectData (class in `dpgen2.op.collect_data`), 149
CollRunCaly (class in `dpgen2.op.collect_run_caly`), 150
command:
 explore[calypso]/config/command (Argument), 55
 explore[lmp]/config/command (Argument), 48
 fp[fpop_abacus]/run_config/command (Argument), 66
 fp[gaussian]/run_config/command (Argument), 64
 fp[vasp]/run_config/command (Argument), 62
 task_group/command (Argument), 74
complete() (`dpgen2.exploration.scheduler.convergence_check_stage_scheduler` method), 115
complete() (`dpgen2.exploration.scheduler.scheduler.ExplorationScheduler` method), 116
complete() (`dpgen2.exploration.scheduler.scheduler.StageScheduler` method), 118
concentration:
 explore[calypso]/configurations[alloy]/concentration (Argument), 60
 explore[lmp]/configurations[alloy]/concentration (Argument), 53
concurrent_learning (class in `dp-gen2.flow.dpgen_loop`), 131

ConcurrentLearningBlock (class in *dp-gen2.superop.block*), 165
 ConcurrentLearningLoop (class in *dp-gen2.flow.dpgen_loop*), 132
 conf_idx:
 task_group[customized-lmp-template]/conf_idx (Argument), 70
 task_group[lmp-md]/conf_idx (Argument), 67
 task_group[lmp-template]/conf_idx (Argument), 69
 ConfFilter (class in *dp-gen2.exploration.selector.conf_filter*), 119
 ConfFilters (class in *dp-gen2.exploration.selector.conf_filter*), 120
 ConfGenerator (class in *dpgen2.conf.conf_generator*), 94
 config:
 explore[calypso]/config (Argument), 54
 explore[lmp]/config (Argument), 48
 train[dp-dist]/config (Argument), 45
 train[dp]/config (Argument), 43
 config_args() (in module *dp-gen2.op.collect_run_caly*), 152
 config_args() (in module *dpgen2.op.run_dp_train*), 162
 config_args() (in module *dpgen2.op.run_lmp*), 163
 config_strip_configidx() (in module *dp-gen2.exploration.task.make_task_group_from_config*), 127
 configurations:
 explore[calypso]/configurations (Argument), 59
 explore[lmp]/configurations (Argument), 52
 ConfSamplingTaskGroup (class in *dp-gen2.exploration.task.conf_sampling_task_group*), 123
 ConfSelector (class in *dp-gen2.exploration.selector.conf_selector*), 120
 ConfSelectorFrames (class in *dp-gen2.exploration.selector.conf_selector_frame*), 120
 continue_on_failed:
 default_step_config/continue_on_failed (Argument), 18
 step_configs/cl_step_config/continue_on_failed (Argument), 40
 step_configs/collect_data_config/continue_on_failed (Argument), 38
 step_configs/prep_explore_config/continue_on_failed (Argument), 26
 step_configs/prep_fp_config/continue_on_failed (Argument), 31
 step_configs/prep_train_config/continue_on_failed (Argument), 57
 (Argument), 21
 step_configs/run_explore_config/continue_on_failed (Argument), 28
 step_configs/run_fp_config/continue_on_failed (Argument), 33
 step_configs/run_train_config/continue_on_failed (Argument), 24
 step_configs/select_confs_config/continue_on_failed (Argument), 35
 continue_on_num_success:
 default_step_config/continue_on_num_success (Argument), 18
 step_configs/cl_step_config/continue_on_num_success (Argument), 40
 step_configs/collect_data_config/continue_on_num_success (Argument), 38
 step_configs/prep_explore_config/continue_on_num_success (Argument), 26
 step_configs/prep_fp_config/continue_on_num_success (Argument), 31
 step_configs/prep_train_config/continue_on_num_success (Argument), 22
 step_configs/run_explore_config/continue_on_num_success (Argument), 29
 step_configs/run_fp_config/continue_on_num_success (Argument), 33
 step_configs/run_train_config/continue_on_num_success (Argument), 24
 step_configs/select_confs_config/continue_on_num_success (Argument), 36
 continue_on_success_ratio:
 default_step_config/continue_on_success_ratio (Argument), 18
 step_configs/cl_step_config/continue_on_success_ratio (Argument), 40
 step_configs/collect_data_config/continue_on_success_ratio (Argument), 38
 step_configs/prep_explore_config/continue_on_success_ratio (Argument), 26
 step_configs/prep_fp_config/continue_on_success_ratio (Argument), 31
 step_configs/prep_train_config/continue_on_success_ratio (Argument), 22
 step_configs/run_explore_config/continue_on_success_ratio (Argument), 29
 step_configs/run_fp_config/continue_on_success_ratio (Argument), 33
 step_configs/run_train_config/continue_on_success_ratio (Argument), 24
 step_configs/select_confs_config/continue_on_success_ratio (Argument), 36
 accuracy:
 explore[calypso]/convergence[fixed-levels-max-select] (Argument), 57

```

explore[calypso]/convergence[fixed-levels]/config[fp[abacus]]/inputs_config/deepks_model
    (Argument), 56
explore[lmp]/convergence[fixed-levels-max-BetaDynamics](akon/dygen2.fp.deepmd), 137
    (Argument), 51
explore[lmp]/convergence[fixed-levels]/conv_accuracy[dp2.op.collect_data.CollectData attribute],
    (Argument), 50
    149
conv_tolerance:                                default_optional_parameter          (dp-
    explore[calypso]/convergence[adaptive-lower]/conv_accuracy_dp_train.RunDPTTrain attribute),
    (Argument), 59
    161
explore[lmp]/convergence[adaptive-lower]/conv_accuracy_dp_train.RunDPTTrain attribute,
    (Argument), 52
    17
converged() (dpgen2.exploration.report.ExplorationReport.default_step_config/continue_on_failed (Argu-
    method), 106
    ment)
converged() (dpgen2.exploration.report_adaptive_lower.default_step_config/continue_on_num_success
    method), 109
    18
converged() (dpgen2.exploration.report_trust_levels_base.ExplorationReportTrustLevels.default_step_config/continue_on_num_success
    method), 111
    18
converged() (dpgen2.exploration.report_trust_levels_base.ExplorationReportTrustLevels.default_step_config/continue_on_num_success_ratio
    method), 112
    (Argument)
converged() (dpgen2.exploration.report_trust_levels_random.default_executor_parallelism/Random
    method), 113
    default_step_config/executor (Argument)
converged() (dpgen2.exploration.scheduler.convergence_check_scheduler.ConvergenceCheckStageScheduler.default_step_config/executor/type (Argument)
    method), 115
    19
converged() (dpgen2.exploration.scheduler.stage_scheduler.StageScheduler.default_step_config/parallelism (Argument)
    method), 118
    parallelism:, 19
convergence:
    explore[calypso]/convergence (Argument), 55
    explore[lmp]/convergence (Argument), 49
        default_step_config/template_config      (Argu-
        ment)
ConvergenceCheckStageScheduler (class in dp-           template_config:, 17
    gen2.exploration.scheduler.convergence_check_stage.default_step_config/template_config/envs (Ar-
    114
        gument)
copy_scheduler_plans() (in module     dp-           envs:, 18
    dp-           gen2.entrypoint.submit), 99
        default_step_config/template_config/image
ctrl_range:
    task_group/ctrl_range (Argument), 75
        image:, 17
custom_shell_commands:
    task_group[customized-lmp-template]/custom_shell_commands
        (Argument), 70
        retry_on_transient_error:, 17
customized_lmp_template_task_group_args()           default_step_config/template_config/timeout
    (in           module           dp-           (Argument)
    gen2.exploration.task.make_task_group_from_config), timeout:, 17
    127
        default_step_config/template_config/timeout_as_transient_e
CustomizedLmpTemplateTaskGroup (class in dp-           (Argument)
    gen2.exploration.task.customized_lmp_template_task_group.default_step_config/template_slice_config
    124
        group_size:, 18
        template_slice_config:, 18
D
decide_init_model()           (dp-           default_step_config/template_slice_config/group_size
    gen2.op.run_dp_train.RunDPTTrain static
    method), 161
        group_size:, 18
deepks_descriptor:
    fp[fp[abacus]]/inputs_config/deepks_descriptor (Argument)
    (Argument), 66
        pool_size:, 18
deepks_model:                 default_step_config:

```

```

default_step_config (Argument), 17
default_step_config_args() (in module dp-
    gen2.entrypoint.args), 97
DeviManager (class in dp-
    gen2.exploration.deviation.deviation_manager),
    103
DeviManagerStd (class in dp-
    gen2.exploration.deviation.deviation_std),
    104
dflow_conf_args() (in module dp-
    gen2.entrypoint.args), 97
dflow_config (Argument)
    dflow_config:, 17
dflow_config() (in module dpgen2.utils.dflow_config),
    174
dflow_config:
    dflow_config (Argument), 17
dflow_config_lower() (in module dp-
    gen2.utils.dflow_config), 174
dflow_s3_config (Argument)
    dflow_s3_config:, 17
dflow_s3_config() (in module dp-
    gen2.utils.dflow_config), 174
dflow_s3_config:
    dflow_s3_config (Argument), 17
dflow_s3_config_lower() (in module dp-
    gen2.utils.dflow_config), 174
DIAMOND (class in dpgen2.conf.unit_cells), 96
dispatcher_args() (in module dp-
    gen2.utils.step_config), 176
distance_of_ions:
    task_group/distance_of_ions (Argument), 72
do_finetune:
    inputs/do_finetune (Argument), 42
doc() (dpgen2.conf.alloy_conf.AlloyConfGenerator
    static method), 93
doc() (dpgen2.conf.file_conf.FileConfGenerator static
    method), 95
doc() (dpgen2.exploration.report.report_adaptive_lower.Ex-
    ploreAndAdaptiveLowerWorkflow
    static method), 110
doc() (dpgen2.exploration.report.report_trust_levels_max.Ex-
    ploreAndExplorationLevelsMax
    static method), 112
doc() (dpgen2.exploration.report.report_trust_levels_random.Ex-
    ploreAndExplorationTrustLevelRandom
    static method), 113
download() (in module dpgen2.entrypoint.download),
    98
download_by_def() (in module dp-
    gen2.entrypoint.download), 98
download_dpgen2_artifacts() (in module dp-
    gen2.utils.download_dpgen2_artifacts), 175
download_dpgen2_artifacts_by_def() (in module dp-
    gen2.utils.download_dpgen2_artifacts), 175
DownloadDefinition (class in dp-
    gen2.utils.download_dpgen2_artifacts), 175
dp_dist_train_args() (in module dp-
    gen2.entrypoint.args), 97
dp_train_args() (in module dpgen2.entrypoint.args),
    97
dpgen2
    module, 91
dpgen2.conf
    module, 91
dpgen2.conf.alloy_conf
    module, 91
dpgen2.conf.conf_generator
    module, 94
dpgen2.conf.file_conf
    module, 95
dpgen2.conf.unit_cells
    module, 95
dpgen2.constants
    module, 177
dpgen2.entrypoint
    module, 97
dpgen2.entrypoint.args
    module, 97
dpgen2.entrypoint.common
    module, 98
dpgen2.entrypoint.download
    module, 98
dpgen2.entrypoint.gui
    module, 98
dpgen2.entrypoint.main
    module, 99
dpgen2.entrypoint.showkey
    module, 99
dpgen2.entrypoint.status
    module, 99
dpgen2.entrypoint.submit
    module, 99
dpgen2.entrypoint.watch
    module, 102
dpgen2.exploration.deviation.deviation_manager
    module, 103
dpgen2.exploration.deviation.deviation_std
    module, 104
dpgen2.exploration.render
    module, 104
dpgen2.exploration.render.traj_render
    module, 104
dpgen2.exploration.render.traj_render_lammps
    module, 105

```

dpigen2.exploration.report
 module, 106
dpigen2.exploration.report.report
 module, 106
dpigen2.exploration.report.report_adaptive_lowed
 module, 107
dpigen2.exploration.report.report_trust_levels_dpgen2.flow.dpgen_loop
 module, 108
dpigen2.exploration.report.report_trust_levels_dpgen2.fp
 module, 109
dpigen2.exploration.report.report_trust_levels_dpgen2.fp.abacus
 module, 110
dpigen2.exploration.report.report_trust_levels_dpgen2.fp.deepmd
 module, 113
dpigen2.exploration.scheduler
 module, 114
dpigen2.exploration.scheduler.convergence_check_dpgen2.scheduler.fsp
 module, 114
dpigen2.exploration.scheduler.scheduler
 module, 116
dpigen2.exploration.scheduler.stage_scheduler
 module, 118
dpigen2.exploration.selector
 module, 119
dpigen2.exploration.selector.conf_filter
 module, 119
dpigen2.exploration.selector.conf_selector
 module, 120
dpigen2.exploration.selector.conf_selector_frame_dpgen2.op.collect_run_caly
 module, 120
dpigen2.exploration.task
 module, 121
dpigen2.exploration.task.caly_task_group
 module, 123
dpigen2.exploration.task.calypso
 module, 121
dpigen2.exploration.task.calypso.caly_input
 module, 121
dpigen2.exploration.task.conf_sampling_task_group_dpgen2.op.prep_lmp
 module, 123
dpigen2.exploration.task.customized_lmp_template_dpgen2.op.select_group_caly_dp_optim
 module, 124
dpigen2.exploration.task.lmp
 module, 122
dpigen2.exploration.task.lmp.lmp_input
 module, 122
dpigen2.exploration.task.lmp_template_task_group_dpgen2.op.run_lmp
 module, 126
dpigen2.exploration.task.make_task_group_from_dpgen2.op.select_confs
 module, 127
dpigen2.exploration.task.npt_task_group
 module, 127
dpigen2.exploration.task.stage
 module, 128
dpigen2.exploration.task.task
 module, 129

dpigen2.exploration.task.task_group
 module, 129
dpigen2.flow
 module, 131
dpigen2.superop
 module, 135
dpigen2.superop.block
 module, 135
dpigen2.superop.caly_evo_step
 module, 136

```

dpigen2.superop.prep_run_calypso
    module, 168
dpigen2.superop.prep_run_dp_train
    module, 169
dpigen2.superop.prep_run_fp
    module, 171
dpigen2.superop.prep_run_lmp
    module, 172
dpigen2.utils
    module, 173
dpigen2.utils.binary_file_input
    module, 173
dpigen2.utils.bohrium_config
    module, 174
dpigen2.utils.chdir
    module, 174
dpigen2.utils.dflow_config
    module, 174
dpigen2.utils.dflow_query
    module, 175
dpigen2.utils.download_dpigen2_artifacts
    module, 175
dpigen2.utils.obj_artifact
    module, 176
dpigen2.utils.run_command
    module, 176
dpigen2.utils.step_config
    module, 176
dpigen_step_config_args() (in module dp-
    gen2.entrypoint.args), 97
dt:
    task_group[lmp-md]/dt (Argument), 68
dump_object_to_file() (in module dp-
    gen2.utils.obj_artifact), 176
E
element_mass:
    fp[fpop_abacus]/inputs_config/element_mass
        (Argument), 65
ens:
    task_group[lmp-md]/ens (Argument), 67
envs:
    default_step_config/template_config/envs
        (Argument), 18
    step_configs/cl_step_config/template_config/envs
        (Argument), 40
    step_configs/collect_data_config/template_config/envs
        (Argument), 37
    step_configs/prep_explore_config/template_config/envs
        (Argument), 26
    step_configs/prep_fp_config/template_config/envs
        (Argument), 30
    step_configs/prep_train_config/template_config/envs
        (Argument), 21
step_configs/run_explore_config/template_config/envs
    (Argument), 28
step_configs/run_fp_config/template_config/envs
    (Argument), 33
step_configs/run_train_config/template_config/envs
    (Argument), 23
step_configs/select_confs_config/template_config/envs
    (Argument), 35
execute() (dpigen2.flow.dpigen_loop.MakeBlockId
    method), 133
execute() (dpigen2.flow.dpigen_loop.SchedulerWrapper
    method), 134
execute() (dpigen2.fp.abacus.PrepFpOpAbacus
    method), 136
execute() (dpigen2.fp.abacus.RunFpOpAbacus
    method), 137
execute() (dpigen2.fp.prep_fp.PrepFp method), 143
execute() (dpigen2.fp.run_fp.RunFp method), 144
execute() (dpigen2.op.collect_data.CollectData
    method), 149
execute() (dpigen2.op.collect_run_caly.CollRunCaly
    method), 151
execute() (dpigen2.op.prep_caly_dp_optim.PrepCalyDPOptim
    method), 153
execute() (dpigen2.op.prep_caly_input.PrepCalyInput
    method), 154
execute() (dpigen2.op.prep_dp_train.PrepDPTrain
    method), 155
dp- execute() (dpigen2.op.prep_lmp.PrepLmp method), 157
dp- execute() (dpigen2.op.run_caly_dp_optim.RunCalyDPOptim
    method), 158
dp- execute() (dpigen2.op.run_caly_model_devi.RunCalyModelDevi
    method), 159
dp- execute() (dpigen2.op.run_dp_train.RunDPTrain
    method), 161
dp- execute() (dpigen2.op.run_lmp.RunLmp method), 163
execute() (dpigen2.op.select_confs.SelectConfs
    method), 164
execute() (dpigen2.superop.prep_run_dp_train.ModifyTrainScript
    method), 169
execute_workflow_subcommand() (in module dp-
    gen2.entrypoint.workflow), 102
executor:
    default_step_config/executor (Argument), 19
    step_configs/cl_step_config/executor (Ar-
        gument), 41
    step_configs/collect_data_config/executor
        (Argument), 38
    step_configs/prep_explore_config/executor
        (Argument), 27
    step_configs/prep_fp_config/executor (Ar-
        gument), 31
    step_configs/prep_train_config/executor
        (Argument), 22

```

```

step_configs/run_explore_config/executor explore[calypso]/configurations/type (Argument)
                                         type:, 59
step_configs/run_fp_config/executor explore[calypso]/configurations[alloy]/atom_pert_dist
                                         (Argument)
                                         atom_pert_dist:, 60
step_configs/run_train_config/executor explore[calypso]/configurations[alloy]/cell_pert_frac
                                         (Argument)
                                         cell_pert_frac:, 60
step_configs/select_confs_config/executor explore[calypso]/configurations[alloy]/concentration
                                         (Argument)
                                         concentration:, 60
expand_idx() (in module dpgen2.entrypoint.common), explore[calypso]/configurations[alloy]/lattice
                                         98                                         (Argument)
expansion_sys_str() (in module dp- explore[calypso]/configurations[alloy]/lattice
gen2.entrypoint.common), 98                                         (Argument)
                                         106
ExplorationReport (class in dp- explore[calypso]/configurations[alloy]/lattice
gen2.exploration.report.report), 106                                         (Argument)
                                         107
ExplorationReportAdaptiveLower (class in dp- explore[calypso]/configurations[alloy]/numb_confs
gen2.exploration.report.report_adaptive_lower), 107                                         (Argument)
ExplorationReportTrustLevels (class in dp- explore[calypso]/configurations[alloy]/replicate
gen2.exploration.report.report_trust_levels_base), 110                                         (Argument)
                                         111
ExplorationReportTrustLevelsMax (class in dp- explore[calypso]/configurations[file]/files
gen2.exploration.report.report_trust_levels_max), 112                                         (Argument)
                                         113
ExplorationReportTrustLevelsRandom (class in dp- explore[calypso]/configurations[file]/fmt
gen2.exploration.report.report_trust_levels_random), 113                                         (Argument)
                                         114
ExplorationScheduler (class in dp- explore[calypso]/configurations[file]/prefix
gen2.exploration.scheduler.scheduler), 116                                         (Argument)
                                         prefix:, 60
ExplorationStage (class in dp- explore[calypso]/configurations[file]/remove_pbc
gen2.exploration.task.stage), 128                                         (Argument)
                                         remove_pbc:, 61
ExplorationTask (class in dp- explore[calypso]/convergence (Argument)
gen2.exploration.task.task), 129                                         convergence:, 55
ExplorationTaskGroup (class in dp- explore[calypso]/convergence/type (Argument)
gen2.exploration.task.task_group), 130                                         type:, 56
explore (Argument) explore[calypso]/convergence[adaptive-lower]/candi_sel_prob
                                         explore:, 48                                         (Argument)
                                         49
explore/type (Argument) explore[calypso]/convergence[adaptive-lower]/conv_tolerance
                                         type:, 48                                         (Argument)
                                         49
explore: explore[calypso]/convergence[adaptive-lower]/level_f_hi
explore (Argument)                                         head:, 55                                         (Argument)
                                         level_f_hi:, 58
explore[calypso]/config (Argument) explore[calypso]/convergence[adaptive-lower]/level_v_hi
                                         config:, 54                                         (Argument)
                                         level_v_hi:, 58
explore[calypso]/config/command (Argument) explore[calypso]/convergence[adaptive-lower]/n_checked_steps
                                         command:, 55                                         (Argument)
                                         n_checked_steps:, 58
explore[calypso]/config/head (Argument) explore[calypso]/convergence[adaptive-lower]/numb_candi_f
                                         head:, 55                                         (Argument)
                                         numb_candi_f:, 59
explore[calypso]/config/shuffle_models (Argument)
                                         shuffle_models:, 55
explore[calypso]/config/teacher_model_path (Argument)
                                         teacher_model_path:, 55
explore[calypso]/configurations (Argument)
                                         configurations:, 59

```


F

level_v_hi:, 51
 explore[lmp]/convergence[adaptive-lower]/n_checked_steps:
 (Argument)
 n_checked_steps:, 52
 explore[lmp]/convergence[adaptive-lower]/numb_candi_f:
 (Argument)
 numb_candi_f:, 51
 explore[lmp]/convergence[adaptive-lower]/numb_candi_v:
 (Argument)
 numb_candi_v:, 51
 explore[lmp]/convergence[adaptive-lower]/rate_candi_f:
 (Argument)
 rate_candi_f:, 51
 explore[lmp]/convergence[adaptive-lower]/rate_candi_v:
 (Argument)
 rate_candi_v:, 52
 explore[lmp]/convergence[fixed-levels-max-select]/conv_accuracy:
 (Argument)
 conv_accuracy:, 51
 explore[lmp]/convergence[fixed-levels-max-select]/level_f_hi:
 (Argument)
 level_f_hi:, 50
 explore[lmp]/convergence[fixed-levels-max-select]/level_f_lo:
 (Argument)
 level_f_lo:, 50
 explore[lmp]/convergence[fixed-levels-max-select]/level_f_hi:
 (Argument)
 level_f_hi:, 50
 explore[lmp]/convergence[fixed-levels-max-select]/level_f_lo:
 (Argument)
 level_f_lo:, 49
 explore[lmp]/convergence[fixed-levels-max-select]/level_v_hi:
 (Argument)
 level_v_hi:, 50
 explore[lmp]/convergence[fixed-levels]/level_f_hi:
 (Argument)
 conv_accuracy:, 50
 explore[lmp]/convergence[fixed-levels]/level_f_hi:
 fmt:
 level_f_hi:, 49
 explore[lmp]/convergence[fixed-levels]/level_f_lo:
 (Argument)
 level_f_lo:, 49
 explore[lmp]/convergence[fixed-levels]/level_v_hi:
 (Argument)
 level_v_hi:, 50
 explore[lmp]/convergence[fixed-levels]/level_v_lo:
 (Argument)
 level_v_lo:, 50
 explore[lmp]/fatal_at_max (Argument)
 fatal_at_max:, 49
 explore[lmp]/max_numb_iter (Argument)
 max_numb_iter:, 48
 explore[lmp]/output_nopbc (Argument)
 output_nopbc:, 49
 explore[lmp]/stages (Argument)
 stages:, 54

failed_ratio() (dpgen2.exploration.report.report_adaptive_lower.Explo-
 method), 110
 failed_ratio() (dpgen2.exploration.report.report_trust_levels_base.Exp-
 method), 111
 fatal_at_max:
 explore[calypso]/fatal_at_max (Argument), 49
 FCC (class in dpgen2.conf.unit_cells), 96
 FileconfGenerator (class in dpgen2.conf.file_conf),
 95
 files() (dpgen2.exploration.task.task.ExplorationTask
 method), 129
 files:
 explore[calypso]/configurations[file]/files
 (Argument), 60
 explore[calypso]/configurations[file]/files
 (Argument), 54
 find_only_one_key() (in module dpgen2.exploration.task.lmp_template_task_group),
 126
 find_only_one_key() (in module dpgen2.op.run_lmp),
 164
 find_slice_ranges() (in module dpgen2.dflw_query), 175
 finetune_args:
 train[dp-dist]/config/finetune_args
 (Argument), 47
 train[dp]/config/finetune_args (Argument),
 44
 task_group/fmax (Argument), 73
 explore[calypso]/configurations[file]/fmax
 (Argument), 60
 explore[calypso]/configurations[file]/fmt
 (Argument), 54
 fold_keys() (in module dpgen2.entrypoint.submit), 99
 FooTask (class in dpgen2.exploration.task.task_group),
 130
 FooTaskGroup (class in dpgen2.exploration.task.task_group), 131
 force_complete() (dp-
 gen2.exploration.scheduler.convergence_check_stage_scheduler.C
 method), 115
 force_complete() (dp-
 gen2.exploration.scheduler.stage_scheduler.StageScheduler
 method), 118
 force_stage_complete() (dp-
 gen2.exploration.scheduler.scheduler.ExplorationScheduler
 method), 117
 fp (Argument)
 fp:, 61

```

fp/type (Argument)
    type:, 61
fp:
    fp (Argument), 61
fp_args() (in module dpgen2.entrypoint.args), 97
FpOpAbacusInputs (class in dpgen2.fp.abacus), 135
fp[deepmd]/inputs_config (Argument)
    inputs_config:, 64
fp[deepmd]/run_config (Argument)
    run_config:, 64
fp[deepmd]/run_config/log (Argument)
    log:, 65
fp[deepmd]/run_config/out (Argument)
    out:, 64
fp[deepmd]/run_config/teacher_model_path (Argument)
    teacher_model_path:, 64
fp[deepmd]/task_max (Argument)
    task_max:, 65
fp[fpop_abacus]/inputs_config (Argument)
    inputs_config:, 65
fp[fpop_abacus]/inputs_config/deepks_descriptor
    (Argument)
    deepks_descriptor:, 66
fp[fpop_abacus]/inputs_config/deepks_model
    (Argument)
    deepks_model:, 66
fp[fpop_abacus]/inputs_config/element_mass
    (Argument)
    element_mass:, 65
fp[fpop_abacus]/inputs_config/input_file (Argument)
    input_file:, 65
fp[fpop_abacus]/inputs_config/kpt_file (Argument)
    kpt_file:, 65
fp[fpop_abacus]/inputs_config/orb_files
    (Argument)
    orb_files:, 66
fp[fpop_abacus]/inputs_config/pp_files (Argument)
    pp_files:, 65
fp[fpop_abacus]/run_config (Argument)
    run_config:, 66
fp[fpop_abacus]/run_config/command (Argument)
    command:, 66
fp[fpop_abacus]/run_config/out (Argument)
    out:, 66
fp[fpop_abacus]/task_max (Argument)
    task_max:, 66
fp[gaussian]/inputs_config (Argument)
    inputs_config:, 62
fp[gaussian]/inputs_config/basis_set (Argument)
    basis_set:, 63
fp[gaussian]/inputs_config/charge (Argument)
    charge:, 63
fp[gaussian]/inputs_config/fragment_guesses
    (Argument)
    fragment_guesses:, 63
fp[gaussian]/inputs_config/keywords (Argument)
    keywords:, 63
fp[gaussian]/inputs_config/keywords_high_multiplicity
    (Argument)
    keywords_high_multiplicity:, 63
fp[gaussian]/inputs_config/multiplicity
    (Argument)
    multiplicity:, 63
fp[gaussian]/inputs_config/nproc (Argument)
    nproc:, 63
fp[gaussian]/run_config (Argument)
    run_config:, 64
fp[gaussian]/run_config/command (Argument)
    command:, 64
fp[gaussian]/run_config/out (Argument)
    out:, 64
fp[gaussian]/task_max (Argument)
    task_max:, 64
fp[vasp]/inputs_config (Argument)
    inputs_config:, 61
fp[vasp]/inputs_config/incar (Argument)
    incar:, 61
fp[vasp]/inputs_config/kgamma (Argument)
    kgamma:, 62
fp[vasp]/inputs_config/kspacing (Argument)
    kspacing:, 62
fp[vasp]/inputs_config/pp_files (Argument)
    pp_files:, 61
fp[vasp]/run_config (Argument)
    run_config:, 62
fp[vasp]/run_config/command (Argument)
    command:, 62
fp[vasp]/run_config/log (Argument)
    log:, 62
fp[vasp]/run_config/out (Argument)
    out:, 62
fp[vasp]/task_max (Argument)
    task_max:, 62
fragment_guesses:
    fp[gaussian]/inputs_config/fragment_guesses
        (Argument), 63

```

G

GaussianInputs (*class in dpgen2.fp.gaussian*), 140
gen_box() (*dpgen2.conf.unit_cells.BCC method*), 96
gen_box() (*dpgen2.conf.unit_cells.DIAMOND method*), 96

gen_box() (*dpgen2.conf.unit_cells.FCC method*), 96
 gen_box() (*dpgen2.conf.unit_cells.HCP method*), 97
 gen_box() (*dpgen2.conf.unit_cells.SC method*), 97
 gen_doc() (*in module dpgen2.conf.alloy_conf*), 93
 gen_doc() (*in module dpgen2.entrypoint.args*), 97
 gen_doc() (*in module dpgen2.utils.step_config*), 176
 generate() (*dpgen2.conf.alloy_conf.AlloyConfGenerator method*), 93
 generate() (*dpgen2.conf.conf_generator.ConfGenerator method*), 94
 generate() (*dpgen2.conf.file_conf.FileConfGenerator method*), 95
 generate_alloy_conf_args() (*in module dpgen2.conf.alloy_conf*), 93
 generate_alloy_conf_file_content() (*in module dpgen2.conf.alloy_conf*), 93
 generate_file_content() (*dpgen2.conf.alloy_conf.AlloyConf method*), 91
 generate_mixed() (*dpgen2.conf.file_conf.FileConfGenerator method*), 95
 generate_std() (*dpgen2.conf.file_conf.FileConfGenerator method*), 95
 generate_systems() (*dpgen2.conf.alloy_conf.AlloyConf method*), 92
 generate_unit_cell() (*in module dpgen2.conf.unit_cells*), 97
 get() (*dpgen2.exploration.deviation.deviation_manager.DevManager method*), 103
 get_all_schedulers() (*in module dpgen2.utils.dflow_query*), 175
 get_candidate_ids() (*dpgen2.exploration.report.report.ExplorationReport method*), 106
 get_candidate_ids() (*dpgen2.exploration.report.report_adaptive_lower.ExplorationReport method*), 110
 get_candidate_ids() (*dpgen2.exploration.report.report_trust_levels_base.ExplorationReport method*), 111
 get_candidate_ids() (*dpgen2.exploration.report.report_trust_levels_max.ExplorationReport method*), 113
 get_candidate_ids() (*dpgen2.exploration.report.report_trust_levels_random.ExplorationReport method*), 114
 get_confs() (*dpgen2.exploration.render.traj_render.TrajRender method*), 104
 get_confs() (*dpgen2.exploration.render.traj_render_lammps.TrajRender method*), 105
 get_convergence_ratio() (*dpgen2.exploration.scheduler.ExplorationScheduler class method*), 170
 get_file_content() (*dpgen2.conf.conf_generator.ConfGenerator method*), 94
 get_input_sign() (*dpgen2.flow.dpgen_loop.MakeBlockId method*), 134
 get_input_sign() (*dpgen2.flow.dpgen_loop.SchedulerWrapper class method*), 134
 get_input_sign() (*dpgen2.fp.abacus.PrepFpOpAbacus method*), 136
 get_input_sign() (*dpgen2.fp.abacus.RunFpOpAbacus method*), 137
 get_input_sign() (*dpgen2.fp.prep_fp.PrepFp method*), 143
 get_input_sign() (*dpgen2.fp.run_fp.RunFp method*), 145
 get_input_sign() (*dpgen2.op.collect_data.CollectData method*), 150
 get_input_sign() (*dpgen2.op.collect_run_caly.CollRunCaly method*), 151
 get_input_sign() (*dpgen2.op.prep_caly_dp_optim.PrepCalyDPOptim class method*), 153
 get_input_sign() (*dpgen2.op.prep_caly_input.PrepCalyInput class method*), 155
 get_input_sign() (*dpgen2.op.prep_dp_train.PrepDPTrain method*), 156
 get_input_sign() (*dpgen2.op.prep_lmp.PrepLmp class method*), 157
 get_input_sign() (*dpgen2.op.run_caly_dp_optim.RunCalyDPOptim class method*), 158
 get_input_sign() (*dpgen2.op.run_caly_model_devi.RunCalyModelDevi class method*), 159
 get_input_sign() (*dpgen2.op.run_dp_train.RunDPTrain method*), 161
 get_input_sign() (*dpgen2.op.run_lmp.RunLmp class method*), 163
 get_input_sign() (*dpgen2.op.select_confs.SelectConfs class method*), 165
 get_input_sign() (*dpgen2.superop.prep_run_dp_train.ModifyTrainScript class method*), 170

get_iteration()	(dp-	gen2.op.run_caly_model_devi.RunCalyModelDevi
gen2.exploration.scheduler.scheduler.ExplorationScheduler class method), 160	method), 117	
get_iteration()	(in module dp-	get_output_sign() (dp-
gen2.utils.dflow_query), 175	gen2.op.run_dp_train.RunDPTTrain class	
get_kspacing_kgamma_from_incar()	(in module dp-	get_output_sign() (dpgen2.op.run_lmp.RunLmp
gen2.entrypoint.submit), 99	class method), 163	
get_last_iteration()	(in module dp-	get_output_sign() (dp-
gen2.utils.dflow_query), 175	gen2.op.select_confs.SelectConfs class	
get_last_scheduler()	(in module dp-	method), 165
gen2.utils.dflow_query), 175	get_output_sign() (dp-	
get_model_devi()	(dp-	gen2.superop.prep_run_dp_train.ModifyTrainScript
gen2.exploration.render.traj_render.TrajRender class method), 105	class method), 170	
get_model_devi()	(dp-	get_reports() (dpgen2.exploration.scheduler.convergence_check_stage_
gen2.exploration.render.traj_render_lammps.TrajRender class method), 105	method), 115	
get_output_sign()	(dp-	get_reports() (dpgen2.exploration.scheduler.stage_scheduler.StageScheduler
gen2.flow.dpgen_loop.MakeBlockId class method), 134	method), 118	
get_output_sign()	(dp-	get_resubmit_keys() (in module dp-
gen2.flow.dpgen_loop.SchedulerWrapper class method), 134	gen2.entrypoint.submit), 99	
get_output_sign()	(dp-	get_scheduler_ids() (in module dp-
gen2.fp.abacus.PrepFpOpAbacus class method), 136	gen2.entrypoint.submit), 99	
get_output_sign()	(dp-	get_stage() (dpgen2.exploration.scheduler.ExplorationScheduler
gen2.fp.abacus.RunFpOpAbacus class method), 137	method), 117	
get_output_sign()	(dp-	get_stage_of_iterations() (dp-
gen2.fp.prep_fp.PrepFp class method), 143	gen2.exploration.scheduler.ExplorationScheduler method), 117	
get_output_sign()	(dp-	get_subkey() (in module dpgen2.utils.dflow_query),
gen2.fp.run_fp.RunFp class method), 145	175	
get_output_sign()	(dp-	get_superop() (in module dpgen2.entrypoint.submit),
gen2.op.collect_data.CollectData class method), 150	99	
get_output_sign()	(dp-	get_value_from_inputdat() (in module dp-
gen2.op.collect_run_caly.CollRunCaly class method), 151	gen2.op.collect_run_caly), 152	
get_output_sign()	(dp-	global_config_workflow() (in module dp-
gen2.op.prep_caly_dp_optim.PrepCalyDPOptim class method), 153	gen2.entrypoint.common), 98	
get_output_sign()	(dp-	group_size:
gen2.op.prep_caly_input.PrepCalyInput class method), 155	default_step_config/template_slice_config/group_size (Argument), 18	
get_output_sign()	(dp-	step_configs/cl_step_config/template_slice_config/group_size (Argument), 40
gen2.op.prep_dp_train.PrepDPTTrain class method), 156	step_configs/collect_data_config/template_slice_config (Argument), 37	
get_output_sign()	(dp-	step_configs/prep_explore_config/template_slice_config (Argument), 26
gen2.op.prep_lmp.PrepLmp class method), 157	step_configs/prep_fp_config/template_slice_config (Argument), 30	
get_output_sign()	(dp-	step_configs/prep_train_config/template_slice_config (Argument), 21
gen2.op.run_caly_dp_optim.RunCalyDPOptim class method), 158	step_configs/run_explore_config/template_slice_config (Argument), 28	
get_output_sign()	(dp-	step_configs/run_fp_config/template_slice_config/group_size (Argument), 33
gen2.op.select_confs_config/template_slice_config (Argument), 23	step_configs/run_train_config/template_slice_config/group_size (Argument), 23	

(*Argument*), 35

H

HCP (*class in dpgen2.conf.unit_cells*), 96

head:

- explore[calypso]/config/head (*Argument*), 55
- explore[lmp]/config/head (*Argument*), 48
- inputs/head (*Argument*), 42
- train[dp-dist]/config/head (*Argument*), 47
- train[dp]/config/head (*Argument*), 45

host:

- bohrium_config/host (*Argument*), 19

|

ialgo:

- task_group/ialgo (*Argument*), 73

icode:

- task_group/icode (*Argument*), 73

image:

- default_step_config/template_config/image (*Argument*), 17
- step_configs/cl_step_config/template_config/image (*Argument*), 39
- step_configs/collect_data_config/template_config/image (*Argument*), 37
- step_configs/prep_explore_config/template_config/image (*Argument*), 25
- step_configs/prep_fp_config/template_config/image (*Argument*), 30
- step_configs/prep_train_config/template_config/image (*Argument*), 20
- step_configs/run_explore_config/template_config/image (*Argument*), 27
- step_configs/run_fp_config/template_config/image (*Argument*), 32
- step_configs/run_train_config/template_config/image (*Argument*), 23
- step_configs/select_confs_config/template_config/image (*Argument*), 34

impl:

- train[dp-dist]/config/impl (*Argument*), 45
- train[dp]/config/impl (*Argument*), 43

incar:

- fp[vasp]/inputs_config/incar (*Argument*), 61
- incar_from_file() (*dpgen2.fp.vasp_input.VaspInputs* method), 148
- incar_template (*dpgen2.fp.vasp_input.VaspInputs* property), 148

init_data_prefix:

- inputs/init_data_prefix (*Argument*), 41

init_data_sys:

- inputs/init_data_sys (*Argument*), 42

init_executor() (*in module dpgen2.utils.step_config*), 176

init_keys (*dpgen2.flow.dpgen_loop.ConcurrentLearning* property), 132

init_model_numb_steps:

- train[dp-dist]/config/init_model_numb_steps (*Argument*), 46
- train[dp]/config/init_model_numb_steps (*Argument*), 44

init_model_old_ratio:

- train[dp-dist]/config/init_model_old_ratio (*Argument*), 46
- train[dp]/config/init_model_old_ratio (*Argument*), 43

init_model_policy:

- train[dp-dist]/config/init_model_policy (*Argument*), 46
- train[dp]/config/init_model_policy (*Argument*), 43

init_model_start_lr:

- train[dp-dist]/config/init_model_start_lr (*Argument*), 46
- train[dp]/config/init_model_start_lr (*Argument*), 44

init_model_start_pref_e:

- train[dp-dist]/config/init_model_start_pref_e (*Argument*), 46
- train[dp]/config/init_model_start_pref_e (*Argument*), 44

init_model_start_pref_f:

- train[dp-dist]/config/init_model_start_pref_f (*Argument*), 46
- train[dp]/config/init_model_start_pref_f (*Argument*), 44

init_model_start_pref_v:

- train[dp-dist]/config/init_model_start_pref_v (*Argument*), 47
- train[dp]/config/init_model_start_pref_v (*Argument*), 44

init_model_with_finetune:

- train[dp-dist]/config/init_model_with_finetune (*Argument*), 47
- train[dp]/config/init_model_with_finetune (*Argument*), 44

init_models_paths:

- train[dp]/init_models_paths (*Argument*), 45

input_args() (*in module dpgen2.entrypoint.args*), 97

input_artifacts (*dpgen2.flow.dpgen_loop.ConcurrentLearning* property), 132

input_artifacts (*dpgen2.flow.dpgen_loop.ConcurrentLearningLoop* property), 133

input_artifacts (*dpgen2.superop.block.ConcurrentLearningBlock* property), 166

<code>input_artifacts</code>	(dp-	<code>input_parameters</code>	(dp-
<code>gen2.superop.caly_evo_step.CalyEvoStep</code>	<code>gen2.superop.prep_run_lmp.PrepRunLmp</code>		
<code>property), 167</code>	<code>property), 173</code>		
<code>input_artifacts</code>	(dp-	<code>input_plm_tmpl_name:</code>	
<code>gen2.superop.prep_run_calypso.PrepRunCaly</code>	<code>task_group[customized-lmp-template]/input_plm_tmpl_name</code>		
<code>property), 168</code>	<code>(Argument), 71</code>		
<code>input_artifacts</code>	(dp-	<code>inputs(Argument)</code>	
<code>gen2.superop.prep_run_dp_train.PrepRunDPTTrain</code>	<code>inputs:, 41</code>		
<code>property), 171</code>	<code>inputs/do_finetune(Argument)</code>		
<code>input_artifacts</code>	(dp-	<code>do_finetune:, 42</code>	
<code>gen2.superop.prep_run_fp.PrepRunFp</code>	<code>inputs/head(Argument)</code>		
<code>property), 172</code>	<code>head:, 42</code>		
<code>input_artifacts</code>	(dp-	<code>inputs/init_data_prefix(Argument)</code>	
<code>gen2.superop.prep_run_lmp.PrepRunLmp</code>	<code>init_data_prefix:, 41</code>		
<code>property), 173</code>	<code>inputs/init_data_sys(Argument)</code>		
<code>input_extra_files:</code>		<code>init_data_sys:, 42</code>	
<code>task_group[customized-lmp-template]/input_</code>	<code>extra_files_map(Argument)</code>		
<code>(Argument), 71</code>	<code>mass_map:, 41</code>		
<code>input_file:</code>		<code>inputs/mixed_type(Argument)</code>	
<code>fp[fpop_abacus]/inputs_config/input_file</code>	<code>mixed_type:, 42</code>		
<code>(Argument), 65</code>	<code>inputs/multi_init_data(Argument)</code>		
<code>input_files()</code>	(<code>dpgen2.fp.deepmd.RunDeepmd</code>	<code>multi_init_data:, 42</code>	
<code>method), 139</code>	<code>method), 139</code>	<code>inputs/multitask(Argument)</code>	
<code>input_files()</code>	(<code>dpgen2.fp.gaussian.RunGaussian</code>	<code>multitask:, 42</code>	
<code>method), 141</code>	<code>method), 141</code>	<code>inputs/type_map(Argument)</code>	
<code>input_files()</code>	(<code>dpgen2.fp.run_fp.RunFp</code>	<code>type_map:, 41</code>	
<code>method), 145</code>	<code>method), 145</code>	<code>inputs/valid_data_prefix(Argument)</code>	
<code>input_files()</code>	(<code>dpgen2.fp.vasp.RunVasp</code>	<code>valid_data_prefix:, 42</code>	
<code>method), 147</code>	<code>method), 147</code>	<code>inputs/valid_data_sys(Argument)</code>	
<code>input_lmp_conf_name:</code>		<code>valid_data_sys:, 42</code>	
<code>task_group[customized-lmp-template]/input_</code>	<code>inputs/valid_data_sys(Argument)</code>		
<code>(Argument), 71</code>	<code>inputs/valid_data_sys:, 42</code>		
<code>input_lmp_tmpl_name:</code>		<code>inputs:</code>	
<code>task_group[customized-lmp-template]/input_</code>	<code>lmp_tmpl_name(Argument), 41</code>		
<code>(Argument), 71</code>	<code>inputs_config:</code>		
<code>input_parameters</code>	(dp-	<code>fp[deepmd]/inputs_config(Argument), 64</code>	
<code>gen2.flow.dpgen_loop.ConcurrentLearning</code>	<code>fp[fpop_abacus]/inputs_config(Argument),</code>		
<code>property), 132</code>	<code>65</code>		
<code>input_parameters</code>	(dp-	<code>fp[gaussian]/inputs_config(Argument), 62</code>	
<code>gen2.flow.dpgen_loop.ConcurrentLearningLoop</code>	<code>fp[vasp]/inputs_config(Argument), 61</code>		
<code>property), 133</code>			
<code>input_parameters</code>	(dp-	K	
<code>gen2.superop.block.ConcurrentLearningBlock</code>		<code>k8s_api_server:</code>	
<code>property), 166</code>		<code>bohrium_config/k8s_api_server(Argument),</code>	
<code>input_parameters</code>	(dp-	<code>20</code>	
<code>gen2.superop.caly_evo_step.CalyEvoStep</code>	<code>keys(dpgen2.flow.dpgen_loop.ConcurrentLearningLoop</code>		
<code>property), 167</code>	<code>property), 133</code>		
<code>input_parameters</code>	(dp-	<code>keys(dpgen2.superop.block.ConcurrentLearningBlock</code>	
<code>gen2.superop.prep_run_calypso.PrepRunCaly</code>	<code>property), 166</code>		
<code>property), 168</code>	<code>keys(dpgen2.superop.caly_evo_step.CalyEvoStep prop-</code>		
<code>input_parameters</code>	(dp-	<code>erty), 167</code>	
<code>gen2.superop.prep_run_dp_train.PrepRunDPTTrain</code>	<code>keys(dpgen2.superop.prep_run_calypso.PrepRunCaly</code>		
<code>property), 171</code>	<code>property), 168</code>		
<code>input_parameters</code>	(dp-	<code>keys(dpgen2.superop.prep_run_dp_train.PrepRunDPTTrain</code>	
<code>gen2.superop.prep_run_fp.PrepRunFp</code>	<code>property), 171</code>		
<code>property), 172</code>			

keys (*dpgen2.superop.prep_run_fp.PrepRunFp* property), 172
keys (*dpgen2.superop.prep_run_lmp.PrepRunLmp* property), 173
keywords:
 fp[gaussian]/inputs_config/keywords (Argument), 63
keywords_high_multiplicity:
 fp[gaussian]/inputs_config/keywords_high_multiplicity (Argument), 63
kgamma:
 fp[vasp]/inputs_config/kgamma (Argument), 62
kpt_file:
 fp[fpop_abacus]/inputs_config/kpt_file (Argument), 65
kspacing:
 fp[vasp]/inputs_config/kspacing (Argument), 62

L

lattice:
 explore[calypso]/configurations[alloy]/lattice (Argument), 59
 explore[lmp]/configurations[alloy]/lattice (Argument), 53
level_f_hi:
 explore[calypso]/convergence[adaptive-lower]/level_f_hi (Argument), 58
 explore[calypso]/convergence[fixed-levels-max-select]/level_f_hi (Argument), 57
 explore[calypso]/convergence[fixed-levels]/level_f_hi (Argument), 56
 explore[lmp]/convergence[adaptive-lower]/level_f_hi (Argument), 51
 explore[lmp]/convergence[fixed-levels-max-select]/level_f_hi (Argument), 50
 explore[lmp]/convergence[fixed-levels]/level_f_hi (Argument), 49
level_f_lo:
 explore[calypso]/convergence[fixed-levels-max-select]/level_f_lo (Argument), 57
 explore[calypso]/convergence[fixed-levels]/level_f_lo (Argument), 56
 explore[lmp]/convergence[fixed-levels-max-select]/level_f_lo (Argument), 50
 explore[lmp]/convergence[fixed-levels]/level_f_lo (Argument), 49
level_v_hi:
 explore[calypso]/convergence[adaptive-lower]/level_v_hi (Argument), 58
 explore[calypso]/convergence[fixed-levels-max-select]/level_v_hi (Argument), 57
 explore[calypso]/convergence[fixed-levels]/level_v_hi (Argument), 57
level_v_lo:
 explore[calypso]/convergence[fixed-levels-max-select]/level_v_lo (Argument), 56
 explore[calypso]/convergence[fixed-levels]/level_v_lo (Argument), 50
 explore[lmp]/convergence[adaptive-lower]/level_v_lo (Argument), 56
 explore[lmp]/convergence[fixed-levels-max-select]/level_v_lo (Argument), 50
 explore[lmp]/convergence[fixed-levels]/level_v_lo (Argument), 50
 lmp_args() (*dpgen2.op.run_lmp.RunLmp* static method), 163
 lmp_args() (in module *dpgen2.entrypoint.args*), 98
 lmp_normalize() (in module *dpgen2.exploration.task.make_task_group_from_config*), 127
 lmp_task_group_args() (in module *dpgen2.exploration.task.make_task_group_from_config*), 127
 lmp_template_fname:
 task_group[lmp-template]/lmp_template_fname (Argument), 69
 lmp_template_task_group_args() (in module *dpgen2.exploration.task.lmp_template_task_group*), 126
 LmpTemplateTaskGroup (class in *dpgen2.exploration.task.lmp_template_task_group*), 126
 load_obj_file_from_file() (in module *dpgen2.utils.obj_artifact*), 176
 log_f_hi
 fp[deepmd]/run_config/log (Argument), 65
 fp[vasp]/run_config/log (Argument), 62
 keep_separate_dpgen_flow_dpgen_loop.ConcurrentLearning property), 132

M

main() (in module *dpgen2.entrypoint.main*), 99
main_parser() (in module *dpgen2.entrypoint.main*), 99
make_Block_optional_parameter() (in module *dpgen2.flow.dpgen_loop*), 134
make_calypso_input() (in module *dpgen2.exploration.task.calypso.calypso_input*), 121
make_separable_parallel_scheduler() (in module *dpgen2.entrypoint.submit*), 99

make_calypso_task_group_from_config() (in module `dpgen2.exploration.task.make_task_group_from_config`), 127
make_collect_data_optional_parameter() (in module `dpgen2.superop.block`), 166
make_concurrent_learning_op() (in module `dpgen2.entrypoint.submit`), 100
make_cont() (`dpgen2.exploration.task.lmp_template_task_group` method), 126
make_finetune_step() (in module `dpgen2.entrypoint.submit`), 102
make_kpoints() (`dpgen2.fp.vasp_input.VaspInputs` method), 148
make_kspacing_kpoints() (in module `dpgen2.fp.vasp_input`), 149
make_link() (in module `dpgen2.entrypoint.args`), 98
make_lmp_input() (in module `dpgen2.exploration.task.lmp.lmp_input`), 122
make_lmp_naive_exploration_scheduler() (in module `dpgen2.entrypoint.submit`), 102
make_lmp_task_group_from_config() (in module `dpgen2.exploration.task.make_task_group_from_config`), 127
make_naive_exploration_scheduler() (in module `dpgen2.entrypoint.submit`), 102
make_optional_parameter() (in module `dpgen2.entrypoint.submit`), 102
make_potcar() (`dpgen2.fp.vasp_input.VaspInputs` method), 148
make_run_dp_train_optional_parameter() (in module `dpgen2.superop.block`), 166
make_task() (`dpgen2.exploration.task.caly_task_group.CalyTask` method), 123
make_task() (`dpgen2.exploration.task.customized_lmp_template` method), 125
make_task() (`dpgen2.exploration.task.lmp_template_task_group` method), 126
make_task() (`dpgen2.exploration.task.npt_task_group.NPTTask` method), 128
make_task() (`dpgen2.exploration.task.stage.ExplorationStage` method), 128
make_task() (`dpgen2.exploration.task.task_group.ExplorationTask` method), 130
make_task_group_from_config() (in module `dpgen2.exploration.task.make_task_group_from_config`), 127
MakeBlockId (class in `dpgen2.flow.dpgen_loop`), 133
mass_map:
 inputs/mass_map (Argument), 41
matched_step_key() (in module `dpgen2.utils.dflow_query`), 175
MAX_DEVI_F (`dpgen2.exploration.deviation_manager.DeviManager` attribute), 103
 dp- gen2.exploration.deviation_manager.DeviManager, 103
 MAX_DEVI_V (`dpgen2.exploration.deviation.deviation_manager.DeviManager` attribute), 103
 max_numb_atoms:
 task_group/max_numb_atoms (Argument), 75
 max_numb_iter:
 explore[calypso]/max_numb_iter (Argument), 55
 explore[calypso]/max_numb_iter (Argument), 48
 max_step:
 task_group/max_step (Argument), 73
 max_time:
 task_group/max_time (Argument), 74
MDSettings (class in `dpgen2.op.md_settings`), 152
MIN_DEVI_F (`dpgen2.exploration.deviation_manager.DeviManager` attribute), 103
 dp- gen2.exploration.deviation_manager.DeviManager, 103
 MIN_DEVI_V (`dpgen2.exploration.deviation.deviation_manager.DeviManager` attribute), 103
 mixed_type:
 inputs/mixed_type (Argument), 42
ModifyTrainScript (class in `dpgen2.superop.prep_run_dp_train`), 169
 dp- gen2.superop.prep_run_dp_train, 169
 module
 dpgen2.conf, 91
 dpgen2.conf.alloy_conf, 91
 dpgen2.conf.conf_generator, 94
 dpgen2.conf.file_conf, 95
 dpgen2.conf.unit_cells, 95
 dpgen2.constants, 177
 dpgen2.entrypoint, 97
 dpgen2.entrypoint.args, 97
 dpgen2.entrypoint.common, 98
 dpgen2.entrypoint.download, 98
 dpgen2.entrypoint.gui, 98
 dpgen2.entrypoint.showkey, 99
 dpgen2.entrypoint.submit, 99
 dpgen2.entrypoint.watch, 102
 dpgen2.entrypoint.workflow, 102
 dpgen2.exploration, 103
 dpgen2.exploration.deviation, 103
 dpgen2.exploration.deviation.deviation_manager, 103
 dpgen2.exploration.deviation.deviation_std, 104
 dpgen2.exploration.render, 104
 dpgen2.exploration.render.traj_render, 104
 dpgen2.exploration.render.traj_render_lammps, 105
 dpgen2.exploration.report, 106
 dpgen2.exploration.report.report, 106

```
dpigen2.exploration.report.report_adaptive_low, 149
    107                                         dpigen2.op.collect_data, 149
dpigen2.exploration.report.report_trust_levels, 150
    110                                         dpigen2.op.collect_run_caly, 150
                                                dpigen2.op.md_settings, 152
dpigen2.exploration.report.report_trust_levels, 152
    112                                         dpigen2.op.prep_caly_dp_optim, 152
dpigen2.exploration.report.report_trust_levels, 155
    113                                         dpigen2.op.prep_dp_train, 155
                                                dpigen2.op.prep_lmp, 156
dpigen2.exploration.scheduler, 114           dpigen2.op.run_caly_dp_optim, 157
dpigen2.exploration.scheduler.convergence_check, 159
    114                                         dpigen2.op.schedule_early_model_devi, 159
dpigen2.exploration.scheduler.scheduler, 160
    116                                         dpigen2.op.run_dp_train, 160
dpigen2.exploration.scheduler.stage_scheduler, 165
    118                                         dpigen2.op.run_lmp, 162
                                                dpigen2.op.select_confs, 164
dpigen2.exploration.selector, 119           dpigen2.superop, 165
dpigen2.exploration.selector.conf_filter, 166
    119                                         dpigen2.superop.block, 165
dpigen2.exploration.selector.conf_selector, 166
    120                                         dpigen2.superop.caly_evo_step, 166
dpigen2.exploration.selector.conf_selector, 168
    120                                         dpigen2.superop.prep_run_calypso, 168
dpigen2.exploration.task, 121               dpigen2.superop.prep_run_dp_train, 169
dpigen2.exploration.task.caly_task_group, 169
    123                                         dpigen2.superop.prep_run_fp, 171
dpigen2.exploration.task.calypso, 121        dpigen2.superop.prep_run_lmp, 172
dpigen2.exploration.task.calypso.caly_input, 173
    121                                         dpigen2.utils, 173
dpigen2.exploration.task.conf_sampling_task_group, 173
    123                                         dpigen2.utils.binary_file_input, 173
dpigen2.exploration.task.customized_lmp_template, 174
    124                                         dpigen2.utils.bohrium_config, 174
                                                dpigen2.utils.chdir, 174
                                                dpigen2.utils.dflow_config, 174
                                                dpigen2.utils.dflow_query, 175
dpigen2.exploration.task.lmp, 122           dpigen2.utils.download_dpgen2_artifacts, 175
dpigen2.exploration.task.lmp.lmp_input, 176
    122                                         dpigen2.utils.obj_artifact, 176
dpigen2.exploration.task.lmp_template_task_group, 176
    126                                         dpigen2.utils.run_command, 176
dpigen2.exploration.task.make_task_group_from_config, 45
    127                                         dpigen2.utils.task_group, step_config, 176
                                                multi_init_data:
                                                inputs/multi_init_data (Argument), 42
dpigen2.exploration.task.npt_task_group, 63
    127                                         multi_init_data_idx:
                                                train[dp-dist]/config/multi_init_data_idx
dpigen2.exploration.task.stage, 128          train[dp]/config/multi_init_data_idx (Argument), 47
dpigen2.exploration.task.task, 129           dpigen2.exploration.task.make_task_group_from_config, 45
                                                multiplicity:
                                                fp[gaussian]/inputs_config/multiplicity
dpigen2.exploration.task.task_group, 129       (Argument), 63
dpigen2.flow, 131                           multitask:
dpigen2.flow.dpgen_loop, 131                 inputs/multitask (Argument), 42
dpigen2.fp, 135                           train[dp-dist]/config/multitask (Argument), 47
dpigen2.fp.abacus, 135                      train[dp]/config/multitask (Argument), 44
dpigen2.fp.deepmd, 137
dpigen2.fp.gaussian, 140
dpigen2.fp.prep_fp, 142
dpigen2.fp.run_fp, 144
dpigen2.fp.vasp, 146
dpigen2.fp.vasp_input, 148
```

N

```
n_checked_steps:
    explore[calypso]/convergence[adaptive-lower]/n_checked_steps,
        (Argument), 58
    explore[lmp]/convergence[adaptive-lower]/n_checked_steps,
        (Argument), 52
n_sample:
```

```

task_group[customized-lmp-template]/n_sample      method), 96
    (Argument), 70
task_group[lmp-md]/n_sample (Argument), 67
task_group[lmp-template]/n_sample (Argument),
    69
name (Argument)
    name:, 66
name:
    name (Argument), 66
name_of_atoms:
    task_group/name_of_atoms (Argument), 72
neidelay:
    task_group[lmp-md]/neidelay (Argument), 68
next_iteration() (dp-
    gen2.exploration.scheduler.convergence_check_stage_scheduler,
    method), 115
next_iteration() (dp-
    gen2.exploration.scheduler.StageScheduler,
    method), 118
no_candidate() (dpgen2.exploration.report.report.Explor
    method), 107
no_pbc:
    task_group[lmp-md]/no_pbc (Argument), 68
normalize() (in module dpgen2.conf.alloy_conf), 94
normalize() (in module dpgen2.entrypoint.args), 98
normalize() (in module dpgen2.utils.step_config), 176
normalize_config() (dp-
    gen2.conf.conf_generator.ConfGenerator
    class method), 94
normalize_config() (dpgen2.fp.run_fp.RunFp
    class method), 145
normalize_config() (dp-
    gen2.fp.vasp_input.VaspInputs static method),
    148
normalize_config() (dp-
    gen2.op.collect_run_caly.CollRunCaly
    static method), 152
normalize_config() (dp-
    gen2.op.run_dp_train.RunDPTTrain
    static method), 162
normalize_config() (dpgen2.op.run_lmp.RunLmp
    static method), 163
nproc:
    fp[gaussian]/inputs_config/nproc (Argument),
        63
npt_task_group_args() (in module dp-
    gen2.exploration.task.make_task_group_from_config),
    127
NPTTaskGroup (class in dp-
    gen2.exploration.task.npt_task_group), 127
nsteps:
    task_group[lmp-md]/nsteps (Argument), 68
numb_atoms() (dpgen2.conf.unit_cells.BCC method), 96
numb_atoms() (dpgen2.conf.unit_cells.DIAMOND
    method), 96
numb_atoms() (dpgen2.conf.unit_cells.FCC method), 96
numb_atoms() (dpgen2.conf.unit_cells.HCP method), 97
numb_atoms() (dpgen2.conf.unit_cells.SC method), 97
numb_candi_f:
    explore[calypso]/convergence[adaptive-lower]/numb_candi
        f (Argument), 58
    explore[lmp]/convergence[adaptive-lower]/numb_candi_f
        (Argument), 51
numb_candi_v:
    explore[calypso]/convergence[adaptive-lower]/numb_candi
        v (Argument), 58
    explore[lmp]/convergence[adaptive-lower]/numb_candi_v
        (Argument), 51
numb_models:
    train[dp]/numb_models (Argument), 45
numb_of_atoms:
    task_group/numb_of_atoms (Argument), 72
numb_of_formula:
    task_group/numb_of_formula (Argument), 73
numb_of_lbtest:
    task_group/numb_of_lbtest (Argument), 73
numb_of_local_optim:
    task_group/numb_of_local_optim (Argument),
        74
numb_of_species:
    task_group/numb_of_species (Argument), 72
O
optional_input_files() (dp-
    gen2.fp.deepmd.RunDeepmd method), 139
optional_input_files() (dp-
    gen2.fp.gaussian.RunGaussian method),
    141
optional_input_files() (dpgen2.fp.run_fp.RunFp
    method), 145
optional_input_files() (dpgen2.fp.vasp.RunVasp
    method), 147
orb_files:
    fp[fpop_abacus]/inputs_config/orb_files
        (Argument), 66
out:
    fp[deepmd]/run_config/out (Argument), 64
    fp[fpop_abacus]/run_config/out (Argument),
        66
    fp[gaussian]/run_config/out (Argument), 64
    fp[vasp]/run_config/out (Argument), 62
output_artifacts (dp-
    gen2.flow.dpgen_loop.ConcurrentLearning
        method), 62

```

```

    property), 132
output_artifacts           (dp-
    gen2.flow.dpgen_loop.ConcurrentLearningLoop property), 133
output_artifacts           (dp-
    gen2.superop.block.ConcurrentLearningBlock property), 166
output_artifacts           (dp-
    gen2.superop.caly_evo_step.CalyEvoStep property), 167
output_artifacts           (dp-
    gen2.superop.prep_run_calypso.PrepRunCaly property), 169
output_artifacts           (dp-
    gen2.superop.prep_run_dp_train.PrepRunDPTrain property), 171
output_artifacts           (dp-
    gen2.superop.prep_run_fp.PrepRunFp property), 172
output_artifacts           (dp-
    gen2.superop.prep_run_lmp.PrepRunLmp property), 173
output_dir_pattern:
    task_group[customized-lmp-template]/output_dir_
        (Argument), 71
output_lmp_conf_name:
    task_group[customized-lmp-template]/output_lmp_
        (Argument), 71
output_lmp_tmpl_name:
    task_group[customized-lmp-template]/output_lmp_tmpl_
        (Argument), 71
output_nopbc:
    explore[calypso]/output_nopbc (Argument),
        55
    explore[lmp]/output_nopbc (Argument), 49
output_parameters           (dp-
    gen2.flow.dpgen_loop.ConcurrentLearning property), 132
output_parameters           (dp-
    gen2.flow.dpgen_loop.ConcurrentLearningLoop property), 133
output_parameters           (dp-
    gen2.superop.block.ConcurrentLearningBlock property), 166
output_parameters           (dp-
    gen2.superop.caly_evo_step.CalyEvoStep property), 167
output_parameters           (dp-
    gen2.superop.prep_run_calypso.PrepRunCaly property), 169
output_parameters           (dp-
    gen2.superop.prep_run_dp_train.PrepRunDPTrain property), 171
output_parameters           (dp-
    gen2.superop.prep_run_fp.PrepRunFp property), 172
output_parameters           (dp-
    gen2.superop.prep_run_lmp.PrepRunLmp property), 173
output_plm_tmpl_name:
    task_group[customized-lmp-template]/output_plm_tmpl_na-
        (Argument), 72

```

P

```

parallel:
    task_group/parallel (Argument), 74
parallelism:
    default_step_config/parallelism (Argu-
        ment), 19
    step_configs/cl_step_config/parallelism
        (Argument), 41
    step_configs/collect_data_config/parallelism
        (Argument), 38
    step_configs/prep_explore_config/parallelism
        (Argument), 27
    step_configs/prep_fp_config/parallelism
        (Argument), 31
    step_configs/prep_train_config/parallelism
        (Argument), 22
    step_configs/run_explore_config/parallelism
        (Argument), 29
    step_configs/run_fp_config/parallelism
        (Argument), 33
    step_configs/run_train_config/parallelism
        (Argument), 24
    step_configs/select_confs_config/parallelism
        (Argument), 36
parse_args() (in module dpgen2.entrypoint.main), 99
parse_traj() (in module dpgen2.op.run_caly_model_devi), 160
password:
    bohrium_config/password (Argument), 19
pick_step:
    task_group/pick_step (Argument), 74
pick_up:
    task_group/pick_up (Argument), 74
pka_e:
    task_group[lmp-md]/pka_e (Argument), 68
plan_next_iteration() (dp-
    gen2.exploration.scheduler.convergence_check_stage_scheduler.
        method), 115
plan_next_iteration() (dp-
    gen2.exploration.scheduler.ExplorationScheduler
        method), 117
plan_next_iteration() (dp-
    gen2.exploration.scheduler.stage_scheduler.StageScheduler
        method), 118
plm_template_fname:

```

```

task_group[lmp-template]/plm_template_fnamp;prep_last_calypso_file() (in module dp-
    (Argument), 69
gen2.op.collect_run_caly), 152
pool_size:
    default_step_config/template_slice_config/pool_size88 prep_task() (dpgen2.fp.deepmd.PrepDeepmd method),
    (Argument), 18
step_configs/cl_step_config/template_slice_config/pool_size140 prep_task() (dpgen2.fp.gaussian.PrepGaussian
    (Argument), 40
step_configs/collect_data_config/template_slice_config/pool_size142 prep_task() (dpgen2.fp.prep_fp.PrepFp method), 143
    (Argument), 38
step_configs/prep_explore_config/template_slice_config/pool_size146 prep_task() (dpgen2.fp.prep_fp.PrepVasp method), 146
    (Argument), 26
step_configs/prep_fp_config/template_sliceConfig/pool_size20 prep_train_config:
    (Argument), 20
step_configs/prep_fp_config/template_slicePrepFpConfig/pool_size31 (class in dpgen2.op.prep_caly_dp_optim), 152
    (Argument), 31
step_configs/prep_train_config/template_slicePrepCalyConfig/pool_size21 (class in dpgen2.op.prep_caly_input), 154
    (Argument), 21
step_configs/run_explore_config/template_slicePrepDeepmdConfig/pool_size28 PrepDPTrain (class in dpgen2.op.prep_dp_train), 155
    (Argument), 28
step_configs/run_fp_config/template_slicePrepFpConfig/pool_size33 TaskGroup (in module dp-
    (Argument), 33
gen2.op.prep_lmp), 156
step_configs/run_train_config/template_slicePrepFpConfig/pool_size24 PrepFp0pAbacus (class in dpgen2.fp.abacus), 135
    (Argument), 24
step_configs/select_confs_config/template_BigCausonFan/pool_size35 PrepLmp (class in dpgen2.op.prep_lmp), 156
    (Argument), 35
pop_size:
    task_group/pop_size (Argument), 72
poscar_unit() (dpgen2.conf.unit_cells.BCC method), 96
poscar_unit() (dpgen2.conf.unit_cells.DIAMOND
    method), 96
poscar_unit() (dpgen2.conf.unit_cells.FCC method), 96
poscar_unit() (dpgen2.conf.unit_cells.HCP method), 97
poscar_unit() (dpgen2.conf.unit_cells.SC method), 97
potcars (dpgen2.fp.vasp_input.VaspInputs property), 148
potcars_from_file() (dp-
    gen2.fp.vasp_input.VaspInputs
    method), 149
pp_files:
    fp[fpop_abacus]/inputs_config/pp_files print() (dpgen2.exploration.report.report.ExplorationReport
    (Argument), 65
    method), 107
    fp[vasp]/inputs_config/pp_files print() (dpgen2.exploration.report.report_adaptive_lower.ExplorationReport
    (Argument), 61
    method), 110
prefix:
    explore[calypso]/configurations[file]/prefix print() (dpgen2.exploration.report.report.ExplorationReport
    (Argument), 60
    method), 107
    explore[lmp]/configurations[file]/prefix print() (dpgen2.exploration.report.report_adaptive_lower.ExplorationReport
    (Argument), 54
    method), 110
prep_explore_config:
    step_configs/prep_explore_config print() (dpgen2.exploration.report.report.ExplorationReport
    (Argument), 25
    method), 111
prep_fp_config:
    step_configs/prep_fp_config (Argument), 29
    print_keys_in_nice_format() (in module dp-
        gen2.utils.dflow_query), 175
    print_last_iteration() (dp-
        gen2.exploration.scheduler.ExplorationScheduler
        method), 117

```

method), 117
print_list_steps() (in module *dpgen2.entrypoint.submit*), 102
print_op_download_setting() (in module *dpgen2.utils.download_dpgen2_artifacts*), 176
project_id:
 bohrium_config/project_id (Argument), 19
pso_ratio:
 task_group/pso_ratio (Argument), 73

R

rate_candi_f:
 explore[calypso]/convergence[adaptive-lower]/rate_candi_f (Argument), 58
explore[lmp]/convergence[adaptive-lower]/rate_candi_f (Argument), 51

rate_candi_v:
 explore[calypso]/convergence[adaptive-lower]/rate_candi_v (Argument), 58
 explore[lmp]/convergence[adaptive-lower]/rate_candi_v (Argument), 52

reached_max_iteration() (in module *dpgen2.exploration.scheduler.convergence_check_stage_scheduler*), 116
record() (*dpgen2.exploration.report.ExplorationReport* method), 107
record() (*dpgen2.exploration.report.report_adaptive_lower.ExplorationReportAdaptiveLower* method), 110
record() (*dpgen2.exploration.report.report_trust_levels_base.ExplorationReportTrustLevelsBase* method), 112
relative_f_epsilon:
 task_group[lmp-md]/relative_f_epsilon (Argument), 69
relative_v_epsilon:
 task_group[lmp-md]/relative_v_epsilon (Argument), 69

remove_pbc:
 explore[calypso]/configurations[file]/remove_pbc (Argument), 61
 explore[lmp]/configurations[file]/remove_pbc (Argument), 54

replicate:
 explore[calypso]/configurations[alloy]/replicate (Argument), 60
 explore[lmp]/configurations[alloy]/replicate (Argument), 53

repo_key:
 bohrium_config/repo_key (Argument), 20

resubmit_concurrent_learning() (in module *dpgen2.entrypoint.submit*), 102

retry_on_transient_error:
 default_step_config/template_config/retry_on_transient_error (Argument), 17

step_configs/cl_step_config/template_config/retry_on_transient_error (Argument), 39
step_configs/collect_data_config/template_config/retry_on_transient_error (Argument), 37
step_configs/prep_explore_config/template_config/retry_on_transient_error (Argument), 25
step_configs/prep_fp_config/template_config/retry_on_transient_error (Argument), 30
step_configs/prep_train_config/template_config/retry_on_transient_error (Argument), 21
step_configs/run_explore_config/template_config/retry_on_transient_error (Argument), 28
step_configs/run_fp_config/template_config/retry_on_transient_error (Argument), 32
step_configs/run_train_config/template_config/retry_on_transient_error (Argument), 23
step_configs/select_confs_config/template_config/retry_on_transient_error (Argument), 35
revise_by_keys() (in module *dpgen2.exploration.task.lmp_template_task_group*), 126
revise_lmp_input_dump() (in module *dpgen2.exploration.task.lmp_template_task_group*), 126
revise_lmp_input_model() (in module *dpgen2.exploration.task.lmp_template_task_group*), 126
revise_lmp_input_plm() (in module *dpgen2.exploration.task.lmp_template_task_group*), 126
revisions:
 task_group[customized-lmp-template]/revisions (Argument), 70
 task_group[lmp-template]/revisions (Argument), 70
run_command() (in module *dpgen2.utils.run_command*), 176
run_pbc_config:
 fp[deepmd]/run_config (Argument), 64
 fp[fpop_abacus]/run_config (Argument), 66
 fp[gaussian]/run_config (Argument), 64
 fp[vasp]/run_config (Argument), 62
run_explore_config:
 step_configs/run_explore_config (Argument), 27
run_fp_config:
 step_configs/run_fp_config (Argument), 32
run_task() (*dpgen2.fp.deepmd.RunDeepmd* method), 139
run_task() (*dpgen2.fp.gaussian.RunGaussian* method), 142
run_transient_error() (*dpgen2.fp.RunFp* method), 145
run_task() (*dpgen2.fp.vasp.RunVasp* method), 148
run_train_config:

`step_configs/run_train_config` (*Argument*), 22
`RunCalyDPOptim` (class in `gen2.op.run_caly_dp_optim`), 157
`RunCalyModelDevi` (class in `gen2.op.run_caly_model_devi`), 159
`RunDeepmd` (class in `dpgen2.fp.deepmd`), 138
`RunDPTTrain` (class in `dpgen2.op.run_dp_train`), 160
`RunFp` (class in `dpgen2.fp.run_fp`), 144
`RunFpOpAbacus` (class in `dpgen2.fp.abacus`), 136
`RunGaussian` (class in `dpgen2.fp.gaussian`), 141
`RunLmp` (class in `dpgen2.op.run_lmp`), 162
`RunVasp` (class in `dpgen2.fp.vasp`), 147

S

`save_as_file()` (`dpgen2.utils.binary_file_input.BinaryFileInput`, *method*), 173
`SC` (class in `dpgen2.conf.unit_cells`), 97
`SchedulerWrapper` (class in `dpgen2.flow.dpgen_loop`), 134
`select()` (`dpgen2.exploration.selector.conf_selector`, *method*), 120
`select()` (`dpgen2.exploration.selector.conf_selector_frame`, *method*), 121
`select_confs_config:`
 `step_configs/select_confs_config` (*Argument*), 34
`SelectConfs` (class in `dpgen2.op.select_confs`), 164
`set_conf()` (`dpgen2.exploration.task.conf_sampling_task_group`, *method*), 124
`set_directory()` (in module `dpgen2.utils.chdir`), 174
`set_lmp()` (`dpgen2.exploration.task.customized_lmp_template_task_group`, *method*), 125
`set_lmp()` (`dpgen2.exploration.task.lmp_template_task_group`, *method*), 126
`set_md()` (`dpgen2.exploration.task.npt_task_group`, *method*), 128
`set_models()` (in module `dpgen2.op.run_lmp`), 164
`set_params()` (`dpgen2.exploration.task.caly_task_group`, *method*), 123
`showkey()` (in module `dpgen2.entrypoint.showkey`), 99
`shuffle_models:`
 `explore[calypso]/config/shuffle_models` (*Argument*), 55
 `explore[lmp]/config/shuffle_models` (*Argument*), 48
`skip_training()` (`gen2.op.run_dp_train.RunDPTTrain`, *method*), 162
`sort_slice_ops()` (in module `gen2.utils.dflow_query`), 175
`spec_space_group:`
 `task_group/spec_space_group` (*Argument*), 74
`split:`

`task_group/split` (*Argument*), 74
`stages:`
 `explore[calypso]/stages` (*Argument*), 61
 `explore[lmp]/stages` (*Argument*), 54
`StageScheduler` (class in `dp-gen2.exploration.scheduler.stage_scheduler`), 118
`start_dpgui()` (in module `dpgen2.entrypoint.gui`), 98
`status()` (in module `dpgen2.entrypoint.status`), 99
`step_conf_args()` (in module `dp-gen2.utils.step_config`), 176
`step_configs` (*Argument*)
 `step_configs:`, 20
`step_configs/cl_step_config` (*Argument*)
 `cl_step_config:`, 39
`step_configs/cl_step_config/continue_on_failed` (*Argument*)
 `continue_on_failed:`, 40
`step_configs/cl_step_config/continue_on_num_success` (*Argument*)
 `continue_on_num_success:`, 40
 `step_configs/cl_step_config/continue_on_success_ratio` (*Argument*)
 `continue_on_success_ratio:`, 40
`step_configs/cl_step_config/executor` (*Argument*)
 `executor:`, 41
`step_configs/cl_step_config/executor/type` (*Argument*)
`TaskGroup` (*Argument*)
 `type:`, 41
`step_configs/cl_step_config/parallelism` (*Argument*)
`CustomizedLmpTemplateTaskGroup` (*Argument*)
 `parallelism:`, 41
`step_configs/cl_step_config/template_config` (*Argument*)
`NPTTaskGroup` (*Argument*)
 `template_config:`, 39
`step_configs/cl_step_config/template_config/envs` (*Argument*)
`CalyTaskGroup` (*Argument*)
 `image:`, 39
`step_configs/cl_step_config/template_config/image` (*Argument*)
 `image:`, 39
`step_configs/cl_step_config/template_config/retry_on_transient_error` (*Argument*)
 `retry_on_transient_error:`, 39
`step_configs/cl_step_config/template_config/timeout` (*Argument*)
 `timeout:`, 39
`step_configs/cl_step_config/template_config/timeout_as_transient_error` (*Argument*)
 `timeout_as_transient_error:`, 39
`step_configs/cl_step_config/template_slice_config` (*Argument*)
 `template_slice_config:`, 40

```

step_configs/cl_step_config/template_slice_config/group_exploration_config:, 25
    (Argument)
    group_size:, 40
step_configs/cl_step_config/template_slice_config/prep_explore_config/continue_on_failed
    (Argument)
    step_configs/prep_explore_config/continue_on_failed
        (Argument)
        group_size:, 40
step_configs/collect_data_config(Argument)
    collect_data_config:, 36
step_configs/collect_data_config/continue_on_failed (Argument)
    continue_on_failed:, 38
step_configs/collect_data_config/continue_on_num_success (Argument)
    continue_on_num_success:, 38
step_configs/collect_data_config/continue_on_num_success_ratio (Argument)
    continue_on_num_success_ratio:, 38
step_configs/collect_data_config/executor (Argument)
    executor:, 38
step_configs/collect_data_config/executor/type (Argument)
    type:, 39
step_configs/collect_data_config/parallelism (Argument)
    parallelism:, 38
step_configs/collect_data_config/template_config (Argument)
    template_config:, 37
step_configs/collect_data_config/template_config/envs (Argument)
    envs:, 37
step_configs/collect_data_config/template_config/image (Argument)
    image:, 37
step_configs/collect_data_config/template_config/retry_transient_error (Argument)
    retry_on_transient_error:, 37
step_configs/collect_data_config/template_config/timeout (Argument)
    timeout:, 37
step_configs/collect_data_config/template_config/timeout_as_transient_error (Argument)
    timeout_as_transient_error:, 37
step_configs/collect_data_config/template_slice_config (Argument)
    template_slice_config:, 37
step_configs/collect_data_config/template_slice_config/group_size (Argument)
    group_size:, 37
step_configs/collect_data_config/template_slice_config/pool_size (Argument)
    pool_size:, 38
step_configs/collect_data_config/template_slice_config/prep_fp_config (Argument)
    prep_fp_config/group_size:, 29
step_configs/collect_data_config/template_slice_config/prep_fp_config/continue_on_failed
    (Argument)
    group_size:, 37
step_configs/collect_data_config/template_slice_config/prep_fp_config/continue_on_num_success
    (Argument)
    pool_size:, 38
step_configs/prep_explore_config(Argument)
    continue_on_num_success:, 31

```

```

step_configs/prep_fp_config/continue_on_success_ratio(Argument)
    (Argument)
    continue_on_success_ratio:, 31
step_configs/prep_fp_config/executor (Argument)
    executor:, 31
step_configs/prep_fp_config/executor/type
    (Argument)
    type:, 31
step_configs/prep_fp_config/parallelism
    (Argument)
    parallelism:, 31
step_configs/prep_fp_config/template_config
    (Argument)
    template_config:, 30
step_configs/prep_fp_config/template_config/envs
    (Argument)
    envs:, 30
step_configs/prep_fp_config/template_config/image
    (Argument)
    image:, 30
step_configs/prep_fp_config/template_config/retry_on_transient_error
    (Argument)
    retry_on_transient_error:, 30
step_configs/prep_fp_config/template_config/timeout (Argument)
    (Argument)
    timeout:, 30
step_configs/prep_fp_config/template_config/timeout_as_transient_error
    (Argument)
    timeout_as_transient_error:, 30
step_configs/prep_fp_config/template_slice_config/group_size
    (Argument)
    group_size:, 31
step_configs/prep_fp_config/template_slice_config/pool_size
    (Argument)
    pool_size:, 31
step_configs/run_explore_config (Argument)
group_size:, 27
step_configs/run_explore_config/continue_on_failed
    (Argument)
    group_size:, 30
step_configs/prep_fp_config/template_slice_config/continue_on_failed:, 28
    (Argument)
    pool_size:, 31
step_configs/prep_train_config (Argument)
    prep_train_config:, 20
step_configs/prep_train_config/continue_on_failed
    (Argument)
    continue_on_failed:, 21
step_configs/prep_train_config/continue_on_num_success
    (Argument)
    continue_on_num_success:, 22
step_configs/prep_train_config/continue_on_success_ratio
    (Argument)
    continue_on_success_ratio:, 22
step_configs/prep_train_config/executor
    (Argument)
    executor:, 22
step_configs/prep_train_config/executor/type
    (Argument)
    type:, 22
step_configs/run_explore_config/parallelism
    (Argument)
    parallelism:, 22
step_configs/run_explore_config/template_config
    (Argument)
    template_config:, 20
step_configs/run_explore_config/envs
    (Argument)
    envs:, 21
step_configs/run_explore_config/template_config/image
    (Argument)
    image:, 20
step_configs/run_explore_config/template_config/retry_on_transient_error
    (Argument)
    retry_on_transient_error:, 21
step_configs/run_explore_config/template_config/timeout
    (Argument)
    timeout:, 21
step_configs/run_explore_config/template_config/timeout_as_transient_error
    (Argument)
    timeout_as_transient_error:, 21
step_configs/run_explore_config/template_slice_config/group_size
    (Argument)
    group_size:, 21
step_configs/run_explore_config/template_slice_config/pool_size
    (Argument)
    pool_size:, 21
step_configs/run_explore_config/continue_on_failed
    (Argument)
    group_size:, 28
step_configs/run_explore_config/continue_on_num_success
    (Argument)
    continue_on_num_success:, 29
step_configs/run_explore_config/continue_on_success_ratio
    (Argument)
    continue_on_success_ratio:, 29
step_configs/run_explore_config/executor (Argument)
    executor:, 29
step_configs/run_explore_config/executor/type
    (Argument)
    type:, 29
step_configs/run_explore_config/parallelism
    (Argument)
    parallelism:, 29
step_configs/run_explore_config/template_config
    (Argument)
    template_config:, 20

```

```

    template_config:, 27           step_configs/run_fp_config/template_config/retry_on_transient_error:, 32
step_configs/run_explore_config/template_config/envs (Argument)
    (Argument)                   step_configs/run_fp_config/template_config/timeout
envs:, 28                      step_configs/run_fp_config/template_config/timeout_as_transient_error:, 32
step_configs/run_explore_config/template_config/image (Argument)
    (Argument)                   step_configs/run_fp_config/template_config/timeout_as_transient_error:, 32
image:, 27                      step_configs/run_fp_config/template_slice_config
step_configs/run_explore_config/template_config/retry_on_transient_error (Argument)
    (Argument)                   step_configs/run_fp_config/template_slice_config/group_size:, 33
retry_on_transient_error:, 28      step_configs/run_fp_config/template_slice_config/group_size:, 33
step_configs/run_explore_config/template_config/timeout (Argument)
    (Argument)                   step_configs/run_fp_config/template_slice_config/pool_size:, 33
timeout:, 28                     step_configs/run_fp_config/template_slice_config/pool_size:, 33
step_configs/run_explore_config/template_slice_config (Argument)
    (Argument)                   pool_size:, 33
template_slice_config:, 28        step_configs/run_train_config (Argument)
step_configs/run_explore_config/template_slice_config/group_size:, 22
    (Argument)                   step_configs/run_train_config/continue_on_failed
group_size:, 28                  (Argument)
step_configs/run_explore_config/template_slice_config/pool_size:, 24
    (Argument)                   step_configs/run_train_config/continue_on_num_success
pool_size:, 28                   (Argument)
step_configs/run_fp_config (Argument) continue_on_num_success:, 24
run_fp_config:, 32                step_configs/run_train_config/continue_on_success_ratio
step_configs/run_fp_config/continue_on_failed (Argument)
    (Argument)                   (Argument)
continue_on_failed:, 33           continue_on_success_ratio:, 24
step_configs/run_fp_config/continue_on_num_success (Argument)
    (Argument)                   step_configs/run_train_config/executor (Argument)
continue_on_num_success:, 33       executor:, 24
step_configs/run_fp_config/continue_on_success_ratio (Argument)
    (Argument)                   step_configs/run_train_config/executor/type
continue_on_success_ratio:, 33     type:, 24
step_configs/run_fp_config/executor (Argument) step_configs/run_train_config/parallelism
executor:, 34                   (Argument)
step_configs/run_fp_config/executor/type (Argument) parallelism:, 24
type:, 34                         step_configs/run_train_config/template_config
step_configs/run_fp_config/parallelism (Argument) (Argument)
parallelism:, 33                  template_config:, 23
step_configs/run_fp_config/template_config (Argument) step_configs/run_train_config/template_config/envs
template_config:, 32               (Argument)
step_configs/run_fp_config/template_config/envs (Argument) envs:, 23
envs:, 33                         step_configs/run_train_config/template_config/image
step_configs/run_fp_config/template_config/image (Argument) (Argument)
image:, 32                        retry_on_transient_error:, 23
step_configs/run_fp_config/template_config/retry_on_transient_error (Argument)
    (Argument)                   step_configs/run_train_config/template_config/timeout
retry_on_transient_error:, 23      timeout:, 23
step_configs/run_train_config/template_config/timeout_as_transient_error:, 32

```

```

(Argument) group_size:, 35
timeout_as_transient_error:, 23 step_configs/select_confs_config/template_slice_config/poo
step_configs/run_train_config/template_slice_config (Argument)
(Argument) pool_size:, 35
template_slice_config:, 23 step_configs:
(Argument) storage_client:
group_size:, 23 bohrium_config/storage_client (Argument),
step_configs/run_train_config/template_slice_config/group_size(Argument), 20
(Argument) student_model_path:
pool_size:, 24 train[dp-dist]/student_model_path (Argument),
step_configs/select_confs_config (Argument) submit_args() (in module dpgen2.entrypoint.args), 98
select_confs_config:, 34 submit_concurrent_learning() (in module dp
step_configs/select_confs_config/continue_on_failed concurrent_learning() (in module dp
(Argument) gen2.entrypoint.submit), 102
continue_on_failed:, 35 successful_step_keys() (in module dp
step_configs/select_confs_config/continue_on_num_success 2.entrypoint.submit), 102
(Argument) system_name:
continue_on_num_success:, 36 task_group/system_name (Argument), 73
step_configs/select_confs_config/continue_on_success_ratio T
(Argument) task_group (Argument)
continue_on_success_ratio:, 36 task_group:, 67, 72
step_configs/select_confs_config/executor (Argument) task_group/atomic_number (Argument)
executor:, 36 atomic_number:, 72
step_configs/select_confs_config/executor/type task_group/command (Argument)
(Argument) command:, 74
type:, 36 task_group/ctrl_range (Argument)
step_configs/select_confs_config/parallelism ctrl_range:, 75
(Argument) task_group/distance_of_ions (Argument)
parallelism:, 36 distance_of_ions:, 72
step_configs/select_confs_config/template_config task_group/fmax (Argument)
(Argument) fmax:, 73
template_config:, 34 task_group/ialgo (Argument)
step_configs/select_confs_config/template_config/envs 73
(Argument) task_group/icode (Argument)
envs:, 35 icode:, 73
step_configs/select_confs_config/template_config/image task_group/max_numb_atoms (Argument)
(Argument) max_numb_atoms:, 75
image:, 34 task_group/max_step (Argument)
step_configs/select_confs_config/template_config/retry_on_transient_error max_step:, 73
(Argument) task_group/max_time (Argument)
retry_on_transient_error:, 35 max_time:, 74
step_configs/select_confs_config/template_config/timeout task_group/name_of_atoms (Argument)
(Argument) name_of_atoms:, 72
timeout:, 34 task_group/numb_of_atoms (Argument)
step_configs/select_confs_config/template_config/timeout_as_transient_error numb_of_atoms:, 72
(Argument) task_group/numb_of_formula (Argument)
timeout_as_transient_error:, 35 numb_of_formula:, 73
step_configs/select_confs_config/template_slice_config task_group/numb_of_lbtest (Argument)
(Argument) numb_of_lbtest:, 73
template_slice_config:, 35 task_group/numb_of_local_optim (Argument)
step_configs/select_confs_config/template_slice_config/group_size numb_of_local_optim:, 74
(Argument) task_group/numb_of_species (Argument)

```

```

    numb_of_species:, 72
task_group/parallel (Argument)
    parallel:, 74
task_group/pick_step (Argument)
    pick_step:, 74
task_group/pick_up (Argument)
    pick_up:, 74
task_group/pop_size (Argument)
    pop_size:, 72
task_group/pressure (Argument)
    pressure:, 73
task_group/pso_ratio (Argument)
    pso_ratio:, 73
task_group/spec_space_group (Argument)
    spec_space_group:, 74
task_group/split (Argument)
    split:, 74
task_group/system_name (Argument)
    system_name:, 73
task_group/type (Argument)
    type:, 67
task_group/volume (Argument)
    volume:, 73
task_group/vsc (Argument)
    vsc:, 74
task_group:
    task_group (Argument), 67, 72
task_group[customized-lmp-template]/conf_idx
    (Argument)
    conf_idx:, 70
task_group[customized-lmp-template]/custom_shell_commands
    (Argument)
    custom_shell_commands:, 70
task_group[customized-lmp-template]/input_extra_files
    (Argument)
    input_extra_files:, 71
task_group[customized-lmp-template]/input_lmp_conf_name
    (Argument)
    input_lmp_conf_name:, 71
task_group[customized-lmp-template]/input_lmp_tmpl_name
    (Argument)
    input_lmp_tmpl_name:, 71
task_group[customized-lmp-template]/input_plm_tmpl_name
    (Argument)
    input_plm_tmpl_name:, 71
task_group[customized-lmp-template]/n_sample
    (Argument)
    n_sample:, 70
task_group[customized-lmp-template]/output_dir_pattern
    (Argument)
    output_dir_pattern:, 71
task_group[customized-lmp-template]/output_lmp_tmpl_name
    (Argument)
    output_lmp_tmpl_name:, 71
task_group[customized-lmp-template]/output_lmp_tmpl_name
    (Argument)
    output_lmp_tmpl_name:, 71
task_group[customized-lmp-template]/output_plm_tmpl_name
    (Argument)
    output_plm_tmpl_name:, 72
task_group[customized-lmp-template]/revisions
    (Argument)
    revisions:, 70
task_group[customized-lmp-template]/traj_freq
    (Argument)
    traj_freq:, 71
task_group[lmp-md]/conf_idx (Argument)
    conf_idx:, 67
task_group[lmp-md]/dt (Argument)
    dt:, 68
task_group[lmp-md]/ens (Argument)
    ens:, 67
task_group[lmp-md]/n_sample (Argument)
    n_sample:, 67
task_group[lmp-md]/neidelay (Argument)
    neidelay:, 68
task_group[lmp-md]/no_pbc (Argument)
    no_pbc:, 68
task_group[lmp-md]/nsteps (Argument)
    nsteps:, 68
task_group[lmp-md]/pka_e (Argument)
    pka_e:, 68
task_group[lmp-md]/press (Argument)
    press:, 67
task_group[lmp-md]/relative_f_epsilon (Argument)
    relative_f_epsilon:, 69
task_group[lmp-md]/relative_v_epsilon (Argument)
    relative_v_epsilon:, 69
task_group[lmp-md]/tau_p (Argument)
    tau_p:, 68
task_group[lmp-md]/tau_t (Argument)
    tau_t:, 68
task_group[lmp-md]/temps (Argument)
    temps:, 67
task_group[lmp-md]/trj_freq (Argument)
    trj_freq:, 68
task_group[lmp-md]/use_clusters (Argument)
    use_clusters:, 68
task_group[lmp-template]/conf_idx (Argument)
    conf_idx:, 69
task_group[lmp-template]/lmp_template_fname
    (Argument)
    lmp_template_fname:, 69
task_group[lmp-template]/n_sample (Argument)
    n_sample:, 69

```

```

task_group[lmp-template]/plm_template_fname
    (Argument)
    plm_template_fname:, 69
task_group[lmp-template]/revisions (Argument)
    revisions:, 70
task_group[lmp-template]/traj_freq (Argument)
    traj_freq:, 70
task_list(dpgen2.exploration.task.task_group.BaseExploratio
    property), 130
task_list(dpgen2.exploration.task.task_group.FooTaskGroup
    property), 131
task_max:
    fp[deepmd]/task_max (Argument), 65
    fp[fpop_abacus]/task_max (Argument), 66
    fp[gaussian]/task_max (Argument), 64
    fp[vasp]/task_max (Argument), 62
tau_p:
    task_group[lmp-md]/tau_p (Argument), 68
tau_t:
    task_group[lmp-md]/tau_t (Argument), 68
teacher_model_path:
    explore[calypso]/config/teacher_model_path
        (Argument), 55
    explore[lmp]/config/teacher_model_path
        (Argument), 48
    fp[deepmd]/run_config/teacher_model_path
        (Argument), 64
template_conf_args() (in module dpgen2.utils.step_config), 176
template_config:
    default_step_config/template_config
        (Argument), 17
    step_configs/cl_step_config/template_config
        (Argument), 39
    step_configs/collect_data_config/template_config
        (Argument), 37
    step_configs/prep_explore_config/template_config
        (Argument), 25
    step_configs/prep_fp_config/template_config
        (Argument), 30
    step_configs/run_explore_config/template_config
        (Argument), 28
    step_configs/run_fp_config/template_config
        (Argument), 33
    step_configs/run_train_config/template_config
        (Argument), 23
    step_configs/select_confs_config/template_config
        (Argument), 35
temp:
    task_group[lmp-md]/temp (Argument), 67
timeout:
    default_step_config/template_config/timeout
        (Argument), 17
    step_configs/cl_step_config/template_config/timeout
        (Argument), 39
    step_configs/collect_data_config/template_config/timeout
        (Argument), 37
    step_configs/prep_explore_config/template_config/timeout
        (Argument), 25
    step_configs/prep_fp_config/template_config/timeout
        (Argument), 30
    step_configs/prep_train_config/template_config/timeout
        (Argument), 21
    step_configs/run_explore_config/template_config/timeout
        (Argument), 28
    step_configs/run_fp_config/template_config/timeout
        (Argument), 32
    step_configs/run_train_config/template_config/timeout
        (Argument), 23
    step_configs/select_confs_config/template_config/timeout
        (Argument), 34
step_configs/run_train_config/template_config/timeout_as_transient_error:
    default_step_config/template_config/timeout_as_transient_error
        (Argument), 18
    step_configs/cl_step_config/template_config/timeout_as_transient_error
        (Argument), 39
    step_configs/collect_data_config/template_config/timeout_as_transient_error
        (Argument), 37
    step_configs/prep_explore_config/template_config/timeout_as_transient_error
        (Argument), 25

```

```

(Argument), 25
step_configs/prep_fp_config/template_config/timeouttransient_error
(Argument), 30
step_configs/prep_train_config/template_config/fdp-distconfig/initinit_data_idx
(Argument), 21
step_configs/run_explore_config/template_config/multitaskconfig/initinit_data_idx
(Argument), 28
step_configs/run_fp_config/template_config/timeoutmultitaskconfig/inittransient_error
(Argument), 32
step_configs/run_train_config/template_config/studentmodelconfig/initmultitask
(Argument), 23
step_configs/select_confs_config/template_config/studentmodelconfig/initscript
(Argument), 35
to_str() (dpgen2.op.md_settings.MDSettings method),
152
train (Argument)
  train:, 43
train/type (Argument)
  type:, 43
train:
  train (Argument), 43
training_args()
  gen2.op.run_dp_train.RunDPTTrain
  method), 162
train[dp-dist]/config (Argument)
  config:, 45
train[dp-dist]/config/finetune_args
  (Argument)
    finetune_args:, 47
train[dp-dist]/config/head (Argument)
  head:, 47
train[dp-dist]/config/impl (Argument)
  impl:, 45
train[dp-dist]/config/init_model_numb_steps
  (Argument)
    init_model_numb_steps:, 46
train[dp-dist]/config/init_model_old_ratio
  (Argument)
    init_model_old_ratio:, 46
train[dp-dist]/config/init_model_policy
  (Argument)
    init_model_policy:, 46
train[dp-dist]/config/init_model_start_lr
  (Argument)
    init_model_start_lr:, 46
train[dp-dist]/config/init_model_start_pref_e
  (Argument)
    init_model_start_pref_e:, 46
train[dp-dist]/config/init_model_start_pref_f
  (Argument)
    init_model_start_pref_f:, 46
train[dp-dist]/config/init_model_start_pref_v
  (Argument)
    init_model_start_pref_v:, 47
train[dp-dist]/config/init_model_with_finetune
  (Argument)
    init_model_with_finetune:, 44
train[dp]/config/multi_init_data_idx
  (Argument)
    multi_init_data_idx:, 45
train[dp]/config/multitask (Argument)
  multitask:, 44
train[dp]/init_models_paths (Argument)
  init_models_paths:, 45
train[dp]/numb_models (Argument)
  numb_models:, 45
train[dp]/template_script (Argument)
  template_script:, 45

```

t

- traj_freq:
 - task_group[customized-lmp-template]/traj_freq (Argument), 71
 - task_group[lmp-template]/traj_freq (Argument), 70
- TrajRender (class in dp-gen2.exploration.render.traj_render), 104
- TrajRenderLammps (class in dp-gen2.exploration.render.traj_render_lammps), 105
- trj_freq:
 - task_group[lmp-md]/trj_freq (Argument), 68
- type:
 - default_step_config/executor/type (Argument), 19
 - explore/type (Argument), 48
 - explore[calypso]/configurations/type (Argument), 59
 - explore[calypso]/convergence/type (Argument), 56
 - explore[lmp]/configurations/type (Argument), 52
 - explore[lmp]/convergence/type (Argument), 49
 - fp/type (Argument), 61
 - step_configs/cl_step_config/executor/type (Argument), 41
 - step_configs/collect_data_config/executor/type (Argument), 39
 - step_configs/prep_explore_config/executor/type (Argument), 27
 - step_configs/prep_fp_config/executor/type (Argument), 31
 - step_configs/prep_train_config/executor/type (Argument), 22
 - step_configs/run_explore_config/executor/type (Argument), 29
 - step_configs/run_fp_config/executor/type (Argument), 34
 - step_configs/run_train_config/executor/type (Argument), 24
 - step_configs/select_confs_config/executor/type (Argument), 36
 - task_group/type (Argument), 67
 - train/type (Argument), 43
- type_map:
 - inputs/type_map (Argument), 41

U

- update_finished_steps() (in module dp-gen2.entrypoint.watch), 102
- update_reuse_step_scheduler() (in module dp-gen2.entrypoint.submit), 102
- upload_python_packages (Argument)
- upload_python_packages():
 - upload_python_packages (Argument), 41
- use_clusters:
 - task_group[lmp-md]/use_clusters (Argument), 68
- username:
 - bohrium_config/username (Argument), 19

V

- valid_data_prefix:
 - inputs/valid_data_prefix (Argument), 42
- valid_data_sys:
 - inputs/valid_data_sys (Argument), 42
- validate_trajs() (dp-gen2.op.select_confs.SelectConfs static method), 165
- variant_conf() (in module dp-gen2.entrypoint.args), 98
- variant_conv() (in module dp-gen2.entrypoint.args), 98
- variant_executor() (in module dp-gen2.utils.step_config), 176
- variant_explore() (in module dp-gen2.entrypoint.args), 98
- variant_fp() (in module dp-gen2.entrypoint.args), 98
- variant_task_group() (in module dp-gen2.exploration.task.make_task_group_from_config), 127
- variant_train() (in module dp-gen2.entrypoint.args), 98
- vaspInputs (class in dp-gen2.fp.vasp_input), 148
- volume:
 - task_group/volume (Argument), 73
- vsc:
 - task_group/vsc (Argument), 74

W

- watch() (in module dp-gen2.entrypoint.watch), 102
- workflow_concurrent_learning() (in module dp-gen2.entrypoint.submit), 102
- workflow_config_from_dict() (in module dp-gen2.utils.dflow_config), 174
- write_data_to_input_script() (dp-gen2.op.run_dp_train.RunDPTTrain static method), 162
- write_model_devi_out() (in module dp-gen2.op.run_caly_model_devi), 160
- write_other_to_input_script() (dp-gen2.op.run_dp_train.RunDPTTrain static method), 162