DP-GEN

DeepModeling

Sep 04, 2022

OVERVIEW

CHAPTER

ONE

OVERVIEW

1.1 About DP-GEN

DP-GEN (Deep Generator) is a software written in Python, delicately designed to generate a deep learning based model of interatomic potential energy and force field. DP-GEN is dependent on DeepMD-kit. With highly scalable interface with common softwares for molecular simulation, DP-GEN is capable to automatically prepare scripts and maintain job queues on HPC machines (High Performance Cluster) and analyze results.

If you use this software in any publication, please cite:

Yuzhi Zhang, Haidi Wang, Weijie Chen, Jinzhe Zeng, Linfeng Zhang, Han Wang, and Weinan E, DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models, Computer Physics Communications, 2020, 107206.

1.1.1 Highlighted features

- **Accurate and efficient**: DP-GEN is capable to sample more than tens of million structures and select only a few for first principles calculation. DP-GEN will finally obtain a uniformly accurate model.
- **User-friendly and automatic**: Users may install and run DP-GEN easily. Once succusefully running, DP-GEN can dispatch and handle all jobs on HPCs, and thus there's no need for any personal effort.
- **Highly scalable**: With modularized code structures, users and developers can easily extend DP-GEN for their most relevant needs. DP-GEN currently supports for HPC systems (Slurm, PBS, LSF and cloud machines), Deep Potential interface with DeePMD-kit, MD interface with LAMMPS, Gromacs and *ab-initio* calculation interface with VASP, PWSCF, CP2K, SIESTA and Gaussian, Abacus, PWMAT, etc . We're sincerely welcome and embraced to users' contributions, with more possibilities and cases to use DP-GEN.

1.2 Download and install

Please follow our GitHub webpage to download the latest released version and development version. One can download the source code of dpgen by

```
git clone https://github.com/deepmodeling/dpgen.git
```

DP-GEN offers multiple installation methods. It is recommend using easily methods like:

- offline packages: find them in releases,
- pip: use `pip install dpgen`, see dpgen-PyPI

- conda: use `conda install -c deepmodeling dpgen`, see dpgen-conda

To test if the installation is successful, you may execute

```
dpgen -h
```

or just

```
dpgen
```

1.3 Use DP-GEN

A quick-start on using DPGEN can be found here. You can follow the Handson tutorial, it is friendly to new users.

1.4 Case Studies

- Practical-Guidelines-for-DP

Before starting a new Deep Potential (DP) project, we suggest people (especially those who are newbies) read the following context first to get some insights into what tools we can use, what kinds of risks and difficulties we may meet, and how we can advance a new DP project smoothly.

- Convergence-Test

to ensure the data quality, the reliability of the final model, as well as the feasibility of the project, a convergence test should be done first.

- Gas-phase

In this tutorial, we will take the simulation of methane combustion as an example and introduce the procedure of DP-based MD simulation.

- Mg-Y_alloy

We will briefly analyze the candidate configurational space of a metallic system by taking Mg-based Mg-Y binary alloy as an example. The task is divided into steps during the DP-GEN process.

- Transfer-learning

This tutorial will introduce how to implement potential energy surface (PES) transfer-learning by using the DP-GEN software. In DP-GEN (version > 0.8.0), the "simplify" module is designed for this purpose.

1.5 License

The project dpgen is licensed under GNU LGPLv3.0

CHAPTER

TWO

COMMAND LINE INTERFACE

dpgen is a convenient script that uses DeepGenerator to prepare initial data, drive DeepMDkit and analyze results. This script works based on several sub-commands with their own options. To see the options for the sub-commands, type "dpgen sub-command -h".

```
usage: dpgen [-h]
             {init_surf,init_bulk,auto_gen_param,init_reaction,run,run/report,collect,
↪simplify,autotest,db}
             ...
```

2.1 Sub-commands:

2.1.1 init_surf

Generating initial data for surface systems.

```
dpgen init_surf [-h] PARAM [MACHINE]
```

Positional Arguments

PARAM	parameter file, json/yaml format
MACHINE	machine file, json/yaml format

2.1.2 init_bulk

Generating initial data for bulk systems.

```
dpgen init_bulk [-h] PARAM [MACHINE]
```

Positional Arguments

PARAM parameter file, json/yaml format

MACHINE machine file, json/yaml format

2.1.3 auto_gen_param

auto gen param.json

```
dpgen auto_gen_param [-h] PARAM
```

Positional Arguments

PARAM parameter file, json/yaml format

2.1.4 init_reaction

Generating initial data for reactive systems.

```
dpgen init_reaction [-h] PARAM [MACHINE]
```

Positional Arguments

PARAM parameter file, json/yaml format

MACHINE machine file, json/yaml format

2.1.5 run

Main process of Deep Potential Generator.

```
dpgen run [-h] [-d] PARAM MACHINE
```

Positional Arguments

PARAM parameter file, json/yaml format

MACHINE machine file, json/yaml format

Named Arguments

-d, --debug log debug info

Default: False

2.1.6 run/report

Report the systems and the thermodynamic conditions of the labeled frames.

```
dpgen run/report [-h] [-s] [-i] [-t] [-p PARAM] [-v] JOB_DIR
```

Positional Arguments

JOB_DIR the directory of the DP-GEN job,

Named Arguments

-s, --stat-sys count the labeled frames for each system

Default: False

-i, --stat-iter print the iteration candidate,failed,accurate count and fp calculation,success and fail count

Default: False

-t, --stat-time print the iteration time, warning!! assume model_devi parallel cores == 1

Default: False

-p, --param the json file provides DP-GEN paramters, should be located in JOB_DIR

Default: "param.json"

-v, --verbose being loud

Default: False

2.1.7 collect

Collect data.

```
dpgen collect [-h] [-p PARAMETER] [-v] [-m] [-s] JOB_DIR OUTPUT
```

Positional Arguments

JOB_DIR the directory of the DP-GEN job

OUTPUT the output directory of data

Named Arguments

-p, --parameter	the json file provides DP-GEN paramters, should be located in JOB_DIR Default: "param.json"
-v, --verbose	print number of data in each system Default: False
-m, --merge	merge the systems with the same chemical formula Default: False
-s, --shuffle	shuffle the data systems Default: False

2.1.8 simplify

Simplify data.

```
dpgen simplify [-h] [-d] PARAM MACHINE
```

Positional Arguments

PARAM	parameter file, json/yaml format
MACHINE	machine file, json/yaml format

Named Arguments

-d, --debug	log debug info Default: False

2.1.9 autotest

Auto-test for Deep Potential.

```
dpgen autotest [-h] [-d] TASK PARAM [MACHINE]
```

Positional Arguments

TASK	task can be make, run or post
PARAM	parameter file, json/yaml format
MACHINE	machine file, json/yaml format

Named Arguments

-d, --debug log debug info

Default: False

2.1.10 db

Collecting data from DP-GEN.

```
dpgen db [-h] PARAM
```

Positional Arguments

PARAM parameter file, json format

CHAPTER

THREE

CODE STRUCTURE

Let's look at the home page of DP-GEN. https://github.com/deepmodeling/dpgen

```
├── build
├── CITATION.cff
├── conda
├── dist
├── doc
├── dpgen
├── dpgen.egg-info
├── examples
├── LICENSE
├── README.md
├── requirements.txt
├── setup.py
└── tests
```

- `tests` : unittest tools for developers.
- `examples`: templates for PARAM and MACHINE files for different software, versions and tasks. For details of the parameters in PARAM, you can refer to `TASK parameters` chapters in this document. If you are confused about how to set up a JSON file, you can also use dpgui

Most of the code related to DP-GEN functions is in the `dpgen` directory. Open the `dpgen` directory, and we can see

```
├── arginfo.py
├── auto_test
├── collect
├── data
├── database
├── _date.py
├── dispatcher
├── generator
├── __init__.py
├── main.py
├── __pycache__
├── remote
├── simplify
├── tools
├── util.py
└── _version.py
```

- `auto_test` corresponds to `dpgen autotest`, for undertaking materials property analysis.

- `collect` corresponds to `dpgen collect`.
- `data` corresponds to `dpgen init_bulk`, `dpgen init_surf` and `dpgen init_reaction`, for preparing initial data of bulk and surf systems.
- `database` is the source code for collecting data generated by DP-GEN and interface with database.
- `simplify` corresponds to `dpgen simplify`.
- `remote` and `dispatcher` : source code for automatically submiting scripts,maintaining job queues and collecting results. **Notice this part hase been integrated into dpdispatcher** `generator` is the core part of DP-GEN. It's for main process of deep generator. Let's open this folder.

```
├── arginfo.py
├── ch4
├── __init__.py
├── lib
└── run.py
```

`run.py` is the core of DP-GEN, corresponding to `dpgen run`. We can find `make_train`, `run_train`, ... `post_fp`, and other steps related functions here.

CHAPTER

FOUR

RUN

4.1 Overview of the Run process

The run process contains a series of successive iterations, undertaken in order such as heating the system to certain temperatures. Each iteration is composed of three steps: exploration, labeling, and training. Accordingly, there are three sub-folders: 00.train, 01.model_devi, and 02.fp in each iteration.

00.train: DP-GEN will train several (default 4) models based on initial and generated data. The only difference between these models is the random seed for neural network initialization.

01.model_devi : represent for model-deviation. DP-GEN will use models obtained from 00.train to run Molecular Dynamics(default LAMMPS). Larger deviation for structure properties (default is the force of atoms) means less accuracy of the models. Using this criterion, a few structures will be selected and put into the next stage 02.fp for more accurate calculation based on First Principles.

02.fp : Selected structures will be calculated by first-principles methods(default VASP). DP-GEN will obtain some new data and put them together with initial data and data generated in previous iterations. After that, new training will be set up and DP-GEN will enter the next iteration!

In the run process of the DP-GEN, we need to specify the basic information about the system, the initial data, and details of the training, exploration, and labeling tasks. In addition, we need to specify the software, machine environment, and computing resource and enable the process of job generation, submission, query, and collection automatically. We can perform the run process as we expect by specifying the keywords in param.json and machine.json, and they will be introduced in detail in the following sections.

Here, we give a general description of the run process. We can execute the run process of DP-GEN easily by:

```
dpgen run param.json machine.json
```

The following files or folders will be created and upgraded by codes

- iter.00000x contains the main results that DP-GEN generates in the first iteration.
- record.dpgen records the current stage of the run process.
- dpgen.log includes time and iteration information.

When the first iteration is completed, the folder structure of iter.000000 is like this:

```
$ ls iter.000000
00.train 01.model_devi 02.fp
```

In folder iter.000000/ 00.train:

- Folder 00x contains the input and output files of the DeePMD-kit, in which a model is trained.

- graph.00x.pb is the model DeePMD-kit generates. The only difference between these models is the random seed for neural network initialization.

In folder iter.000000/ 01.model_devi

- Folder confs contains the initial configurations for LAMMPS MD converted from POSCAR you set in "sys_configs" of param.json.
- Folder task.000.00000x contains the input and output files of the LAMMPS. In folder task.000.00000x, file model_devi.out records the model deviation of concerned labels, energy and force in MD. It serves as the criterion for selecting which structures and doing first-principle calculations.

In folder iter.000000/ 02.fp

- candidate.shuffle.000.out records which structures will be selected from last step 01.model_devi. There are always far more candidates than the maximum you expect to calculate at one time. In this condition, DP-GEN will randomly choose up to "`fp_task_max`" structures and form the folder task.*.
- rest_accurate.shuffle.000.out records the other structures where our model is accurate ("max_devi_f" is less than "`model_devi_f_trust_lo`", no need to calculate any more),
- rest_failed.shuffled.000.out records the other structures where our model is too inaccurate (larger than "`model_devi_f_trust_hi`", there may be some error).
- data.000: After first-principle calculations, DP-GEN will collect these data and change them into the format DeePMD-kit needs. In the next iteration's 00.train, these data will be trained together as well as the initial data.

DP-GEN identifies the stage of the run process by a record file, record.dpgen, which will be created and upgraded by codes. Each line contains two numbers: the first is the index of iteration, and the second, ranging from 0 to 9, records which stage in each iteration is currently running.

Index of iterations	Stage in eachiteration	Process
0	0	make_train
0	1	run_train
0	2	post_train
0	3	make_model_devi
0	4	run_model_devi
0	5	post_model_devi
0	6	make_fp
0	7	run_fp
0	8	post_fp

0,1,2 correspond to make_train, run_train, post_train. DP-GEN will write scripts in make_train, run the task by specific machine in run_train and collect result in post_train. The records for model_devi and fp stage follow similar rules.

If the process of DP-GEN stops for some reasons, DP-GEN will automatically recover the main process by record.dpgen. You may also change it manually for your purpose, such as removing the last iterations and recovering from one checkpoint.

4.2 Example-of-param.json

We have provided different examples of param.json in dpgen/examples/run/. In this section, we give a description of the param.json, taking dpgen/examples/run/dp2.x-lammps-vasp/param_CH4_deepmd-kit-2.0.1.json as an example. This is a param.json for a gas-phase methane molecule. Here, DeePMD-kit (v2.x), LAMMPS and VASP codes are used for training, exploration and labeling respectively.

4.2.1 basics

The basics related keys in param.json are given as follows

```
"type_map": [
  "H",
  "C"
],
"mass_map": [
  1,
  12
],
```

The basics related keys specify the basic information about the system. "type_map" gives the atom types, i.e. "H" and "C". "mass_map" gives the standard atom weights, i.e. "1" and "12".

4.2.2 data

The data related keys in param.json are given as follows

```
"init_data_prefix": "....../init/",
"init_data_sys": [
  "CH4.POSCAR.01x01x01/02.md/sys-0004-0001/deepmd"
],

"sys_configs_prefix": "....../init/",
"sys_configs": [
  [
    "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale*/00000*/POSCAR"
  ],
  [
    "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale*/00001*/POSCAR"
  ]
],
```

The data related keys specify the init data for training initial DP models and structures used for model_devi calculations. "init_data_prefix" and "init_data_sys" specify the location of the init data. "sys_configs_prefix" and "sys_configs" specify the location of the structures.

Here, the init data is provided at "..... /init/CH4.POSCAR.01x01x01/02.md/sys-0004-0001/deepmd". These structures are divided into two groups and provided at ".....init/CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale*/00000*/POSCAR" and ".....init/CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale*/00001*/POSCAR".

4.2.3 training

The training related keys in param.json are given as follows

```
"numb_models": 4,
"default_training_param": {
},
```

The training related keys specify the details of training tasks. "numb_models" specifies the number of models to be trained. "default_training_param" specifies the training parameters for `deepmd-kit`.

Here, 4 DP models will be trained in `00.train`. A detailed explanation of training parameters can be found in DeePMD-kit's documentation (https://docs.deepmodeling.com/projects/deepmd/en/master/).

4.2.4 exploration

The exploration related keys in param.json are given as follows

```
"model_devi_dt": 0.002,
"model_devi_skip": 0,
"model_devi_f_trust_lo": 0.05,
"model_devi_f_trust_hi": 0.15,
"model_devi_clean_traj": true,
"model_devi_jobs": [
  {
    "sys_idx": [
      0
    ],
    "temps": [
      100
    ],
    "press": [
      1.0
    ],
    "trj_freq": 10,
    "nsteps": 300,
    "ensemble": "nvt",
    "_idx": "00"
  },
  {
    "sys_idx": [
      1
    ],
    "temps": [
      100
    ],
    "press": [
      1.0
    ],
    "trj_freq": 10,
    "nsteps": 3000,
    "ensemble": "nvt",
    "_idx": "01"
```

(continues on next page)

(continued from previous page)

```
    }
  ],
```

The exploration related keys specify the details of exploration tasks. "model_devi_dt" specifies timestep for MD simulation. "model_devi_skip" specifies the number of structures skipped for saving in each MD. "model_devi_f_trust_lo" and "model_devi_f_trust_hi" specify the lower and upper bound of model_devi of forces for the selection. "model_devi_clean_traj" specifies whether to clean traj folders in MD. If type of model_devi_clean_traj is boolean type then it denote whether to clean traj folders in MD since they are too large.In "model_devi_jobs", "sys_idx" specifies the group of structures used for model_devi calculations, "temps" specifies the temperature (K) in MD, "press" specifies the pressure (Bar) in MD, "trj_freq" specifies the frequency of trajectory saved in MD, "nsteps" specifies the running steps of MD, "ensemble" specifies the ensemble used in MD, and "_idx" specifies the index of iteration.

Here, MD simulations are performed at the temperature of 100 K and the pressure of 1.0 Bar with an integrator time of 2 fs under the nvt ensemble. Two iterations are set in "model_devi_jobs". MD simulations are run for 300 and 3000 time steps with the first and second groups of structures in "sys_configs" in 00 and 01 iterations. We choose to save all structures generated in MD simulations and have set `"trj_freq"` as 10, so 30 and 300 structures are saved in 00 and 01 iterations. If the "max_devi_f" of saved structure falls between 0.05 and 0.15, DP-GEN will treat the structure as a candidate. We choose to clean traj folders in MD since they are too large. If you want to save the most recent n iterations of traj folders, you can set "model_devi_clean_traj" to be an integer.

4.2.5 labeling

The labeling related keys in param.json are given as follows

```
  "fp_style": "vasp",
  "shuffle_poscar": false,
  "fp_task_max": 20,
  "fp_task_min": 1,
  "fp_pp_path": "....../methane/",
  "fp_pp_files": [
    "POTCAR"
  ],
  "fp_incar": "....../INCAR_methane"
```

The labeling related keys specify the details of labeling tasks. "fp_style" specifies software for First Principles. "fp_task_max" and "fp_task_min" specify the minimum and maximum of structures to be calculated in `02.fp` of each iteration. "fp_pp_path" and "fp_pp_files" specify the location of the psuedo-potential file to be used for 02.fp. "fp_incar" specifies input file for VASP. INCAR must specify KSPACING and KGAMMA.

Here, a minimum of 1 and a maximum of 20 structures will be labeled using the VASP code with the INCAR provided at "....../INCAR_methane" and POTCAR provided at "....../methane/POTCAR" in each iteration. Note that the order of elements in POTCAR should correspond to the order in `type_map`.

All the keys of the DP-GEN are explained in detail in the section Parameters.

4.3 Example of machine.json

4.3.1 DPDispatcher Update Note

DPDispatcher has updated and the api of machine.json is changed. DP-GEN will use the new DPDispatcher if the value of key "api_version" in machine.json is equal to or large than 1.0. And for now, DPDispatcher is maintained on a separate repo (https://github.com/deepmodeling/dpdispatcher). Please check the documents (https://deepmd.readthedocs.io/projects/dpdispatcher/en/latest/) for more information about the new DPDispatcher.

DP-GEN will use the old DPDispatcher if the key "api_version" is not specified in machine.json or the "api_version" is smaller than 1.0. This gurantees that the old machine.json still works.

4.3.2 New DPDispatcher

Each iteration in the run process of DP-GEN is composed of three steps: exploration, labeling, and training. Accordingly, machine.json is composed of three parts: train, model_devi, and fp. Each part is a list of dicts. Each dict can be considered as an independent environment for calculation.

In this section, we will show you how to perform train task at a local workstation, model_devi task at a local Slurm cluster, and fp task at a remote PBS cluster using the new DPDispatcher. For each task, three types of keys are needed:

- Command: provides the command used to execute each step.
- Machine: specifies the machine environment (local workstation, local or remote cluster, or cloud server).
- Resources: specify the number of groups, nodes, CPU, and GPU; enable the virtual environment.

Performing train task at a local workstation

In this example, we perform the `train` task on a local workstation.

```
"train":
    {
      "command": "dp",
      "machine": {
        "batch_type": "Shell",
        "context_type": "local",
        "local_root": "./",
        "remote_root": "/home/user1234/work_path"
      },
      "resources": {
        "number_node": 1,
        "cpu_per_node": 4,
        "gpu_per_node": 1,
        "group_size": 1,
        "source_list": ["/home/user1234/deepmd.env"]
      }
    },
```

The "command" for the train task in the DeePMD-kit is "dp".

In machine parameters, "batch_type" specifies the type of job scheduling system. If there is no job scheduling system, we can use the "Shell" to perform the task. "context_type" specifies the method of data transfer, and "local" means copying and moving data via local file storage systems (e.g. cp, mv, etc.). In DP-GEN, the paths of all tasks are

automatically located and set by the software, and therefore "local_root" is always set to "./". The input file for each task will be sent to the "remote_root" and the task will be performed there, so we need to make sure that the path exists.

In the resources parameter, "number_node", "cpu_per_node", and "gpu_per_node" specify the number of nodes, the number of CPUs, and the number of GPUs required for a task respectively. "group_size", which needs to be highlighted, specifies how many tasks will be packed into a group. In the training tasks, we need to train 4 models. If we only have one GPU, we can set the "group_size" to 4. If "group_size" is set to 1, 4 models will be trained on one GPU at the same time, as there is no job scheduling system. Finally, the environment variables can be activated by "source_list". In this example, "source /home/user1234/deepmd.env" is executed before "dp" to load the environment variables necessary to perform the training task.

Perform model_devi task at a local Slurm cluster

In this example, we perform the model_devi task at a local Slurm workstation.

```
"model_devi":
    {
      "command": "lmp",
      "machine": {
       "context_type": "local",
        "batch_type": "Slurm",
        "local_root": "./",
        "remote_root": "/home/user1234/work_path"
      },
      "resources": {
        "number_node": 1,
        "cpu_per_node": 4,
        "gpu_per_node": 1,
        "queue_name": "QueueGPU",
        "custom_flags" : ["#SBATCH --mem=32G"],
        "group_size": 10,
        "source_list": ["/home/user1234/lammps.env"]
      }
    }
```

The "command" for the model_devi task in the LAMMPS is "lmp".

In the machine parameter, we specify the type of job scheduling system by changing the "batch_type" to "Slurm".

In the resources parameter, we specify the name of the queue to which the task is submitted by adding "queue_name". We can add additional lines to the calculation script via the "custom_flags". In the model_devi steps, there are frequently many short tasks, so we usually pack multiple tasks (e.g. 10) into a group for submission. Other parameters are similar to that of the local workstation.

Perform fp task in a remote PBS cluster

In this example, we perform the fp task at a remote PBS cluster that can be accessed via SSH.

```
"fp":
    {
      "command": "mpirun -n 32 vasp_std",
      "machine": {
       "context_type": "SSHContext",
        "batch_type": "PBS",
```

(continues on next page)

(continued from previous page)

```
        "local_root": "./",
        "remote_root": "/home/user1234/work_path",
        "remote_profile": {
          "hostname": "39.xxx.xx.xx",
          "username": "user1234"
         }
      },
      "resources": {
        "number_node": 1,
        "cpu_per_node": 32,
        "gpu_per_node": 0,
        "queue_name": "QueueCPU",
        "group_size": 5,
        "source_list": ["/home/user1234/vasp.env"]
      }
    }
```

VASP code is used for fp task and mpi is used for parallel computing, so "mpirun -n 32" is added to specify the number of parallel threads.

In the machine parameter, "context_type" is modified to "SSHContext" and "batch_type" is modified to "PBS". It is worth noting that "remote_root" should be set to an accessible path on the remote PBS cluster. "remote_profile" is added to specify the information used to connect the remote cluster, including hostname, username, port, etc.

In the resources parameter, we set "gpu_per_node" to 0 since it is cost-effective to use the CPU for VASP calculations.

Explicit descriptions of keys in machine.json will be given in the following section.

4.4 dpgen run param parameters

run_jdata:

type: dict
argument path: run_jdata

param.json file

type_map:

type: list
argument path: run_jdata/type_map

Atom types.

mass_map:

type: list | str, optional, default: auto
argument path: run_jdata/mass_map

Standard atomic weights (default: "auto"). if one want to use isotopes, or non-standard element names, chemical symbols, or atomic number in the type_map list, please customize the mass_map list instead of using "auto". Tips: at present the default value will not be applied automatically, so you need to set "mass_map" manually in param.json.

use_ele_temp:

type: int, optional, default: 0

argument path: run_jdata/use_ele_temp

Currently only support fp_style vasp.
- 0: no electron temperature.
- 1: eletron temperature as frame parameter.
- 2: electron temperature as atom parameter.

init_data_prefix:

type: str, optional

argument path: run_jdata/init_data_prefix

Prefix of initial data directories.

init_data_sys:

type: list

argument path: run_jdata/init_data_sys

Directories of initial data. You may use either absolute or relative path here. Systems will be detected recursively in the directories.

sys_format:

type: str, optional, default: vasp/poscar

argument path: run_jdata/sys_format

Format of initial data.

init_batch_size:

type: list | str, optional

argument path: run_jdata/init_batch_size

Each number is the batch_size of corresponding system for training in init_data_sys. One recommended rule for setting the sys_batch_size and init_batch_size is that batch_size mutiply number of atoms ot the stucture should be larger than 32. If set to auto, batch size will be 32 divided by number of atoms.

sys_configs_prefix:

type: str, optional

argument path: run_jdata/sys_configs_prefix

Prefix of sys_configs.

sys_configs:

type: list

argument path: run_jdata/sys_configs

Containing directories of structures to be explored in iterations.Wildcard characters are supported here.

sys_batch_size:

type: list, optional

argument path: run_jdata/sys_batch_size

Each number is the batch_size for training of corresponding system in sys_configs. If set to auto, batch size will be 32 divided by number of atoms.

numb_models:

type: int

argument path: run_jdata/numb_models

Number of models to be trained in 00.train. 4 is recommend.

training_iter0_model_path:

type: list, optional

argument path: run_jdata/training_iter0_model_path

The model used to init the first iter training. Number of element should be equal to numb_models.

training_init_model:

type: bool, optional

argument path: run_jdata/training_init_model

Iteration > 0, the model parameters will be initilized from the model trained at the previous iteration. Iteration == 0, the model parameters will be initialized from training_iter0_model_path.

default_training_param:

type: dict

argument path: run_jdata/default_training_param

Training parameters for deepmd-kit in 00.train. You can find instructions from here: (https://github.com/deepmodeling/deepmd-kit).

dp_compress:

type: bool, optional, default: False

argument path: run_jdata/dp_compress

Use dp compress to compress the model.

training_reuse_iter:

type: int | NoneType, optional

argument path: run_jdata/training_reuse_iter

The minimal index of iteration that continues training models from old models of last iteration.

training_reuse_old_ratio:

type: NoneType | float, optional

argument path: run_jdata/training_reuse_old_ratio

The probability proportion of old data during training. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_numb_steps:

type: int | NoneType, optional, default: 400000, alias: *training_reuse_stop_batch*

argument path: run_jdata/training_reuse_numb_steps

Number of training batch. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_start_lr:

type: NoneType | float, optional, default: 0.0001

argument path: run_jdata/training_reuse_start_lr

The learning rate the start of the training. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_start_pref_e:

type: int | NoneType | float, optional, default: 0.1

argument path: run_jdata/training_reuse_start_pref_e

The prefactor of energy loss at the start of the training. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_start_pref_f:

type: int | NoneType | float, optional, default: 100

argument path: run_jdata/training_reuse_start_pref_f

The prefactor of force loss at the start of the training. This option is only adopted when continuing training models from old models. This option will override default parameters.

model_devi_activation_func:

type: list | NoneType, optional

argument path: run_jdata/model_devi_activation_func

The activation function in the model. The shape of list should be (N_models, 2), where 2 represents the embedding and fitting network. This option will override default parameters.

fp_task_max:

type: int

argument path: run_jdata/fp_task_max

Maximum of structures to be calculated in 02.fp of each iteration.

fp_task_min:

type: int

argument path: run_jdata/fp_task_min

Minimum of structures to be calculated in 02.fp of each iteration.

fp_accurate_threshold:

type: float, optional

argument path: run_jdata/fp_accurate_threshold

If the accurate ratio is larger than this number, no fp calculation will be performed, i.e. fp_task_max = 0.

fp_accurate_soft_threshold:

type: float, optional

argument path: run_jdata/fp_accurate_soft_threshold

If the accurate ratio is between this number and fp_accurate_threshold, the fp_task_max linearly decays to zero.

fp_cluster_vacuum:

type: float, optional

argument path: run_jdata/fp_cluster_vacuum

If the vacuum size is smaller than this value, this cluster will not be choosen for labeling.

detailed_report_make_fp:

type: bool, optional, default: True

argument path: `run_jdata/detailed_report_make_fp`

If set to true, detailed report will be generated for each iteration.

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

model_devi_engine:

type: `str` (flag key), default: `lammps`
argument path: `run_jdata/model_devi_engine`
possible choices: *lammps*, *amber*

Engine for the model deviation task.

When model_devi_engine is set to `lammps`:

LAMMPS

model_devi_jobs:

type: `list`
argument path: `run_jdata[model_devi_engine=lammps]/model_devi_jobs`

Settings for exploration in 01.model_devi. Each dict in the list corresponds to one iteration. The index of model_devi_jobs exactly accord with index of iterations

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

sys_idx:

type: `list`
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_jobs/sys_idx`

Systems to be selected as the initial structure of MD and be explored. The index corresponds exactly to the sys_configs.

temps:

type: `list`
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_jobs/temps`

Temperature (K) in MD.

press:

type: `list`, optional
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_jobs/press`

Pressure (Bar) in MD. Required when ensemble is npt.

trj_freq:

type: `int`
argument path: `run_jdata[model_devi_engine=lammps]/model_devi_jobs/trj_freq`

Frequecy of trajectory saved in MD.

nsteps:

type: `int`
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_jobs/nsteps`

Running steps of MD.

ensemble:

type: str

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/ensemble

Determining which ensemble used in MD, options include “npt” and “nvt”.

neidelay:

type: int, optional

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/neidelay

delay building until this many steps since last build.

taut:

type: float, optional

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/taut

Coupling time of thermostat (ps).

taup:

type: float, optional

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/taup

Coupling time of barostat (ps).

model_devi_f_trust_lo:

type: dict | float, optional

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/model_devi_f_trust_lo

Lower bound of forces for the selection. If dict, should be set for each index in sys_idx, respectively.

model_devi_f_trust_hi:

type: dict | float, optional

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/model_devi_f_trust_hi

Upper bound of forces for the selection. If dict, should be set for each index in sys_idx, respectively.

model_devi_v_trust_lo:

type: dict | float, optional

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/model_devi_v_trust_lo

Lower bound of virial for the selection. If dict, should be set for each index in sys_idx, respectively. Should be used with DeePMD-kit v2.x.

model_devi_v_trust_hi:

type: dict | float, optional

argument path: run_jdata[model_devi_engine=lammps]/model_devi_jobs/model_devi_v_trust_hi

Upper bound of virial for the selection. If dict, should be set for each index in sys_idx, respectively. Should be used with DeePMD-kit v2.x.

model_devi_dt:

type: float
argument path: run_jdata[model_devi_engine=lammps]/model_devi_dt

Timestep for MD. 0.002 is recommend.

model_devi_skip:

type: int
argument path: run_jdata[model_devi_engine=lammps]/model_devi_skip

Number of structures skipped for fp in each MD.

model_devi_f_trust_lo:

type: list | dict | float
argument path:
run_jdata[model_devi_engine=lammps]/model_devi_f_trust_lo

Lower bound of forces for the selection. If list or dict, should be set for each index in sys_configs, respectively.

model_devi_f_trust_hi:

type: list | dict | float
argument path:
run_jdata[model_devi_engine=lammps]/model_devi_f_trust_hi

Upper bound of forces for the selection. If list or dict, should be set for each index in sys_configs, respectively.

model_devi_v_trust_lo:

type: list | dict | float, optional, default: 10000000000.0
argument path:
run_jdata[model_devi_engine=lammps]/model_devi_v_trust_lo

Lower bound of virial for the selection. If list or dict, should be set for each index in sys_configs, respectively. Should be used with DeePMD-kit v2.x.

model_devi_v_trust_hi:

type: list | dict | float, optional, default: 10000000000.0
argument path:
run_jdata[model_devi_engine=lammps]/model_devi_v_trust_hi

Upper bound of virial for the selection. If list or dict, should be set for each index in sys_configs, respectively. Should be used with DeePMD-kit v2.x.

model_devi_adapt_trust_lo:

type: bool, optional
argument path:
run_jdata[model_devi_engine=lammps]/model_devi_adapt_trust_lo

Adaptively determines the lower trust levels of force and virial. This option should be used together with model_devi_numb_candi_f, model_devi_numb_candi_v and optionally with model_devi_perc_candi_f and model_devi_perc_candi_v. dpgen will make two sets:

- 1. From the frames with force model deviation lower than model_devi_f_trust_hi, select max(model_devi_numb_candi_f, model_devi_perc_candi_f*n_frames) frames with largest force model deviation.
- 2. From the frames with virial model deviation lower than model_devi_v_trust_hi, select max(model_devi_numb_candi_v, model_devi_perc_candi_v*n_frames) frames with largest virial model deviation.

The union of the two sets is made as candidate dataset.

model_devi_numb_candi_f:

type: `int`, optional
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_numb_candi_f`

See model_devi_adapt_trust_lo.

model_devi_numb_candi_v:

type: `int`, optional
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_numb_candi_v`

See model_devi_adapt_trust_lo.

model_devi_perc_candi_f:

type: `float`, optional
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_perc_candi_f`

See model_devi_adapt_trust_lo.

model_devi_perc_candi_v:

type: `float`, optional
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_perc_candi_v`

See model_devi_adapt_trust_lo.

model_devi_f_avg_relative:

type: `bool`, optional
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_f_avg_relative`

Normalized the force model deviations by the RMS force magnitude along the trajectory. This key should not be used with use_relative.

model_devi_clean_traj:

type: `int` | `bool`, optional, default: `True`
argument path:
`run_jdata[model_devi_engine=lammps]/model_devi_clean_traj`

If type of model_devi_clean_traj is bool type then it denote whether to clean traj folders in MD since they are too large. If it is Int type, then the most recent n iterations of traj folders will be retained, others will be removed.

model_devi_merge_traj:

type: bool, optional, default: False

argument path: run_jdata[model_devi_engine=lammps]/model_devi_merge_traj

If model_devi_merge_traj is set as True, only all.lammpstrj will be generated, instead of lots of small traj files.

model_devi_nopbc:

type: bool, optional, default: False

argument path: run_jdata[model_devi_engine=lammps]/model_devi_nopbc

Assume open boundary condition in MD simulations.

shuffle_poscar:

type: bool, optional, default: False

argument path: run_jdata[model_devi_engine=lammps]/shuffle_poscar

Shuffle atoms of each frame before running simulations. The purpose is to sample the element occupation of alloys.

use_relative:

type: bool, optional, default: False

argument path: run_jdata[model_devi_engine=lammps]/use_relative

Calculate relative force model deviation.

epsilon:

type: float, optional

argument path: run_jdata[model_devi_engine=lammps]/epsilon

The level parameter for computing the relative force model deviation.

use_relative_v:

type: bool, optional, default: False

argument path: run_jdata[model_devi_engine=lammps]/use_relative_v

Calculate relative virial model deviation.

epsilon_v:

type: float, optional

argument path: run_jdata[model_devi_engine=lammps]/epsilon_v

The level parameter for computing the relative virial model deviation.

When model_devi_engine is set to amber:

Amber DPRc engine. The command argument in the machine file should be path to sander.

model_devi_jobs:

type: list

argument path: run_jdata[model_devi_engine=amber]/model_devi_jobs

List of dicts. The list including the dict for information of each cycle.

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

sys_idx:

type: list

argument path: run_jdata[model_devi_engine=amber]/model_devi_jobs/sys_idx

List of ints. List of systems to run.

trj_freq:

type: int

argument path: run_jdata[model_devi_engine=amber]/model_devi_jobs/trj_freq

Frequency to dump trajectory.

low_level:

type: str

argument path: run_jdata[model_devi_engine=amber]/low_level

Low level method. The value will be filled into mdin file as @qm_theory@.

cutoff:

type: float

argument path: run_jdata[model_devi_engine=amber]/cutoff

Cutoff radius for the DPRc model.

parm7_prefix:

type: str, optional

argument path: run_jdata[model_devi_engine=amber]/parm7_prefix

The path prefix to AMBER PARM7 files.

parm7:

type: list

argument path: run_jdata[model_devi_engine=amber]/parm7

List of paths to AMBER PARM7 files. Each file maps to a system.

mdin_prefix:

type: str, optional

argument path: run_jdata[model_devi_engine=amber]/mdin_prefix

The path prefix to AMBER mdin template files.

mdin:

type: list

argument path: run_jdata[model_devi_engine=amber]/mdin

List of paths to AMBER mdin template files. Each files maps to a system. In the template, the following keywords will be replaced by the actual value: @*freq*@: freq to dump trajectory; @*nstlim*@: total time step to run; @*qm_region*@: AMBER mask of the QM region; @*qm_theory*@: The low level QM theory, such as DFTB2; @*qm_charge*@: The total charge of the QM theory, such as -2; @*rcut*@: cutoff radius of the DPRc model; @*GRAPH_FILE0*@, @*GRAPH_FILE1*@, ... : graph files.

qm_region:

type: list
argument path: run_jdata[model_devi_engine=amber]/qm_region

List of strings. AMBER mask of the QM region. Each mask maps to a system.

qm_charge:

type: list
argument path: run_jdata[model_devi_engine=amber]/qm_charge

List of ints. Charge of the QM region. Each charge maps to a system.

nsteps:

type: list
argument path: run_jdata[model_devi_engine=amber]/nsteps

List of ints. The number of steps to run. Each number maps to a system.

r:

type: list
argument path: run_jdata[model_devi_engine=amber]/r

3D or 4D list of floats. Constrict values for the enhanced sampling. The first dimension maps to systems. The second dimension maps to confs in each system. The third dimension is the constrict value. It can be a single float for 1D or list of floats for nD.

disang_prefix:

type: str, optional
argument path: run_jdata[model_devi_engine=amber]/disang_prefix

The path prefix to disang prefix.

disang:

type: list
argument path: run_jdata[model_devi_engine=amber]/disang

List of paths to AMBER disang files. Each file maps to a sytem. The keyword RVAL will be replaced by the constrict values, or RVAL1, RVAL2, ... for an nD system.

model_devi_f_trust_lo:

type: list | dict | float
argument path:
run_jdata[model_devi_engine=amber]/model_devi_f_trust_lo

Lower bound of forces for the selection. If dict, should be set for each index in sys_idx, respectively.

model_devi_f_trust_hi:

type: list | dict | float
argument path:
run_jdata[model_devi_engine=amber]/model_devi_f_trust_hi

Upper bound of forces for the selection. If dict, should be set for each index in sys_idx, respectively.

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

fp_style:

type: str (flag key)
argument path: run_jdata/fp_style
possible choices: *vasp*, *gaussian*, *siesta*, *cp2k*, *abacus*, *amber/diff*

Software for First Principles.

When fp_style is set to vasp:

fp_pp_path:

type: str
argument path: run_jdata[fp_style=vasp]/fp_pp_path

Directory of psuedo-potential file to be used for 02.fp exists.

fp_pp_files:

type: list
argument path: run_jdata[fp_style=vasp]/fp_pp_files

Psuedo-potential file to be used for 02.fp. Note that the order of elements should correspond to the order in type_map.

fp_incar:

type: str
argument path: run_jdata[fp_style=vasp]/fp_incar

Input file for VASP. INCAR must specify KSPACING and KGAMMA.

fp_aniso_kspacing:

type: list, optional
argument path: run_jdata[fp_style=vasp]/fp_aniso_kspacing

Set anisotropic kspacing. Usually useful for 1-D or 2-D materials. Only support VASP. If it is setting the KSPACING key in INCAR will be ignored.

cvasp:

type: bool, optional
argument path: run_jdata[fp_style=vasp]/cvasp

If cvasp is true, DP-GEN will use Custodian to help control VASP calculation.

ratio_failed:

type: float, optional
argument path: run_jdata[fp_style=vasp]/ratio_failed

Check the ratio of unsuccessfully terminated jobs. If too many FP tasks are not converged, RuntimeError will be raised.

fp_skip_bad_box:

type: str, optional
argument path: run_jdata[fp_style=vasp]/fp_skip_bad_box

Skip the configurations that are obviously unreasonable before 02.fp

When fp_style is set to `gaussian`:

use_clusters:

type: `bool`, optional, default: `False`
argument path: `run_jdata[fp_style=gaussian]/use_clusters`

If set to true, clusters will be taken instead of the whole system.

cluster_cutoff:

type: `float`, optional
argument path: `run_jdata[fp_style=gaussian]/cluster_cutoff`

The soft cutoff radius of clusters if *use_clusters* is set to true. Molecules will be taken as whole even if part of atoms is out of the cluster. Use *cluster_cutoff_hard* to only take atoms within the hard cutoff radius.

cluster_cutoff_hard:

type: `float`, optional
argument path: `run_jdata[fp_style=gaussian]/cluster_cutoff_hard`

The hard cutoff radius of clusters if *use_clusters* is set to true. Outside the hard cutoff radius, atoms will not be taken even if they are in a molecule where some atoms are within the cutoff radius.

cluster_minify:

type: `bool`, optional, default: `False`
argument path: `run_jdata[fp_style=gaussian]/cluster_minify`

If enabled, when an atom within the soft cutoff radius connects a single bond with a non-hydrogen atom out of the soft cutoff radius, the outer atom will be replaced by a hydrogen atom. When the outer atom is a hydrogen atom, the outer atom will be kept. In this case, other atoms out of the soft cutoff radius will be removed.

fp_params:

type: `dict`
argument path: `run_jdata[fp_style=gaussian]/fp_params`

Parameters for Gaussian calculation.

keywords:

type: `list | str`
argument path:
`run_jdata[fp_style=gaussian]/fp_params/keywords`

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

multiplicity:

type: `int | str`, optional, default: `auto`
argument path:
`run_jdata[fp_style=gaussian]/fp_params/multiplicity`

Spin multiplicity for Gaussian input. If *auto*, multiplicity will be detected automatically, with the following rules: when fragment_guesses=True, multiplicity will +1 for each radical, and +2 for each oxygen molecule; when

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

nproc:

type: int

argument path: run_jdata[fp_style=gaussian]/fp_params/nproc

The number of processors for Gaussian input.

charge:

type: int, optional, default: 0

argument path: run_jdata[fp_style=gaussian]/fp_params/charge

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

fragment_guesses:

type: bool, optional, default: False

argument path: run_jdata[fp_style=gaussian]/fp_params/fragment_guesses

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

basis_set:

type: str, optional

argument path: run_jdata[fp_style=gaussian]/fp_params/basis_set

Custom basis set.

keywords_high_multiplicity:

type: str, optional

argument path: run_jdata[fp_style=gaussian]/fp_params/keywords_high_multiplicity

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

ratio_failed:

type: float, optional

argument path: run_jdata[fp_style=gaussian]/ratio_failed

Check the ratio of unsuccessfully terminated jobs. If too many FP tasks are not converged, RuntimeError will be raised.

When fp_style is set to siesta:

use_clusters:

type: bool, optional

argument path: run_jdata[fp_style=siesta]/use_clusters

If set to true, clusters will be taken instead of the whole system. This option does not work with DeePMD-kit 0.x.

cluster_cutoff:

type: float, optional

argument path: run_jdata[fp_style=siesta]/cluster_cutoff

The cutoff radius of clusters if use_clusters is set to true.

fp_params:

type: dict
argument path: run_jdata[fp_style=siesta]/fp_params

Parameters for siesta calculation.

ecut:

type: int
argument path: run_jdata[fp_style=siesta]/fp_params/ecut

Define the plane wave cutoff for grid.

ediff:

type: float
argument path: run_jdata[fp_style=siesta]/fp_params/ediff

Tolerance of Density Matrix.

kspacing:

type: float
argument path: run_jdata[fp_style=siesta]/fp_params/kspacing

Sample factor in Brillouin zones.

mixingWeight:

type: float
argument path:
run_jdata[fp_style=siesta]/fp_params/mixingWeight

Proportion a of output Density Matrix to be used for the input Density Matrix of next SCF cycle (linear mixing).

NumberPulay:

type: int
argument path:
run_jdata[fp_style=siesta]/fp_params/NumberPulay

Controls the Pulay convergence accelerator.

fp_pp_path:

type: str
argument path: run_jdata[fp_style=siesta]/fp_pp_path

Directory of psuedo-potential or numerical orbital files to be used for 02.fp exists.

fp_pp_files:

type: list
argument path: run_jdata[fp_style=siesta]/fp_pp_files

Psuedo-potential file to be used for 02.fp. Note that the order of elements should correspond to the order in type_map.

When fp_style is set to cp2k:

user_fp_params:

type: dict, optional, alias: *fp_params*
argument path: run_jdata[fp_style=cp2k]/user_fp_params

Parameters for cp2k calculation. find detail in manual.cp2k.org. only the kind section must be set before use. we assume that you have basic knowledge for cp2k input.

external_input_path:

type: str, optional
argument path: run_jdata[fp_style=cp2k]/external_input_path

Conflict with key:user_fp_params, use the template input provided by user, some rules should be followed, read the following text in detail.

ratio_failed:

type: float, optional
argument path: run_jdata[fp_style=cp2k]/ratio_failed

Check the ratio of unsuccessfully terminated jobs. If too many FP tasks are not converged, RuntimeError will be raised.

When fp_style is set to abacus:

fp_pp_path:

type: str
argument path: run_jdata[fp_style=abacus]/fp_pp_path

Directory of psuedo-potential or numerical orbital files to be used for 02.fp exists.

fp_pp_files:

type: list
argument path: run_jdata[fp_style=abacus]/fp_pp_files

Psuedo-potential file to be used for 02.fp. Note that the order of elements should correspond to the order in type_map.

fp_orb_files:

type: list, optional
argument path: run_jdata[fp_style=abacus]/fp_orb_files

numerical orbital file to be used for 02.fp when using LCAO basis. Note that the order of elements should correspond to the order in type_map.

fp_incar:

type: str, optional
argument path: run_jdata[fp_style=abacus]/fp_incar

Input file for ABACUS. This is optinal but priority over user_fp_params, one can also setting the key and value of INPUT in user_fp_params.

fp_kpt_file:

type: str, optional
argument path: run_jdata[fp_style=abacus]/fp_kpt_file

KPT file for ABACUS.

fp_dpks_descriptor:

type: str, optional
argument path: run_jdata[fp_style=abacus]/fp_dpks_descriptor

DeePKS descriptor file name. The file should be in pseudopotential directory.

user_fp_params:

type: dict, optional
argument path: run_jdata[fp_style=abacus]/user_fp_params

Set the key and value of INPUT.

k_points:

type: list, optional
argument path: run_jdata[fp_style=abacus]/k_points

Monkhorst-Pack k-grids setting for generating KPT file of ABACUS

When fp_style is set to amber/diff:

Amber/diff style for DPRc models. Note: this fp style only supports to be used with model_devi_engine *amber*, where some arguments are reused. The command argument in the machine file should be path to sander. One should also install dpamber and make it visible in the PATH.

high_level:

type: str
argument path: run_jdata[fp_style=amber/diff]/high_level

High level method. The value will be filled into mdin template as @qm_theory@.

fp_params:

type: dict
argument path: run_jdata[fp_style=amber/diff]/fp_params

Parameters for FP calculation.

high_level_mdin:

type: str
argument path: run_jdata[fp_style=amber/diff]/fp_params/high_level_mdin

Path to high-level AMBER mdin template file. %qm_theory%, %qm_region%, and %qm_charge% will be replaced.

low_level_mdin:

type: str
argument path: run_jdata[fp_style=amber/diff]/fp_params/low_level_mdin

Path to low-level AMBER mdin template file. %qm_theory%, %qm_region%, and %qm_charge% will be replaced.

4.5 dpgen run machine parameters

Note: One can load, modify, and export the input file by using our effective web-based tool DP-GUI. All parameters below can be set in DP-GUI. By clicking "SAVE JSON", one can download the input file.

run_mdata:

type: dict
argument path: run_mdata

machine.json file

api_version:

type: str
argument path: run_mdata/api_version

Please set to 1.0

deepmd_version:

type: str, optional, default: 2
argument path: run_mdata/deepmd_version

DeePMD-kit version, e.g. 2.1.3

train:

type: dict
argument path: run_mdata/train

Parameters of command, machine, and resources for train

command:

type: str
argument path: run_mdata/train/command

Command of a program.

machine:

type: dict
argument path: run_mdata/train/machine

batch_type:

type: str
argument path: run_mdata/train/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: run_mdata/train/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional

argument path: run_mdata/train/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False

argument path: run_mdata/train/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)

argument path: run_mdata/train/machine/context_type

possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict

argument path: run_mdata/train/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: run_mdata/train/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str

argument path: run_mdata/train/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path:
run_mdata/train/machine[LebesgueContext]/
remote_profile/program_id

Program ID

keep_backup:

type: bool, optional
argument path:
run_mdata/train/machine[LebesgueContext]/
remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict
argument path:
run_mdata/train/machine[LebesgueContext]/
remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict
argument path: run_mdata/train/
machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str
argument path:
run_mdata/train/machine[SSHContext]/
remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str
argument path:
run_mdata/train/machine[SSHContext]/
remote_profile/username

username of target linux system

password:

type: str, optional
argument path:
run_mdata/train/machine[SSHContext]/
remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22
argument path: run_mdata/train/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None
argument path: run_mdata/train/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None
argument path: run_mdata/train/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10
argument path: run_mdata/train/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None
argument path: run_mdata/train/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True
argument path: run_mdata/train/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: run_mdata/train/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: run_mdata/train/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: run_mdata/train/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: run_mdata/train/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: run_mdata/train/machine[DpCloudServerContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: run_mdata/train/machine[DpCloudServerContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: run_mdata/train/machine[DpCloudServerContext]/remote_profile/input_data

Configuration of job

When context_type is set to `LazyLocalContext` (or its aliases `lazylocalcontext`, `LazyLocal`, `lazylocal`):

remote_profile:

type: `dict`, optional
argument path: `run_mdata/train/machine[LazyLocalContext]/remote_profile`

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to `HDFSContext` (or its aliases `hdfscontext`, `HDFS`, `hdfs`):

remote_profile:

type: `dict`, optional
argument path: `run_mdata/train/machine[HDFSContext]/remote_profile`

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: `dict`
argument path: `run_mdata/train/resources`

number_node:

type: `int`, optional, default: 1
argument path: `run_mdata/train/resources/number_node`

The number of node need for each *job*

cpu_per_node:

type: `int`, optional, default: 1
argument path: `run_mdata/train/resources/cpu_per_node`

cpu numbers of each node assigned to each job.

gpu_per_node:

type: `int`, optional, default: `0`
argument path: `run_mdata/train/resources/gpu_per_node`

gpu numbers of each node assigned to each job.

queue_name:

type: `str`, optional, default: ``````
argument path: `run_mdata/train/resources/queue_name`

The queue name of batch job scheduler system.

group_size:

type: int
argument path:
run_mdata/train/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional
argument path:
run_mdata/train/resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional
argument path:
run_mdata/train/resources/strategy

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False
argument path:
run_mdata/train/resources/strategy/if_cuda_multi_devices

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: float, optional, default: 0.0
argument path:
run_mdata/train/resources/strategy/ratio_unfinished

The ratio of *jobs* that can be unfinished.

para_deg:

type: int, optional, default: 1
argument path:
run_mdata/train/resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: list, optional, default: []

argument path:
run_mdata/train/resources/source_list

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False
argument path:
run_mdata/train/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []
argument path: run_mdata/train/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []
argument path:
run_mdata/train/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path:
run_mdata/train/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path:
run_mdata/train/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path:
run_mdata/train/resources/batch_type

possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to `Shell` (or its alias `shell`):

kwargs:

type: `dict`, optional
argument path: `run_mdata/train/resources[Shell]/kwargs`

This field is empty for this batch.

When batch_type is set to `Slurm` (or its alias `slurm`):

kwargs:

type: `dict`, optional
argument path: `run_mdata/train/resources[Slurm]/kwargs`

Extra arguments.

custom_gpu_line:

type: `NoneType` | `str`, optional, default: `None`
argument path: `run_mdata/train/resources[Slurm]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #SBATCH

When batch_type is set to `PBS` (or its alias `pbs`):

kwargs:

type: `dict`, optional
argument path: `run_mdata/train/resources[PBS]/kwargs`

This field is empty for this batch.

When batch_type is set to `Torque` (or its alias `torque`):

kwargs:

type: `dict`, optional
argument path: `run_mdata/train/resources[Torque]/kwargs`

This field is empty for this batch.

When batch_type is set to `SlurmJobArray` (or its alias `slurmjobarray`):

kwargs:

type: dict, optional
argument path: run_mdata/train/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: run_mdata/train/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional
argument path: run_mdata/train/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict
argument path: run_mdata/train/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False
argument path: run_mdata/train/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False
argument path: run_mdata/train/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True

argument path: `run_mdata/train/resources[LSF]/kwargs/gpu_exclusive`

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

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

argument path: `run_mdata/train/resources[LSF]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #BSUB

When batch_type is set to `Lebesgue` (or its alias `lebesgue`):

kwargs:

type: `dict`, optional

argument path: `run_mdata/train/resources[Lebesgue]/kwargs`

This field is empty for this batch.

When batch_type is set to `DistributedShell` (or its alias `distributedshell`):

kwargs:

type: `dict`, optional

argument path: `run_mdata/train/resources[DistributedShell]/kwargs`

This field is empty for this batch.

user_forward_files:

type: `list`, optional

argument path: `run_mdata/train/user_forward_files`

Files to be forwarded to the remote machine.

user_backward_files:

type: `list`, optional

argument path: `run_mdata/train/user_backward_files`

Files to be backwarded from the remote machine.

model_devi:

type: `dict`

argument path: `run_mdata/model_devi`

Parameters of command, machine, and resources for model_devi

command:

type: str
argument path: run_mdata/model_devi/command

Command of a program.

machine:

type: dict
argument path: run_mdata/model_devi/machine

batch_type:

type: str
argument path: run_mdata/model_devi/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: run_mdata/model_devi/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional
argument path: run_mdata/model_devi/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False
argument path: run_mdata/model_devi/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)
argument path: run_mdata/model_devi/machine/context_type
possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*,

DpCloudServerContext, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to `LebesgueContext` (or its aliases `lebesguecontext`, `Lebesgue`, `lebesgue`):

remote_profile:

type: `dict`

argument path: `run_mdata/model_devi/machine[LebesgueContext]/remote_profile`

The information used to maintain the connection with remote machine.

email:

type: `str`

argument path: `run_mdata/model_devi/machine[LebesgueContext]/remote_profile/email`

Email

password:

type: `str`

argument path: `run_mdata/model_devi/machine[LebesgueContext]/remote_profile/password`

Password

program_id:

type: `int`, alias: *project_id*

argument path: `run_mdata/model_devi/machine[LebesgueContext]/remote_profile/program_id`

Program ID

keep_backup:

type: `bool`, optional

argument path: `run_mdata/model_devi/machine[LebesgueContext]/remote_profile/keep_backup`

keep download and upload zip

input_data:

type: `dict`

argument path:
run_mdata/model_devi/
machine[LebesgueContext]/
remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict
argument path: run_mdata/model_devi/
machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str
argument path:
run_mdata/model_devi/
machine[SSHContext]/
remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str
argument path:
run_mdata/model_devi/
machine[SSHContext]/
remote_profile/username

username of target linux system

password:

type: str, optional
argument path:
run_mdata/model_devi/
machine[SSHContext]/
remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22
argument path:
run_mdata/model_devi/
machine[SSHContext]/
remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: run_mdata/model_devi/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: run_mdata/model_devi/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: run_mdata/model_devi/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: run_mdata/model_devi/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: run_mdata/model_devi/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: run_mdata/model_devi/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: run_mdata/model_devi/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: run_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: run_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: run_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: run_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: run_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/input_data

Configuration of job

When context_type is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

remote_profile:

type: dict, optional

argument path: run_mdata/model_devi/machine[LazyLocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

remote_profile:

type: dict, optional

argument path: run_mdata/model_devi/machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict

argument path: run_mdata/model_devi/resources

number_node:

type: int, optional, default: 1

argument path: run_mdata/model_devi/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1

argument path: run_mdata/model_devi/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0
argument path: run_mdata/model_devi/resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str, optional, default: ````
argument path: run_mdata/model_devi/resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int
argument path: run_mdata/model_devi/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional
argument path: run_mdata/model_devi/resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional
argument path: run_mdata/model_devi/resources/strategy

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False
argument path: run_mdata/model_devi/resources/strategy/if_cuda_multi_devices

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: float, optional, default: 0.0
argument path: run_mdata/model_devi/resources/strategy/ratio_unfinished

The ratio of *jobs* that can be unfinished.

para_deg:

type: int, optional, default: 1
argument path: run_mdata/model_devi/resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: list, optional, default: []
argument path: run_mdata/model_devi/resources/source_list

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False
argument path: run_mdata/model_devi/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []
argument path: run_mdata/model_devi/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []
argument path: run_mdata/model_devi/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path: run_mdata/model_devi/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path: run_mdata/model_devi/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path: run_mdata/model_devi/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional
argument path: run_mdata/model_devi/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: run_mdata/model_devi/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: run_mdata/model_devi/resources[Slurm]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional

argument path: `run_mdata/model_devi/resources[PBS]/kwargs`

This field is empty for this batch.

When batch_type is set to `Torque` (or its alias `torque`):

kwargs:

type: `dict`, optional
argument path: `run_mdata/model_devi/resources[Torque]/kwargs`

This field is empty for this batch.

When batch_type is set to `SlurmJobArray` (or its alias `slurmjobarray`):

kwargs:

type: `dict`, optional
argument path: `run_mdata/model_devi/resources[SlurmJobArray]/kwargs`

Extra arguments.

custom_gpu_line:

type: `NoneType | str`, optional, default: `None`
argument path: `run_mdata/model_devi/resources[SlurmJobArray]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #SBATCH

When batch_type is set to `DpCloudServer` (or its alias `dpcloudserver`):

kwargs:

type: `dict`, optional
argument path: `run_mdata/model_devi/resources[DpCloudServer]/kwargs`

This field is empty for this batch.

When batch_type is set to `LSF` (or its alias `lsf`):

kwargs:

type: `dict`
argument path: `run_mdata/model_devi/resources[LSF]/kwargs`

Extra arguments.

gpu_usage:

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

argument path: `run_mdata/model_devi/resources[LSF]/kwargs/gpu_usage`

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

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

argument path: `run_mdata/model_devi/resources[LSF]/kwargs/gpu_new_syntax`

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

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

argument path: `run_mdata/model_devi/resources[LSF]/kwargs/gpu_exclusive`

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

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

argument path: `run_mdata/model_devi/resources[LSF]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #BSUB

When batch_type is set to `Lebesgue` (or its alias `lebesgue`):

kwargs:

type: `dict`, optional

argument path: `run_mdata/model_devi/resources[Lebesgue]/kwargs`

This field is empty for this batch.

When batch_type is set to `DistributedShell` (or its alias `distributedshell`):

kwargs:

type: `dict`, optional

argument path: `run_mdata/model_devi/resources[DistributedShell]/kwargs`

This field is empty for this batch.

user_forward_files:

type: list, optional
argument path: run_mdata/model_devi/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional
argument path: run_mdata/model_devi/user_backward_files

Files to be backwarded from the remote machine.

fp:

type: dict
argument path: run_mdata/fp

Parameters of command, machine, and resources for fp

command:

type: str
argument path: run_mdata/fp/command

Command of a program.

machine:

type: dict
argument path: run_mdata/fp/machine

batch_type:

type: str
argument path: run_mdata/fp/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: run_mdata/fp/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional
argument path: run_mdata/fp/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False
argument path: run_mdata/fp/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)
argument path: run_mdata/fp/machine/context_type
possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict
argument path: run_mdata/fp/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str
argument path: run_mdata/fp/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str
argument path: run_mdata/fp/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: run_mdata/fp/machine[LebesgueContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: run_mdata/fp/machine[LebesgueContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: run_mdata/fp/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: run_mdata/fp/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: run_mdata/fp/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: run_mdata/fp/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: run_mdata/fp/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: run_mdata/fp/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: run_mdata/fp/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: run_mdata/fp/machine[DpCloudServerContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: run_mdata/fp/machine[DpCloudServerContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: run_mdata/fp/machine[DpCloudServerContext]/remote_profile/input_data

Configuration of job

When context_type is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

remote_profile:

type: dict, optional

argument path: run_mdata/fp/machine[LazyLocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

remote_profile:

type: dict, optional

argument path: run_mdata/fp/machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict

argument path: run_mdata/fp/resources

number_node:

type: int, optional, default: 1

argument path: run_mdata/fp/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1

argument path: run_mdata/fp/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0

argument path:
`run_mdata/fp/resources/gpu_per_node`

gpu numbers of each node assigned to each job.

queue_name:

type: `str`, optional, default: ````
argument path:
`run_mdata/fp/resources/queue_name`

The queue name of batch job scheduler system.

group_size:

type: `int`
argument path:
`run_mdata/fp/resources/group_size`

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: `list`, optional
argument path:
`run_mdata/fp/resources/custom_flags`

The extra lines pass to job submitting script header

strategy:

type: `dict`, optional
argument path:
`run_mdata/fp/resources/strategy`

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: `bool`, optional, default: `False`
argument path: `run_mdata/fp/resources/strategy/if_cuda_multi_devices`

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: `float`, optional, default: `0.0`
argument path:
`run_mdata/fp/resources/strategy/ratio_unfinished`

The ratio of *jobs* that can be unfinished.

para_deg:

type: int, optional, default: 1
argument path: run_mdata/fp/resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: list, optional, default: []
argument path: run_mdata/fp/resources/source_list

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False
argument path: run_mdata/fp/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []
argument path: run_mdata/fp/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []
argument path: run_mdata/fp/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path: run_mdata/fp/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0

argument path:
run_mdata/fp/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path:
run_mdata/fp/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional
argument path: run_mdata/fp/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: run_mdata/fp/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: run_mdata/fp/resources[Slurm]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional
argument path:
run_mdata/fp/resources[PBS]/kwargs

This field is empty for this batch.

When batch_type is set to Torque (or its alias torque):

kwargs:

type: dict, optional
argument path: run_mdata/fp/resources[Torque]/kwargs

This field is empty for this batch.

When batch_type is set to SlurmJobArray (or its alias slurmjobarray):

kwargs:

type: dict, optional
argument path: run_mdata/fp/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: run_mdata/fp/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional
argument path: run_mdata/fp/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict
argument path: run_mdata/fp/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False
argument path: run_mdata/fp/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

argument path: run_mdata/fp/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True

argument path: run_mdata/fp/resources[LSF]/kwargs/gpu_exclusive

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: run_mdata/fp/resources[LSF]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #BSUB

When batch_type is set to Lebesgue (or its alias lebesgue):

kwargs:

type: dict, optional

argument path: run_mdata/fp/resources[Lebesgue]/kwargs

This field is empty for this batch.

When batch_type is set to DistributedShell (or its alias distributedshell):

kwargs:

type: dict, optional

argument path: run_mdata/fp/resources[DistributedShell]/kwargs

This field is empty for this batch.

user_forward_files:

type: list, optional

argument path: run_mdata/fp/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional
argument path: run_mdata/fp/user_backward_files

Files to be backwarded from the remote machine.

CHAPTER

FIVE

INIT

5.1 Init_bulk

You may prepare initial data for bulk systems with VASP by:

```
dpgen init_bulk PARAM [MACHINE]
```

The MACHINE configure file is optional. If this parameter exists, then the optimization tasks or MD tasks will be submitted automatically according to MACHINE.json.

Basically `init_bulk` can be divided into four parts , denoted as `stages` in `PARAM`:

1. Relax in folder `00.place_ele`
2. Perturb and scale in folder `01.scale_pert`
3. Run a short AIMD in folder `02.md`
4. Collect data in folder `02.md`.

All stages must be **in order**. One doesn't need to run all stages. For example, you may run stage 1 and 2, generating supercells as starting point of exploration in `dpgen run`.

If MACHINE is None, there should be only one stage in stages. Corresponding tasks will be generated, but user's intervention should be involved in, to manually run the scripts.

Following is an example for `PARAM`, which generates data from a typical structure hcp.

```
{
    "stages" : [1,2,3,4],
    "cell_type":    "hcp",
    "latt":     4.479,
    "super_cell":   [2, 2, 2],
    "elements":     ["Mg"],
    "potcars":      ["....../POTCAR"],
    "relax_incar": "....../INCAR_metal_rlx",
    "md_incar" : "....../INCAR_metal_md",
    "scale":        [1.00],
    "skip_relax":   false,
    "pert_numb":    2,
    "md_nstep" : 5,
    "pert_box":     0.03,
    "pert_atom":    0.01,
    "coll_ndata":   5000,
    "type_map" : [ "Mg", "Al"],
```

(continues on next page)

(continued from previous page)

```
    "_comment":     "that's all"
}
```

If you want to specify a structure as starting point for `init_bulk`, you may set in `PARAM` as follows.

```
"from_poscar":        true,
"from_poscar_path":        "....../C_mp-47_conventional.POSCAR",
```

`init_bulk` supports both VASP and ABACUS for first-principle calculation. You can choose the software by specifying the key `init_fp_style`. If `init_fp_style` is not specified, the default software will be VASP.

When using ABACUS for `init_fp_style`, the keys of the paths of `INPUT` files for relaxation and MD simulations are the same as `INCAR` for VASP, which are `relax_incar` and `md_incar` respectively. Use `relax_kpt` and `md_kpt` for the relative path for `KPT` files of relaxation and MD simulations. They two can be omitted if `kspacing` (in unit of 1/Bohr) or `gamma_only` has been set in corresponding INPUT files. If `from_poscar` is set to `false`, you have to specify `atom_masses` in the same order as `elements`.

5.2 dpgen init_bulk machine parameters

init_bulk_mdata:

type: dict
argument path: init_bulk_mdata

machine.json file

api_version:

type: str
argument path: init_bulk_mdata/api_version

Please set to 1.0

deepmd_version:

type: str, optional, default: 2
argument path: init_bulk_mdata/deepmd_version

DeePMD-kit version, e.g. 2.1.3

fp:

type: dict
argument path: init_bulk_mdata/fp

Parameters of command, machine, and resources for fp

command:

type: str
argument path: init_bulk_mdata/fp/command

Command of a program.

machine:

type: dict
argument path: init_bulk_mdata/fp/machine

batch_type:

type: str
argument path: init_bulk_mdata/fp/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: init_bulk_mdata/fp/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional
argument path: init_bulk_mdata/fp/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False
argument path: init_bulk_mdata/fp/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)
argument path: init_bulk_mdata/fp/machine/context_type
possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict

argument path: init_bulk_mdata/fp/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: init_bulk_mdata/fp/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str

argument path: init_bulk_mdata/fp/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: init_bulk_mdata/fp/machine[LebesgueContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: init_bulk_mdata/fp/machine[LebesgueContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: init_bulk_mdata/fp/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: init_bulk_mdata/fp/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: init_bulk_mdata/fp/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: init_bulk_mdata/fp/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path:
`init_bulk_mdata/fp/`
`machine[DpCloudServerContext]/`
`remote_profile/email`

Email

password:

type: `str`

argument path:
`init_bulk_mdata/fp/`
`machine[DpCloudServerContext]/`
`remote_profile/password`

Password

program_id:

type: `int`, alias: *project_id*

argument path:
`init_bulk_mdata/fp/`
`machine[DpCloudServerContext]/`
`remote_profile/program_id`

Program ID

keep_backup:

type: `bool`, optional

argument path:
`init_bulk_mdata/fp/`
`machine[DpCloudServerContext]/`
`remote_profile/keep_backup`

keep download and upload zip

input_data:

type: `dict`

argument path:
`init_bulk_mdata/fp/`
`machine[DpCloudServerContext]/`
`remote_profile/input_data`

Configuration of job

When context_type is set to `LazyLocalContext` (or its aliases `lazylocalcontext`, `LazyLocal`, `lazylocal`):

remote_profile:

type: `dict`, optional

argument path: `init_bulk_mdata/fp/`
`machine[LazyLocalContext]/`
`remote_profile`

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to `HDFSContext` (or its aliases `hdfscontext`, `HDFS`, `hdfs`):

remote_profile:

type: dict, optional
argument path: init_bulk_mdata/fp/machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict
argument path: init_bulk_mdata/fp/resources

number_node:

type: int, optional, default: 1
argument path: init_bulk_mdata/fp/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1
argument path: init_bulk_mdata/fp/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0
argument path: init_bulk_mdata/fp/resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str, optional, default: ````
argument path: init_bulk_mdata/fp/resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int
argument path: init_bulk_mdata/fp/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional
argument path: init_bulk_mdata/fp/resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional

argument path: init_bulk_mdata/fp/resources/strategy

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False

argument path: init_bulk_mdata/fp/resources/strategy/if_cuda_multi_devices

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: float, optional, default: 0.0

argument path: init_bulk_mdata/fp/resources/strategy/ratio_unfinished

The ratio of *jobs* that can be unfinished.

para_deg:

type: int, optional, default: 1

argument path: init_bulk_mdata/fp/resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: list, optional, default: []

argument path: init_bulk_mdata/fp/resources/source_list

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False

argument path: init_bulk_mdata/fp/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []
argument path: init_bulk_mdata/fp/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []
argument path: init_bulk_mdata/fp/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path: init_bulk_mdata/fp/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path: init_bulk_mdata/fp/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path: init_bulk_mdata/fp/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional

argument path: `init_bulk_mdata/fp/resources[Shell]/kwargs`

This field is empty for this batch.

When batch_type is set to `Slurm` (or its alias `slurm`):

kwargs:

type: `dict`, optional
argument path: `init_bulk_mdata/fp/resources[Slurm]/kwargs`

Extra arguments.

custom_gpu_line:

type: `NoneType | str`, optional, default: `None`
argument path: `init_bulk_mdata/fp/resources[Slurm]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #SBATCH

When batch_type is set to `PBS` (or its alias `pbs`):

kwargs:

type: `dict`, optional
argument path: `init_bulk_mdata/fp/resources[PBS]/kwargs`

This field is empty for this batch.

When batch_type is set to `Torque` (or its alias `torque`):

kwargs:

type: `dict`, optional
argument path: `init_bulk_mdata/fp/resources[Torque]/kwargs`

This field is empty for this batch.

When batch_type is set to `SlurmJobArray` (or its alias `slurmjobarray`):

kwargs:

type: `dict`, optional
argument path: `init_bulk_mdata/fp/resources[SlurmJobArray]/kwargs`

Extra arguments.

custom_gpu_line:

type: `NoneType | str`, optional, default: `None`
argument path: `init_bulk_mdata/fp/resources[SlurmJobArray]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #SBATCH

When batch_type is set to `DpCloudServer` (or its alias `dpcloudserver`):

kwargs:

type: `dict`, optional
argument path: `init_bulk_mdata/fp/resources[DpCloudServer]/kwargs`

This field is empty for this batch.

When batch_type is set to `LSF` (or its alias `lsf`):

kwargs:

type: `dict`
argument path: `init_bulk_mdata/fp/resources[LSF]/kwargs`

Extra arguments.

gpu_usage:

type: `bool`, optional, default: `False`
argument path: `init_bulk_mdata/fp/resources[LSF]/kwargs/gpu_usage`

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: `bool`, optional, default: `False`
argument path: `init_bulk_mdata/fp/resources[LSF]/kwargs/gpu_new_syntax`

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: `bool`, optional, default: `True`
argument path: `init_bulk_mdata/fp/resources[LSF]/kwargs/gpu_exclusive`

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

type: `NoneType | str`, optional, default: `None`
argument path: `init_bulk_mdata/fp/resources[LSF]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #BSUB

When batch_type is set to `Lebesgue` (or its alias `lebesgue`):

kwargs:

type: `dict`, optional
argument path: `init_bulk_mdata/fp/resources[Lebesgue]/kwargs`

This field is empty for this batch.

When batch_type is set to `DistributedShell` (or its alias `distributedshell`):

kwargs:

type: `dict`, optional
argument path: `init_bulk_mdata/fp/resources[DistributedShell]/kwargs`

This field is empty for this batch.

user_forward_files:

type: `list`, optional
argument path: `init_bulk_mdata/fp/user_forward_files`

Files to be forwarded to the remote machine.

user_backward_files:

type: `list`, optional
argument path: `init_bulk_mdata/fp/user_backward_files`

Files to be backwarded from the remote machine.

5.3 Init_surf

You may prepare initial data for surface systems with VASP by:

```
dpgen init_surf PARAM [MACHINE]
```

The MACHINE configure file is optional. If this parameter exists, then the optimization tasks or MD tasks will be submitted automatically according to MACHINE.json. That is to say, if one only wants to prepare `surf-xxx/sys-xxx` folders for the second stage but wants to skip relaxation, `dpgen init_surf PARAM` should be used (without `MACHINE`). "stages" and "skip_relax" in `PARAM` should be set as:

```
  "stages": [1,2],
  "skip_relax": true,
```

Basically `init_surf` can be divided into two parts , denoted as `stages` in `PARAM`:

1. Build specific surface in folder `00.place_ele`
2. Pertub and scale in folder `01.scale_pert`

All stages must be **in order**.

Following is an example for `PARAM`, which generates data from a typical structure fcc.

```
{
  "stages": [
    1,
    2
  ],
  "cell_type": "fcc",
  "latt": 4.034,
  "super_cell": [
    2,
    2,
    2
  ],
  "layer_numb": 3,
  "vacuum_max": 9,
  "vacuum_resol": [
    0.5,
    1
  ],
  "mid_point": 4.0,
  "millers": [
    [
      1,
      0,
      0
    ],
    [
      1,
      1,
      0
    ],
    [
      1,
      1,
      1
    ]
  ],
  "elements": [
    "Al"
  ],
  "potcars": [
    "....../POTCAR"
  ],
  "relax_incar": "....../INCAR_metal_rlx_low",
  "scale": [
    1.0
  ],
  "skip_relax": true,
  "pert_numb": 2,
  "pert_box": 0.03,
  "pert_atom": 0.01,
```

(continues on next page)

(continued from previous page)

```
  "_comment": "that's all"
}
```

Another example is `from_poscar` method. Here you need to specify the POSCAR file.

```
{
  "stages": [
    1,
    2
  ],
  "cell_type": "fcc",
  "from_poscar":         true,
  "from_poscar_path":          "POSCAR",
  "super_cell": [
    1,
    1,
    1
  ],
  "layer_numb": 3,
  "vacuum_max": 5,
  "vacuum_resol": [0.5,2],
  "mid_point": 2.0,
  "millers": [
    [
      1,
      0,
      0
    ]
  ],
  "elements": [
    "Al"
  ],
  "potcars": [
    "./POTCAR"
  ],
  "relax_incar" : "INCAR_metal_rlx_low",
  "scale": [
    1.0
  ],
  "skip_relax": true,
  "pert_numb": 5,
  "pert_box": 0.03,
  "pert_atom": 0.01,
  "coll_ndata": 5000,
  "_comment": "that's all"
}
```

5.4 dpgen init_surf machine parameters

init_surf_mdata:

type: dict
argument path: init_surf_mdata

machine.json file

api_version:

type: str
argument path: init_surf_mdata/api_version

Please set to 1.0

deepmd_version:

type: str, optional, default: 2
argument path: init_surf_mdata/deepmd_version

DeePMD-kit version, e.g. 2.1.3

fp:

type: dict
argument path: init_surf_mdata/fp

Parameters of command, machine, and resources for fp

command:

type: str
argument path: init_surf_mdata/fp/command

Command of a program.

machine:

type: dict
argument path: init_surf_mdata/fp/machine

batch_type:

type: str
argument path: init_surf_mdata/fp/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: init_surf_mdata/fp/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional

argument path: init_surf_mdata/fp/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False

argument path: init_surf_mdata/fp/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)

argument path: init_surf_mdata/fp/machine/context_type

possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict

argument path: init_surf_mdata/fp/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: init_surf_mdata/fp/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str

argument path: init_surf_mdata/fp/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: init_surf_mdata/fp/machine[LebesgueContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: init_surf_mdata/fp/machine[LebesgueContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: init_surf_mdata/fp/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: init_surf_mdata/fp/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: init_surf_mdata/fp/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: init_surf_mdata/fp/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: init_surf_mdata/fp/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: init_surf_mdata/fp/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path:
init_surf_mdata/fp/
machine[DpCloudServerContext]/
remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path:
init_surf_mdata/fp/
machine[DpCloudServerContext]/
remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path:
init_surf_mdata/fp/
machine[DpCloudServerContext]/
remote_profile/input_data

Configuration of job

When context_type is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

remote_profile:

type: dict, optional

argument path: init_surf_mdata/fp/
machine[LazyLocalContext]/
remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

remote_profile:

type: dict, optional

argument path: init_surf_mdata/fp/
machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict

argument path: init_surf_mdata/fp/resources

number_node:

type: int, optional, default: 1

argument path: `init_surf_mdata/fp/resources/number_node`

The number of node need for each *job*

`cpu_per_node:`

type: `int`, optional, default: 1
argument path: `init_surf_mdata/fp/resources/cpu_per_node`

cpu numbers of each node assigned to each job.

`gpu_per_node:`

type: `int`, optional, default: `0`
argument path: `init_surf_mdata/fp/resources/gpu_per_node`

gpu numbers of each node assigned to each job.

`queue_name:`

type: `str`, optional, default: ````
argument path: `init_surf_mdata/fp/resources/queue_name`

The queue name of batch job scheduler system.

`group_size:`

type: `int`
argument path: `init_surf_mdata/fp/resources/group_size`

The number of *tasks* in a *job*. 0 means infinity.

`custom_flags:`

type: `list`, optional
argument path: `init_surf_mdata/fp/resources/custom_flags`

The extra lines pass to job submitting script header

`strategy:`

type: `dict`, optional
argument path: `init_surf_mdata/fp/resources/strategy`

strategies we use to generation job submitting scripts.

`if_cuda_multi_devices:`

type: `bool`, optional, default: `False`
argument path: `init_surf_mdata/fp/resources/strategy/if_cuda_multi_devices`

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

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

argument path: `init_surf_mdata/fp/resources/strategy/ratio_unfinished`

The ratio of *jobs* that can be unfinished.

para_deg:

type: `int`, optional, default: 1

argument path: `init_surf_mdata/fp/resources/para_deg`

Decide how many tasks will be run in parallel.

source_list:

type: `list`, optional, default: `[]`

argument path: `init_surf_mdata/fp/resources/source_list`

The env file to be sourced before the command execution.

module_purge:

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

argument path: `init_surf_mdata/fp/resources/module_purge`

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: `list`, optional, default: `[]`

argument path: `init_surf_mdata/fp/resources/module_unload_list`

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: `list`, optional, default: `[]`

argument path: `init_surf_mdata/fp/resources/module_list`

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path: init_surf_mdata/fp/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path: init_surf_mdata/fp/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path: init_surf_mdata/fp/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional
argument path: init_surf_mdata/fp/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: init_surf_mdata/fp/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: `init_surf_mdata/fp/resources[Slurm]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #SBATCH

When batch_type is set to `PBS` (or its alias `pbs`):

kwargs:

type: `dict`, optional

argument path: `init_surf_mdata/fp/resources[PBS]/kwargs`

This field is empty for this batch.

When batch_type is set to `Torque` (or its alias `torque`):

kwargs:

type: `dict`, optional

argument path: `init_surf_mdata/fp/resources[Torque]/kwargs`

This field is empty for this batch.

When batch_type is set to `SlurmJobArray` (or its alias `slurmjobarray`):

kwargs:

type: `dict`, optional

argument path: `init_surf_mdata/fp/resources[SlurmJobArray]/kwargs`

Extra arguments.

custom_gpu_line:

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

argument path: `init_surf_mdata/fp/resources[SlurmJobArray]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #SBATCH

When batch_type is set to `DpCloudServer` (or its alias `dpcloudserver`):

kwargs:

type: `dict`, optional

argument path: `init_surf_mdata/fp/resources[DpCloudServer]/kwargs`

This field is empty for this batch.

When batch_type is set to `LSF` (or its alias `lsf`):

kwargs:

type: dict
argument path: init_surf_mdata/fp/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False
argument path: init_surf_mdata/fp/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False
argument path: init_surf_mdata/fp/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True
argument path: init_surf_mdata/fp/resources[LSF]/kwargs/gpu_exclusive

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: init_surf_mdata/fp/resources[LSF]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #BSUB

When batch_type is set to Lebesgue (or its alias lebesgue):

kwargs:

type: dict, optional
argument path: init_surf_mdata/fp/resources[Lebesgue]/kwargs

This field is empty for this batch.

When batch_type is set to DistributedShell (or its alias distributedshell):

kwargs:

type: dict, optional
argument path: init_surf_mdata/fp/resources[DistributedShell]/kwargs

This field is empty for this batch.

user_forward_files:

type: list, optional
argument path: init_surf_mdata/fp/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional
argument path: init_surf_mdata/fp/user_backward_files

Files to be backwarded from the remote machine.

5.5 init_reaction

dpgen init_reaction is a workflow to initilize data for reactive systems of small gas-phase molecules. The workflow was introduced in the "Initialization" section of Energy & Fuels, 2021, 35 (1), 762–769.

To start the workflow, one needs a box containing reactive systems. The following packages are required for each of the step:

- Exploring: LAMMPS
- Sampling: MDDatasetBuilder
- Labeling: Gaussian

The Exploring step uses LAMMPS pair_style reaxff to run a short ReaxMD NVT MD simulation. In the Sampling step, molecular clusters are taken and k-means clustering algorithm is applied to remove the redundancy, which is described in Nature Communications, 11, 5713 (2020). The Labeling step calculates energies and forces using the Gaussian package.

An example of reaction.json is given below:

```
{
    "type_map": [
        "H",
        "O"
    ],
    "reaxff": {
        "data": "data.hydrogen",
        "ff": "ffield.reax.cho",
        "control": "lmp_control",
        "temp": 3000,
        "tau_t": 100,
        "dt": 0.1,
```

(continues on next page)

(continued from previous page)

```
        "nstep": 10000,
        "dump_freq": 100
    },
    "cutoff": 3.5,
    "dataset_size": 100,
    "qmkeywords": "b3lyp/6-31g** force Geom=PrintInputOrient"
}
```

For detailed parameters, see *parametes* and *machine parameters*.

The genereated data can be used to continue DP-GEN concurrent learning workflow. Read Energy & Fuels, 2021, 35 (1), 762–769 for details.

5.6 dpgen init_reaction parameters

init_reaction_jdata:

type: dict
argument path: init_reaction_jdata

Generate initial data for reactive systems for small gas-phase molecules, from a ReaxFF NVT MD trajectory.

type_map:

type: list
argument path: init_reaction_jdata/type_map

Type map, which should match types in the initial data. e.g. [“C”, “H”, “O”]

reaxff:

type: dict
argument path: init_reaction_jdata/reaxff

Parameters for ReaxFF NVT MD.

data:

type: str
argument path: init_reaction_jdata/reaxff/data

Path to initial LAMMPS data file. The atom_style should be charge.

ff:

type: str
argument path: init_reaction_jdata/reaxff/ff

Path to ReaxFF force field file. Available in the lammps/potentials directory.

control:

type: str
argument path: init_reaction_jdata/reaxff/control

Path to ReaxFF control file.

temp:

type: int | float
argument path: init_reaction_jdata/reaxff/temp

Target Temperature for the NVT MD simulation. Unit: K.

dt:

type: int | float
argument path: init_reaction_jdata/reaxff/dt

Real time for every time step. Unit: fs.

tau_t:

type: int | float
argument path: init_reaction_jdata/reaxff/tau_t

Time to determine how rapidly the temperature. Unit: fs.

dump_freq:

type: int
argument path: init_reaction_jdata/reaxff/dump_freq

Frequency of time steps to collect trajectory.

nstep:

type: int
argument path: init_reaction_jdata/reaxff/nstep

Total steps to run the ReaxFF MD simulation.

cutoff:

type: float
argument path: init_reaction_jdata/cutoff

Cutoff radius to take clusters from the trajectory. Note that only a complete molecule or free radical will be taken.

dataset_size:

type: int
argument path: init_reaction_jdata/dataset_size

Collected dataset size for each bond type.

qmkeywords:

type: str
argument path: init_reaction_jdata/qmkeywords

Gaussian keywords for first-principle calculations. e.g. force mn15/6-31g** Geom=PrintInputOrient. Note that "force" job is necessary to collect data. Geom=PrintInputOrient should be used when there are more than 50 atoms in a cluster.

5.7 dpgen init_reaction machine parameters

init_reaction_mdata:

type: dict
argument path: init_reaction_mdata

machine.json file

api_version:

type: str
argument path: init_reaction_mdata/api_version

Please set to 1.0

deepmd_version:

type: str, optional, default: 2
argument path: init_reaction_mdata/deepmd_version

DeePMD-kit version, e.g. 2.1.3

reaxff:

type: dict
argument path: init_reaction_mdata/reaxff

Parameters of command, machine, and resources for reaxff

command:

type: str
argument path: init_reaction_mdata/reaxff/command

Command of a program.

machine:

type: dict
argument path: init_reaction_mdata/reaxff/machine

batch_type:

type: str
argument path: init_reaction_mdata/reaxff/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: init_reaction_mdata/reaxff/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional
argument path: init_reaction_mdata/reaxff/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False
argument path: init_reaction_mdata/reaxff/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)
argument path: init_reaction_mdata/reaxff/machine/context_type
possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict
argument path: init_reaction_mdata/reaxff/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str
argument path: init_reaction_mdata/reaxff/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str

argument path: init_reaction_mdata/reaxff/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: init_reaction_mdata/reaxff/machine[LebesgueContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: init_reaction_mdata/reaxff/machine[LebesgueContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: init_reaction_mdata/reaxff/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: init_reaction_mdata/reaxff/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: init_reaction_mdata/reaxff/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: init_reaction_mdata/reaxff/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: init_reaction_mdata/reaxff/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: init_reaction_mdata/reaxff/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/reaxff/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/reaxff/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: init_reaction_mdata/reaxff/

machine[SSHContext]/
remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/reaxff/ machine[SSHContext]/ remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: init_reaction_mdata/reaxff/ machine[SSHContext]/ remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: init_reaction_mdata/ reaxff/machine[LocalContext]/ remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: init_reaction_mdata/reaxff/ machine[DpCloudServerContext]/ remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path:
`init_reaction_mdata/reaxff/machine[DpCloudServerContext]/remote_profile/email`

Email

password:

type: `str`

argument path:
`init_reaction_mdata/reaxff/machine[DpCloudServerContext]/remote_profile/password`

Password

program_id:

type: `int`, alias: *project_id*

argument path:
`init_reaction_mdata/reaxff/machine[DpCloudServerContext]/remote_profile/program_id`

Program ID

keep_backup:

type: `bool`, optional

argument path:
`init_reaction_mdata/reaxff/machine[DpCloudServerContext]/remote_profile/keep_backup`

keep download and upload zip

input_data:

type: `dict`

argument path:
`init_reaction_mdata/reaxff/machine[DpCloudServerContext]/remote_profile/input_data`

Configuration of job

When context_type is set to `LazyLocalContext` (or its aliases `lazylocalcontext`, `LazyLocal`, `lazylocal`):

remote_profile:

type: `dict`, optional

argument path: `init_reaction_mdata/reaxff/machine[LazyLocalContext]/remote_profile`

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to `HDFSContext` (or its aliases `hdfscontext`, `HDFS`, `hdfs`):

remote_profile:

type: dict, optional
argument path: init_reaction_mdata/reaxff/machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict
argument path: init_reaction_mdata/reaxff/resources

number_node:

type: int, optional, default: 1
argument path: init_reaction_mdata/reaxff/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1
argument path: init_reaction_mdata/reaxff/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0
argument path: init_reaction_mdata/reaxff/resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str, optional, default: ````
argument path: init_reaction_mdata/reaxff/resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int
argument path: init_reaction_mdata/reaxff/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional

argument path: `init_reaction_mdata/reaxff/resources/custom_flags`

The extra lines pass to job submitting script header

strategy:

type: `dict`, optional
argument path: `init_reaction_mdata/reaxff/resources/strategy`

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: `bool`, optional, default: `False`
argument path: `init_reaction_mdata/reaxff/resources/strategy/if_cuda_multi_devices`

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: `float`, optional, default: `0.0`
argument path: `init_reaction_mdata/reaxff/resources/strategy/ratio_unfinished`

The ratio of *jobs* that can be unfinished.

para_deg:

type: `int`, optional, default: 1
argument path: `init_reaction_mdata/reaxff/resources/para_deg`

Decide how many tasks will be run in parallel.

source_list:

type: `list`, optional, default: `[]`
argument path: `init_reaction_mdata/reaxff/resources/source_list`

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False
argument path: init_reaction_mdata/reaxff/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []
argument path: init_reaction_mdata/reaxff/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []
argument path: init_reaction_mdata/reaxff/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path: init_reaction_mdata/reaxff/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path: init_reaction_mdata/reaxff/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path: init_reaction_mdata/reaxff/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/reaxff/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/reaxff/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: init_reaction_mdata/reaxff/resources[Slurm]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/reaxff/resources[PBS]/kwargs

This field is empty for this batch.

When batch_type is set to Torque (or its alias torque):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/reaxff/resources[Torque]/kwargs

This field is empty for this batch.

When batch_type is set to SlurmJobArray (or its alias slurmjobarray):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/reaxff/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/reaxff/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional

argument path: init_reaction_mdata/reaxff/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict

argument path: init_reaction_mdata/reaxff/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False

argument path: init_reaction_mdata/reaxff/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

argument path: init_reaction_mdata/reaxff/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True

argument path: `init_reaction_mdata/reaxff/resources[LSF]/kwargs/gpu_exclusive`

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

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

argument path: `init_reaction_mdata/reaxff/resources[LSF]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #BSUB

When batch_type is set to `Lebesgue` (or its alias `lebesgue`):

kwargs:

type: `dict`, optional

argument path: `init_reaction_mdata/reaxff/resources[Lebesgue]/kwargs`

This field is empty for this batch.

When batch_type is set to `DistributedShell` (or its alias `distributedshell`):

kwargs:

type: `dict`, optional

argument path: `init_reaction_mdata/reaxff/resources[DistributedShell]/kwargs`

This field is empty for this batch.

user_forward_files:

type: `list`, optional

argument path: `init_reaction_mdata/reaxff/user_forward_files`

Files to be forwarded to the remote machine.

user_backward_files:

type: `list`, optional

argument path: `init_reaction_mdata/reaxff/user_backward_files`

Files to be backwarded from the remote machine.

build:

type: `dict`

argument path: `init_reaction_mdata/build`

Parameters of command, machine, and resources for build

command:

type: `str`
argument path: `init_reaction_mdata/build/command`

Command of a program.

machine:

type: `dict`
argument path: `init_reaction_mdata/build/machine`

batch_type:

type: `str`
argument path: `init_reaction_mdata/build/machine/batch_type`

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: `NoneType | str`
argument path: `init_reaction_mdata/build/machine/local_root`

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: `NoneType | str`, optional
argument path: `init_reaction_mdata/build/machine/remote_root`

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: `bool`, optional, default: `False`
argument path: `init_reaction_mdata/build/machine/clean_asynchronously`

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: `str` (flag key)
argument path: `init_reaction_mdata/build/machine/context_type`

possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to `LebesgueContext` (or its aliases `lebesguecontext`, `Lebesgue`, `lebesgue`):

remote_profile:

type: `dict`

argument path: `init_reaction_mdata/build/machine[LebesgueContext]/remote_profile`

The information used to maintain the connection with remote machine.

email:

type: `str`

argument path: `init_reaction_mdata/build/machine[LebesgueContext]/remote_profile/email`

Email

password:

type: `str`

argument path: `init_reaction_mdata/build/machine[LebesgueContext]/remote_profile/password`

Password

program_id:

type: `int`, alias: *project_id*

argument path: `init_reaction_mdata/build/machine[LebesgueContext]/remote_profile/program_id`

Program ID

keep_backup:

type: `bool`, optional

argument path: `init_reaction_mdata/build/machine[LebesgueContext]/remote_profile/keep_backup`

keep download and upload zip

input_data:

type: dict

argument path: init_reaction_mdata/build/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: init_reaction_mdata/build/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to `LocalContext` (or its aliases `localcontext`, `Local`, `local`):

remote_profile:

type: `dict`, optional

argument path: `init_reaction_mdata/build/machine[LocalContext]/remote_profile`

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to `DpCloudServerContext` (or its aliases `dpcloudservercontext`, `DpCloudServer`, `dpcloudserver`):

remote_profile:

type: `dict`

argument path: `init_reaction_mdata/build/machine[DpCloudServerContext]/remote_profile`

The information used to maintain the connection with remote machine.

email:

type: `str`

argument path: `init_reaction_mdata/build/machine[DpCloudServerContext]/remote_profile/email`

Email

password:

type: `str`

argument path: `init_reaction_mdata/build/machine[DpCloudServerContext]/remote_profile/password`

Password

program_id:

type: `int`, alias: *project_id*

argument path: `init_reaction_mdata/build/machine[DpCloudServerContext]/remote_profile/program_id`

Program ID

keep_backup:

type: `bool`, optional

argument path:
`init_reaction_mdata/build/machine[DpCloudServerContext]/remote_profile/keep_backup`

keep download and upload zip

input_data:

type: `dict`

argument path:
`init_reaction_mdata/build/machine[DpCloudServerContext]/remote_profile/input_data`

Configuration of job

When context_type is set to `LazyLocalContext` (or its aliases `lazylocalcontext`, `LazyLocal`, `lazylocal`):

remote_profile:

type: `dict`, optional

argument path: `init_reaction_mdata/build/machine[LazyLocalContext]/remote_profile`

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to `HDFSContext` (or its aliases `hdfscontext`, `HDFS`, `hdfs`):

remote_profile:

type: `dict`, optional

argument path:
`init_reaction_mdata/build/machine[HDFSContext]/remote_profile`

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: `dict`

argument path:
`init_reaction_mdata/build/resources`

number_node:

type: `int`, optional, default: 1

argument path: `init_reaction_mdata/build/resources/number_node`

The number of node need for each *job*

cpu_per_node:

type: `int`, optional, default: 1

argument path: `init_reaction_mdata/build/resources/cpu_per_node`

cpu numbers of each node assigned to each job.

gpu_per_node:

type: `int`, optional, default: `0`
argument path: `init_reaction_mdata/build/resources/gpu_per_node`

gpu numbers of each node assigned to each job.

queue_name:

type: `str`, optional, default: ````
argument path: `init_reaction_mdata/build/resources/queue_name`

The queue name of batch job scheduler system.

group_size:

type: `int`
argument path: `init_reaction_mdata/build/resources/group_size`

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: `list`, optional
argument path: `init_reaction_mdata/build/resources/custom_flags`

The extra lines pass to job submitting script header

strategy:

type: `dict`, optional
argument path: `init_reaction_mdata/build/resources/strategy`

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: `bool`, optional, default: `False`
argument path: `init_reaction_mdata/build/resources/strategy/if_cuda_multi_devices`

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be

used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

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

argument path: `init_reaction_mdata/build/resources/strategy/ratio_unfinished`

The ratio of *jobs* that can be unfinished.

para_deg:

type: `int`, optional, default: 1

argument path: `init_reaction_mdata/build/resources/para_deg`

Decide how many tasks will be run in parallel.

source_list:

type: `list`, optional, default: `[]`

argument path: `init_reaction_mdata/build/resources/source_list`

The env file to be sourced before the command execution.

module_purge:

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

argument path: `init_reaction_mdata/build/resources/module_purge`

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: `list`, optional, default: `[]`

argument path: `init_reaction_mdata/build/resources/module_unload_list`

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: `list`, optional, default: `[]`

argument path: `init_reaction_mdata/build/resources/module_list`

The modules to be loaded on HPC system before submitting jobs

envs:

type: `dict`, optional, default: `{}`

argument path: `init_reaction_mdata/build/resources/envs`

The environment variables to be exported on before submitting jobs

wait_time:

type: `int` | `float`, optional, default: `0`

argument path: `init_reaction_mdata/build/resources/wait_time`

The waitting time in second after a single *task* submitted

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

batch_type:

type: `str` (flag key)

argument path: `init_reaction_mdata/build/resources/batch_type`

possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to `Shell` (or its alias `shell`):

kwargs:

type: `dict`, optional

argument path: `init_reaction_mdata/build/resources[Shell]/kwargs`

This field is empty for this batch.

When batch_type is set to `Slurm` (or its alias `slurm`):

kwargs:

type: `dict`, optional

argument path: `init_reaction_mdata/build/resources[Slurm]/kwargs`

Extra arguments.

custom_gpu_line:

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

argument path: `init_reaction_mdata/build/resources[Slurm]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/build/resources[PBS]/kwargs

This field is empty for this batch.

When batch_type is set to Torque (or its alias torque):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/build/resources[Torque]/kwargs

This field is empty for this batch.

When batch_type is set to SlurmJobArray (or its alias slurmjobarray):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/build/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: init_reaction_mdata/build/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/build/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict

argument path: `init_reaction_mdata/build/resources[LSF]/kwargs`

Extra arguments.

gpu_usage:

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

argument path: `init_reaction_mdata/build/resources[LSF]/kwargs/gpu_usage`

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

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

argument path: `init_reaction_mdata/build/resources[LSF]/kwargs/gpu_new_syntax`

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

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

argument path: `init_reaction_mdata/build/resources[LSF]/kwargs/gpu_exclusive`

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

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

argument path: `init_reaction_mdata/build/resources[LSF]/kwargs/custom_gpu_line`

Custom GPU configuration, starting with #BSUB

When batch_type is set to `Lebesgue` (or its alias `lebesgue`):

kwargs:

type: `dict`, optional

argument path: `init_reaction_mdata/build/resources[Lebesgue]/kwargs`

This field is empty for this batch.

When batch_type is set to DistributedShell (or its alias distributedshell):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/build/resources[DistributedShell]/kwargs

This field is empty for this batch.

user_forward_files:

type: list, optional
argument path: init_reaction_mdata/build/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional
argument path: init_reaction_mdata/build/user_backward_files

Files to be backwarded from the remote machine.

fp:

type: dict
argument path: init_reaction_mdata/fp

Parameters of command, machine, and resources for fp

command:

type: str
argument path: init_reaction_mdata/fp/command

Command of a program.

machine:

type: dict
argument path: init_reaction_mdata/fp/machine

batch_type:

type: str
argument path: init_reaction_mdata/fp/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str

argument path: `init_reaction_mdata/fp/machine/local_root`

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: `NoneType | str`, optional

argument path: `init_reaction_mdata/fp/machine/remote_root`

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

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

argument path: `init_reaction_mdata/fp/machine/clean_asynchronously`

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: `str` (flag key)

argument path: `init_reaction_mdata/fp/machine/context_type`

possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to `LebesgueContext` (or its aliases `lebesguecontext`, `Lebesgue`, `lebesgue`):

remote_profile:

type: `dict`

argument path: `init_reaction_mdata/fp/machine[LebesgueContext]/remote_profile`

The information used to maintain the connection with remote machine.

email:

type: `str`

argument path:
init_reaction_mdata/fp/
machine[LebesgueContext]/
remote_profile/email

Email

password:

type: str
argument path:
init_reaction_mdata/fp/
machine[LebesgueContext]/
remote_profile/password

Password

program_id:

type: int, alias: *project_id*
argument path:
init_reaction_mdata/fp/
machine[LebesgueContext]/
remote_profile/program_id

Program ID

keep_backup:

type: bool, optional
argument path:
init_reaction_mdata/fp/
machine[LebesgueContext]/
remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict
argument path:
init_reaction_mdata/fp/
machine[LebesgueContext]/
remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict
argument path: init_reaction_mdata/fp/
machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path:
`init_reaction_mdata/fp/machine[SSHContext]/remote_profile/hostname`

hostname or ip of ssh connection.

username:

type: `str`

argument path:
`init_reaction_mdata/fp/machine[SSHContext]/remote_profile/username`

username of target linux system

password:

type: `str`, optional

argument path:
`init_reaction_mdata/fp/machine[SSHContext]/remote_profile/password`

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: `int`, optional, default: `22`

argument path:
`init_reaction_mdata/fp/machine[SSHContext]/remote_profile/port`

ssh connection port.

key_filename:

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

argument path:
`init_reaction_mdata/fp/machine[SSHContext]/remote_profile/key_filename`

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

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

argument path:
`init_reaction_mdata/fp/machine[SSHContext]/remote_profile/passphrase`

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10
argument path: init_reaction_mdata/fp/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None
argument path: init_reaction_mdata/fp/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True
argument path: init_reaction_mdata/fp/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional
argument path: init_reaction_mdata/fp/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict
argument path: init_reaction_mdata/fp/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: init_reaction_mdata/fp/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: init_reaction_mdata/fp/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: init_reaction_mdata/fp/machine[DpCloudServerContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: init_reaction_mdata/fp/machine[DpCloudServerContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: init_reaction_mdata/fp/machine[DpCloudServerContext]/remote_profile/input_data

Configuration of job

When context_type is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

remote_profile:

type: dict, optional

argument path: init_reaction_mdata/fp/machine[LazyLocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

remote_profile:

type: dict, optional
argument path: init_reaction_mdata/fp/machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict
argument path: init_reaction_mdata/fp/resources

number_node:

type: int, optional, default: 1
argument path: init_reaction_mdata/fp/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1
argument path: init_reaction_mdata/fp/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0
argument path: init_reaction_mdata/fp/resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str, optional, default: ````
argument path: init_reaction_mdata/fp/resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int
argument path: init_reaction_mdata/fp/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional

argument path: `init_reaction_mdata/fp/resources/custom_flags`

The extra lines pass to job submitting script header

strategy:

type: `dict`, optional
argument path: `init_reaction_mdata/fp/resources/strategy`

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: `bool`, optional, default: `False`
argument path: `init_reaction_mdata/fp/resources/strategy/if_cuda_multi_devices`

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: `float`, optional, default: `0.0`
argument path: `init_reaction_mdata/fp/resources/strategy/ratio_unfinished`

The ratio of *jobs* that can be unfinished.

para_deg:

type: `int`, optional, default: 1
argument path: `init_reaction_mdata/fp/resources/para_deg`

Decide how many tasks will be run in parallel.

source_list:

type: `list`, optional, default: `[]`
argument path: `init_reaction_mdata/fp/resources/source_list`

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False
argument path: init_reaction_mdata/fp/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []
argument path: init_reaction_mdata/fp/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []
argument path: init_reaction_mdata/fp/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path: init_reaction_mdata/fp/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path: init_reaction_mdata/fp/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path: init_reaction_mdata/fp/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/fp/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/fp/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: init_reaction_mdata/fp/resources[Slurm]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/fp/resources[PBS]/kwargs

This field is empty for this batch.

When batch_type is set to Torque (or its alias torque):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/fp/resources[Torque]/kwargs

This field is empty for this batch.

When batch_type is set to SlurmJobArray (or its alias slurmjobarray):

kwargs:

type: dict, optional
argument path: init_reaction_mdata/fp/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/fp/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional

argument path: init_reaction_mdata/fp/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict

argument path: init_reaction_mdata/fp/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False

argument path: init_reaction_mdata/fp/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

argument path: init_reaction_mdata/fp/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True

argument path: init_reaction_mdata/fp/resources[LSF]/kwargs/gpu_exclusive

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: init_reaction_mdata/fp/resources[LSF]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #BSUB

When batch_type is set to Lebesgue (or its alias lebesgue):

kwargs:

type: dict, optional

argument path: init_reaction_mdata/fp/resources[Lebesgue]/kwargs

This field is empty for this batch.

When batch_type is set to DistributedShell (or its alias distributedshell):

kwargs:

type: dict, optional

argument path: init_reaction_mdata/fp/resources[DistributedShell]/kwargs

This field is empty for this batch.

user_forward_files:

type: list, optional

argument path: init_reaction_mdata/fp/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional

argument path: init_reaction_mdata/fp/user_backward_files

Files to be backwarded from the remote machine.

CHAPTER

SIX

SIMPLIFY

6.1 Simplify

When you have a dataset containing lots of repeated data, this step will help you simplify your dataset. The workflow contains three stages: train, model_devi, and fp. The train stage and the fp stage are as the same as the run step, and the model_devi stage will calculate model deviations of the rest data that has not been confirmed accurate. Data with small model deviations will be confirmed accurate, while the program will pick data from those with large model deviations to the new dataset.

Use the following script to start the workflow:

```
dpgen simplify param.json machine.json
```

Here is an example of `param.json` for QM7 dataset:

```
{
    "type_map": [
        "C",
        "H",
        "N",
        "O",
        "S"
    ],
    "mass_map": [
        12.011,
        1.008,
        14.007,
        15.999,
        32.065
    ],
    "pick_data": "/scratch/jz748/simplify/qm7",
    "init_data_prefix": "",
    "init_data_sys": [],
    "sys_batch_size": [
        "auto"
    ],
    "numb_models": 4,
    "default_training_param": {
        "model": {
            "type_map": [
                "C",
```

(continues on next page)

(continued from previous page)

```
                "H",
                "N",
                "O",
                "S"
            ],
            "descriptor": {
                "type": "se_a",
                "sel": [
                    7,
                    16,
                    3,
                    3,
                    1
                ],
                "rcut_smth": 1.00,
                "rcut": 6.00,
                "neuron": [
                    25,
                    50,
                    100
                ],
                "resnet_dt": false,
                "axis_neuron": 12
            },
            "fitting_net": {
                "neuron": [
                    240,
                    240,
                    240
                ],
                "resnet_dt": true
            }
        },
        "learning_rate": {
            "type": "exp",
            "start_lr": 0.001,
            "decay_steps": 10,
            "decay_rate": 0.99
        },
        "loss": {
            "start_pref_e": 0.02,
            "limit_pref_e": 1,
            "start_pref_f": 1000,
            "limit_pref_f": 1,
            "start_pref_v": 0,
            "limit_pref_v": 0,
            "start_pref_pf": 0,
            "limit_pref_pf": 0
        },
        "training": {
            "set_prefix": "set",
            "stop_batch": 10000,
```

(continues on next page)

(continued from previous page)

```
            "disp_file": "lcurve.out",
            "disp_freq": 1000,
            "numb_test": 1,
            "save_freq": 1000,
            "save_ckpt": "model.ckpt",
            "disp_training": true,
            "time_training": true,
            "profiling": false,
            "profiling_file": "timeline.json"
        },
        "_comment": "that's all"
    },
    "fp_style": "gaussian",
    "shuffle_poscar": false,
    "fp_task_max": 1000,
    "fp_task_min": 10,
    "fp_pp_path": "/home/jzzeng/",
    "fp_pp_files": [],
    "fp_params": {
        "keywords": "mn15/6-31g** force nosymm scf(maxcyc=512)",
        "nproc": 28,
        "multiplicity": 1,
        "_comment": " that's all "
    },
    "init_pick_number":100,
    "iter_pick_number":100,
    "f_trust_lo":0.25,
    "f_trust_hi":0.45,
    "_comment": " that's all "
}
```

Here `pick_data` is the directory to data to simplify where the program recursively detects systems `System` with `deepmd/npy` format. `init_pick_number` and `iter_pick_number` are the numbers of picked frames. `e_trust_lo`, `e_trust_hi` mean the range of the deviation of the frame energy, and `f_trust_lo` and `f_trust_hi` mean the range of the max deviation of atomic forces in a frame. `fp_style` can only be `gaussian` currently. Other parameters are as the same as those of generator.

6.2 dpgen simplify parameters

`simplify_jdata:`

type: `dict`

argument path: `simplify_jdata`

Parameters for simplify.json, the first argument of *dpgen simplify*.

`type_map:`

type: `list`

argument path: `simplify_jdata/type_map`

Atom types.

mass_map:

type: list | str, optional, default: auto

argument path: simplify_jdata/mass_map

Standard atomic weights (default: "auto"). if one want to use isotopes, or non-standard element names, chemical symbols, or atomic number in the type_map list, please customize the mass_map list instead of using "auto". Tips: at present the default value will not be applied automatically, so you need to set "mass_map" manually in param.json.

use_ele_temp:

type: int, optional, default: 0

argument path: simplify_jdata/use_ele_temp

Currently only support fp_style vasp.

- 0: no electron temperature.
- 1: eletron temperature as frame parameter.
- 2: electron temperature as atom parameter.

init_data_prefix:

type: str, optional

argument path: simplify_jdata/init_data_prefix

Prefix of initial data directories.

init_data_sys:

type: list

argument path: simplify_jdata/init_data_sys

Directories of initial data. You may use either absolute or relative path here. Systems will be detected recursively in the directories.

sys_format:

type: str, optional, default: vasp/poscar

argument path: simplify_jdata/sys_format

Format of initial data.

init_batch_size:

type: list | str, optional

argument path: simplify_jdata/init_batch_size

Each number is the batch_size of corresponding system for training in init_data_sys. One recommended rule for setting the sys_batch_size and init_batch_size is that batch_size mutiply number of atoms ot the stucture should be larger than 32. If set to auto, batch size will be 32 divided by number of atoms.

sys_configs_prefix:

type: str, optional

argument path: simplify_jdata/sys_configs_prefix

Prefix of sys_configs.

sys_configs:

type: list

argument path: simplify_jdata/sys_configs

Containing directories of structures to be explored in iterations.Wildcard characters are supported here.

sys_batch_size:

type: list, optional

argument path: simplify_jdata/sys_batch_size

Each number is the batch_size for training of corresponding system in sys_configs. If set to auto, batch size will be 32 divided by number of atoms.

labeled:

type: bool, optional, default: False

argument path: simplify_jdata/labeled

If true, the initial data is labeled.

pick_data:

type: str

argument path: simplify_jdata/pick_data

Path to the directory with the pick data with the deepmd/npy format. Systems are detected recursively.

init_pick_number:

type: int

argument path: simplify_jdata/init_pick_number

The number of initial pick data.

iter_pick_number:

type: int

argument path: simplify_jdata/iter_pick_number

The number of pick data in each iteration.

model_devi_f_trust_lo:

type: float

argument path: simplify_jdata/model_devi_f_trust_lo

The lower bound of forces for the selection for the model deviation.

model_devi_f_trust_hi:

type: float

argument path: simplify_jdata/model_devi_f_trust_hi

The higher bound of forces for the selection for the model deviation.

numb_models:

type: int

argument path: simplify_jdata/numb_models

Number of models to be trained in 00.train. 4 is recommend.

training_iter0_model_path:

type: list, optional

argument path: simplify_jdata/training_iter0_model_path

The model used to init the first iter training. Number of element should be equal to numb_models.

training_init_model:

type: bool, optional

argument path: simplify_jdata/training_init_model

Iteration > 0, the model parameters will be initilized from the model trained at the previous iteration. Iteration == 0, the model parameters will be initialized from training_iter0_model_path.

default_training_param:

type: dict

argument path: simplify_jdata/default_training_param

Training parameters for deepmd-kit in 00.train. You can find instructions from here: (https://github.com/deepmodeling/deepmd-kit).

dp_compress:

type: bool, optional, default: False

argument path: simplify_jdata/dp_compress

Use dp compress to compress the model.

training_reuse_iter:

type: int | NoneType, optional

argument path: simplify_jdata/training_reuse_iter

The minimal index of iteration that continues training models from old models of last iteration.

training_reuse_old_ratio:

type: NoneType | float, optional

argument path: simplify_jdata/training_reuse_old_ratio

The probability proportion of old data during training. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_numb_steps:

type: int | NoneType, optional, default: 400000, alias: *training_reuse_stop_batch*

argument path: simplify_jdata/training_reuse_numb_steps

Number of training batch. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_start_lr:

type: NoneType | float, optional, default: 0.0001

argument path: simplify_jdata/training_reuse_start_lr

The learning rate the start of the training. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_start_pref_e:

type: int | NoneType | float, optional, default: 0.1

argument path: simplify_jdata/training_reuse_start_pref_e

The prefactor of energy loss at the start of the training. This option is only adopted when continuing training models from old models. This option will override default parameters.

training_reuse_start_pref_f:

type: int | NoneType | float, optional, default: 100

argument path: simplify_jdata/training_reuse_start_pref_f

The prefactor of force loss at the start of the training. This option is only adopted when continuing training models from old models. This option will override default parameters.

model_devi_activation_func:

type: list | NoneType, optional

argument path: simplify_jdata/model_devi_activation_func

The activation function in the model. The shape of list should be (N_models, 2), where 2 represents the embedding and fitting network. This option will override default parameters.

fp_task_max:

type: int, optional

argument path: simplify_jdata/fp_task_max

Maximum of structures to be calculated in 02.fp of each iteration.

fp_task_min:

type: int, optional

argument path: simplify_jdata/fp_task_min

Minimum of structures to be calculated in 02.fp of each iteration.

fp_accurate_threshold:

type: float, optional

argument path: simplify_jdata/fp_accurate_threshold

If the accurate ratio is larger than this number, no fp calculation will be performed, i.e. fp_task_max = 0.

fp_accurate_soft_threshold:

type: float, optional

argument path: simplify_jdata/fp_accurate_soft_threshold

If the accurate ratio is between this number and fp_accurate_threshold, the fp_task_max linearly decays to zero.

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

fp_style:

type: str (flag key), default: none

argument path: simplify_jdata/fp_style

possible choices: *none*, *vasp*, *gaussian*

Software for First Principles, if *labeled* is false. Options include "vasp", "gaussian" up to now.

When fp_style is set to none:

No fp.

When fp_style is set to vasp:

VASP.

fp_pp_path:

type: str

argument path: simplify_jdata[vasp]/fp_pp_path

Directory of psuedo-potential file to be used for 02.fp exists.

fp_pp_files:

type: list

argument path: simplify_jdata[vasp]/fp_pp_files

Psuedo-potential file to be used for 02.fp. Note that the order of elements should correspond to the order in type_map.

fp_incar:

type: str

argument path: simplify_jdata[vasp]/fp_incar

Input file for VASP. INCAR must specify KSPACING and KGAMMA.

fp_aniso_kspacing:

type: list, optional

argument path: simplify_jdata[vasp]/fp_aniso_kspacing

Set anisotropic kspacing. Usually useful for 1-D or 2-D materials. Only support VASP. If it is setting the KSPACING key in INCAR will be ignored.

cvasp:

type: bool, optional

argument path: simplify_jdata[vasp]/cvasp

If cvasp is true, DP-GEN will use Custodian to help control VASP calculation.

ratio_failed:

type: float, optional

argument path: simplify_jdata[vasp]/ratio_failed

Check the ratio of unsuccessfully terminated jobs. If too many FP tasks are not converged, RuntimeError will be raised.

fp_skip_bad_box:

type: str, optional

argument path: simplify_jdata[vasp]/fp_skip_bad_box

Skip the configurations that are obviously unreasonable before 02.fp

When fp_style is set to gaussian:

Gaussian. The command should be set as *g16 < input*.

use_clusters:

type: bool, optional, default: False

argument path: simplify_jdata[gaussian]/use_clusters

If set to true, clusters will be taken instead of the whole system.

cluster_cutoff:

type: float, optional

argument path: simplify_jdata[gaussian]/cluster_cutoff

The soft cutoff radius of clusters if *use_clusters* is set to true. Molecules will be taken as whole even if part of atoms is out of the cluster. Use *cluster_cutoff_hard* to only take atoms within the hard cutoff radius.

cluster_cutoff_hard:

type: float, optional
argument path: simplify_jdata[gaussian]/cluster_cutoff_hard

The hard cutoff radius of clusters if *use_clusters* is set to true. Outside the hard cutoff radius, atoms will not be taken even if they are in a molecule where some atoms are within the cutoff radius.

cluster_minify:

type: bool, optional, default: False
argument path: simplify_jdata[gaussian]/cluster_minify

If enabled, when an atom within the soft cutoff radius connects a single bond with a non-hydrogen atom out of the soft cutoff radius, the outer atom will be replaced by a hydrogen atom. When the outer atom is a hydrogen atom, the outer atom will be kept. In this case, other atoms out of the soft cutoff radius will be removed.

fp_params:

type: dict
argument path: simplify_jdata[gaussian]/fp_params

Parameters for Gaussian calculation.

keywords:

type: list | str
argument path: simplify_jdata[gaussian]/fp_params/keywords

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

multiplicity:

type: int | str, optional, default: auto
argument path: simplify_jdata[gaussian]/fp_params/multiplicity

Spin multiplicity for Gaussian input. If *auto*, multiplicity will be detected automatically, with the following rules: when fragment_guesses=True, multiplicity will +1 for each radical, and +2 for each oxygen molecule; when fragment_guesses=False, multiplicity will be 1 or 2, but +2 for each oxygen molecule.

nproc:

type: int
argument path: simplify_jdata[gaussian]/fp_params/nproc

The number of processors for Gaussian input.

charge:

type: int, optional, default: 0
argument path: simplify_jdata[gaussian]/fp_params/charge

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

fragment_guesses:

type: bool, optional, default: False
argument path: simplify_jdata[gaussian]/fp_params/fragment_guesses

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

basis_set:

type: str, optional
argument path: simplify_jdata[gaussian]/fp_params/basis_set

Custom basis set.

keywords_high_multiplicity:

type: str, optional
argument path: simplify_jdata[gaussian]/fp_params/keywords_high_multiplicity

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

ratio_failed:

type: float, optional
argument path: simplify_jdata[gaussian]/ratio_failed

Check the ratio of unsuccessfully terminated jobs. If too many FP tasks are not converged, RuntimeError will be raised.

6.3 dpgen simplify machine parameters

simplify_mdata:

type: dict
argument path: simplify_mdata

machine.json file

api_version:

type: str
argument path: simplify_mdata/api_version

Please set to 1.0

deepmd_version:

type: str, optional, default: 2
argument path: simplify_mdata/deepmd_version

DeePMD-kit version, e.g. 2.1.3

train:

type: dict
argument path: simplify_mdata/train

Parameters of command, machine, and resources for train

command:

type: str
argument path: simplify_mdata/train/command

Command of a program.

machine:

type: dict
argument path: simplify_mdata/train/machine

batch_type:

type: str
argument path: simplify_mdata/train/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: simplify_mdata/train/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional
argument path: simplify_mdata/train/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False
argument path: simplify_mdata/train/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)

argument path: simplify_mdata/train/machine/context_type

possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict

argument path: simplify_mdata/train/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: simplify_mdata/train/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str

argument path: simplify_mdata/train/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: simplify_mdata/train/machine[LebesgueContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: simplify_mdata/train/machine[LebesgueContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: simplify_mdata/train/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: simplify_mdata/train/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: simplify_mdata/train/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: simplify_mdata/train/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: simplify_mdata/train/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: simplify_mdata/train/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: simplify_mdata/train/machine[DpCloudServerContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: simplify_mdata/train/machine[DpCloudServerContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: simplify_mdata/train/machine[DpCloudServerContext]/remote_profile/input_data

Configuration of job

When context_type is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

remote_profile:

type: dict, optional

argument path: simplify_mdata/train/machine[LazyLocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

remote_profile:

type: dict, optional

argument path: simplify_mdata/train/machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict

argument path: simplify_mdata/train/resources

number_node:

type: int, optional, default: 1

argument path: simplify_mdata/train/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1

argument path: simplify_mdata/train/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0
argument path: simplify_mdata/train/resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str, optional, default: ````
argument path: simplify_mdata/train/resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int
argument path: simplify_mdata/train/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional
argument path: simplify_mdata/train/resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional
argument path: simplify_mdata/train/resources/strategy

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False
argument path: simplify_mdata/train/resources/strategy/if_cuda_multi_devices

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: float, optional, default: 0.0

argument path: simplify_mdata/train/resources/strategy/ratio_unfinished

The ratio of *jobs* that can be unfinished.

para_deg:

type: int, optional, default: 1

argument path: simplify_mdata/train/resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: list, optional, default: []

argument path: simplify_mdata/train/resources/source_list

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False

argument path: simplify_mdata/train/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []

argument path: simplify_mdata/train/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []

argument path: simplify_mdata/train/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}

argument path: simplify_mdata/train/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path: simplify_mdata/train/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path: simplify_mdata/train/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional
argument path: simplify_mdata/train/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: simplify_mdata/train/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: simplify_mdata/train/resources[Slurm]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional

argument path: simplify_mdata/train/resources[PBS]/kwargs

This field is empty for this batch.

When batch_type is set to Torque (or its alias torque):

kwargs:

type: dict, optional
argument path: simplify_mdata/train/resources[Torque]/kwargs

This field is empty for this batch.

When batch_type is set to SlurmJobArray (or its alias slurmjobarray):

kwargs:

type: dict, optional
argument path: simplify_mdata/train/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: simplify_mdata/train/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional
argument path: simplify_mdata/train/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict
argument path: simplify_mdata/train/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False

argument path: simplify_mdata/train/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

argument path: simplify_mdata/train/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True

argument path: simplify_mdata/train/resources[LSF]/kwargs/gpu_exclusive

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/train/resources[LSF]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #BSUB

When batch_type is set to Lebesgue (or its alias lebesgue):

kwargs:

type: dict, optional

argument path: simplify_mdata/train/resources[Lebesgue]/kwargs

This field is empty for this batch.

When batch_type is set to DistributedShell (or its alias distributedshell):

kwargs:

type: dict, optional

argument path: simplify_mdata/train/resources[DistributedShell]/kwargs

This field is empty for this batch.

user_forward_files:

type: list, optional
argument path: simplify_mdata/train/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional
argument path: simplify_mdata/train/user_backward_files

Files to be backwarded from the remote machine.

model_devi:

type: dict
argument path: simplify_mdata/model_devi

Parameters of command, machine, and resources for model_devi

command:

type: str
argument path: simplify_mdata/model_devi/command

Command of a program.

machine:

type: dict
argument path: simplify_mdata/model_devi/machine

batch_type:

type: str
argument path: simplify_mdata/model_devi/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: simplify_mdata/model_devi/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional
argument path: simplify_mdata/model_devi/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False
argument path: simplify_mdata/model_devi/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)
argument path: simplify_mdata/model_devi/machine/context_type
possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict
argument path: simplify_mdata/model_devi/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str
argument path: simplify_mdata/model_devi/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str
argument path: simplify_mdata/model_devi/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: simplify_mdata/model_devi/machine[LebesgueContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: simplify_mdata/model_devi/machine[LebesgueContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: simplify_mdata/model_devi/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: simplify_mdata/model_devi/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: simplify_mdata/model_devi/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: simplify_mdata/model_devi/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: simplify_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: simplify_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: simplify_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: simplify_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: simplify_mdata/model_devi/machine[DpCloudServerContext]/remote_profile/input_data

Configuration of job

When context_type is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

remote_profile:

type: dict, optional

argument path: simplify_mdata/model_devi/machine[LazyLocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

remote_profile:

type: dict, optional

argument path:
simplify_mdata/model_devi/
machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict
argument path:
simplify_mdata/model_devi/resources

number_node:

type: int, optional, default: 1
argument path: simplify_mdata/
model_devi/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1
argument path: simplify_mdata/
model_devi/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0
argument path: simplify_mdata/
model_devi/resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str, optional, default: ````
argument path: simplify_mdata/
model_devi/resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int
argument path: simplify_mdata/
model_devi/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional
argument path: simplify_mdata/
model_devi/resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional
argument path: simplify_mdata/model_devi/resources/strategy

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False
argument path: simplify_mdata/model_devi/resources/strategy/if_cuda_multi_devices

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: float, optional, default: 0.0
argument path: simplify_mdata/model_devi/resources/strategy/ratio_unfinished

The ratio of *jobs* that can be unfinished.

para_deg:

type: int, optional, default: 1
argument path: simplify_mdata/model_devi/resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: list, optional, default: []
argument path: simplify_mdata/model_devi/resources/source_list

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False
argument path: simplify_mdata/model_devi/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []
argument path: simplify_mdata/model_devi/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []
argument path: simplify_mdata/model_devi/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}
argument path: simplify_mdata/model_devi/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0
argument path: simplify_mdata/model_devi/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)
argument path: simplify_mdata/model_devi/resources/batch_type
possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional

argument path: simplify_mdata/model_devi/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: simplify_mdata/model_devi/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: simplify_mdata/model_devi/resources[Slurm]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional
argument path: simplify_mdata/model_devi/resources[PBS]/kwargs

This field is empty for this batch.

When batch_type is set to Torque (or its alias torque):

kwargs:

type: dict, optional
argument path: simplify_mdata/model_devi/resources[Torque]/kwargs

This field is empty for this batch.

When batch_type is set to SlurmJobArray (or its alias slurmjobarray):

kwargs:

type: dict, optional
argument path: simplify_mdata/model_devi/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/model_devi/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional

argument path: simplify_mdata/model_devi/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict

argument path: simplify_mdata/model_devi/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False

argument path: simplify_mdata/model_devi/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

argument path: simplify_mdata/model_devi/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True

argument path: simplify_mdata/model_devi/resources[LSF]/kwargs/gpu_exclusive

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: simplify_mdata/model_devi/resources[LSF]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #BSUB

When batch_type is set to Lebesgue (or its alias lebesgue):

kwargs:

type: dict, optional
argument path: simplify_mdata/model_devi/resources[Lebesgue]/kwargs

This field is empty for this batch.

When batch_type is set to DistributedShell (or its alias distributedshell):

kwargs:

type: dict, optional
argument path: simplify_mdata/model_devi/resources[DistributedShell]/kwargs

This field is empty for this batch.

user_forward_files:

type: list, optional
argument path: simplify_mdata/model_devi/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional
argument path: simplify_mdata/model_devi/user_backward_files

Files to be backwarded from the remote machine.

fp:

type: dict
argument path: simplify_mdata/fp

Parameters of command, machine, and resources for fp

command:

type: str
argument path: simplify_mdata/fp/command

Command of a program.

machine:

type: dict
argument path: simplify_mdata/fp/machine

batch_type:

type: str
argument path: simplify_mdata/fp/machine/batch_type

The batch job system type. Option: PBS, Shell, LSF, Lebesgue, Slurm, Torque, SlurmJobArray, DistributedShell, DpCloudServer

local_root:

type: NoneType | str
argument path: simplify_mdata/fp/machine/local_root

The dir where the tasks and relating files locate. Typically the project dir.

remote_root:

type: NoneType | str, optional
argument path: simplify_mdata/fp/machine/remote_root

The dir where the tasks are executed on the remote machine. Only needed when context is not lazy-local.

clean_asynchronously:

type: bool, optional, default: False
argument path: simplify_mdata/fp/machine/clean_asynchronously

Clean the remote directory asynchronously after the job finishes.

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

context_type:

type: str (flag key)
argument path: simplify_mdata/fp/machine/context_type
possible choices: *LebesgueContext*, *SSHContext*, *LocalContext*, *DpCloudServerContext*, *LazyLocalContext*, *HDFSContext*

The connection used to remote machine. Option: SSHContext, LazyLocalContext, LebesgueContext, LocalContext, DpCloudServerContext, HDFSContext

When context_type is set to LebesgueContext (or its aliases lebesguecontext, Lebesgue, lebesgue):

remote_profile:

type: dict

argument path: simplify_mdata/fp/machine[LebesgueContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: simplify_mdata/fp/machine[LebesgueContext]/remote_profile/email

Email

password:

type: str

argument path: simplify_mdata/fp/machine[LebesgueContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: simplify_mdata/fp/machine[LebesgueContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: simplify_mdata/fp/machine[LebesgueContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: simplify_mdata/fp/machine[LebesgueContext]/remote_profile/input_data

Configuration of job

When context_type is set to SSHContext (or its aliases sshcontext, SSH, ssh):

remote_profile:

type: dict

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile

The information used to maintain the connection with remote machine.

hostname:

type: str

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/hostname

hostname or ip of ssh connection.

username:

type: str

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/username

username of target linux system

password:

type: str, optional

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/password

(deprecated) password of linux system. Please use SSH keys instead to improve security.

port:

type: int, optional, default: 22

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/port

ssh connection port.

key_filename:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/key_filename

key filename used by ssh connection. If left None, find key in ~/.ssh or use password for login

passphrase:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/passphrase

passphrase of key used by ssh connection

timeout:

type: int, optional, default: 10

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/timeout

timeout of ssh connection

totp_secret:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/totp_secret

Time-based one time password secret. It should be a base32-encoded string extracted from the 2D code.

tar_compress:

type: bool, optional, default: True

argument path: simplify_mdata/fp/machine[SSHContext]/remote_profile/tar_compress

The archive will be compressed in upload and download if it is True. If not, compression will be skipped.

When context_type is set to LocalContext (or its aliases localcontext, Local, local):

remote_profile:

type: dict, optional

argument path: simplify_mdata/fp/machine[LocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to DpCloudServerContext (or its aliases dpcloudservercontext, DpCloudServer, dpcloudserver):

remote_profile:

type: dict

argument path: simplify_mdata/fp/machine[DpCloudServerContext]/remote_profile

The information used to maintain the connection with remote machine.

email:

type: str

argument path: simplify_mdata/fp/machine[DpCloudServerContext]/remote_profile/email

Email

password:

type: str

argument path: simplify_mdata/fp/machine[DpCloudServerContext]/remote_profile/password

Password

program_id:

type: int, alias: *project_id*

argument path: simplify_mdata/fp/machine[DpCloudServerContext]/remote_profile/program_id

Program ID

keep_backup:

type: bool, optional

argument path: simplify_mdata/fp/machine[DpCloudServerContext]/remote_profile/keep_backup

keep download and upload zip

input_data:

type: dict

argument path: simplify_mdata/fp/machine[DpCloudServerContext]/remote_profile/input_data

Configuration of job

When context_type is set to LazyLocalContext (or its aliases lazylocalcontext, LazyLocal, lazylocal):

remote_profile:

type: dict, optional

argument path: simplify_mdata/fp/machine[LazyLocalContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

When context_type is set to HDFSContext (or its aliases hdfscontext, HDFS, hdfs):

remote_profile:

type: dict, optional
argument path: simplify_mdata/fp/machine[HDFSContext]/remote_profile

The information used to maintain the connection with remote machine. This field is empty for this context.

resources:

type: dict
argument path: simplify_mdata/fp/resources

number_node:

type: int, optional, default: 1
argument path: simplify_mdata/fp/resources/number_node

The number of node need for each *job*

cpu_per_node:

type: int, optional, default: 1
argument path: simplify_mdata/fp/resources/cpu_per_node

cpu numbers of each node assigned to each job.

gpu_per_node:

type: int, optional, default: 0
argument path: simplify_mdata/fp/resources/gpu_per_node

gpu numbers of each node assigned to each job.

queue_name:

type: str, optional, default: ````
argument path: simplify_mdata/fp/resources/queue_name

The queue name of batch job scheduler system.

group_size:

type: int
argument path: simplify_mdata/fp/resources/group_size

The number of *tasks* in a *job*. 0 means infinity.

custom_flags:

type: list, optional

argument path: simplify_mdata/fp/resources/custom_flags

The extra lines pass to job submitting script header

strategy:

type: dict, optional
argument path: simplify_mdata/fp/resources/strategy

strategies we use to generation job submitting scripts.

if_cuda_multi_devices:

type: bool, optional, default: False
argument path: simplify_mdata/fp/resources/strategy/if_cuda_multi_devices

If there are multiple nvidia GPUS on the node, and we want to assign the tasks to different GPUS.If true, dpdispatcher will manually export environment variable CUDA_VISIBLE_DEVICES to different task.Usually, this option will be used with Task.task_need_resources variable simultaneously.

ratio_unfinished:

type: float, optional, default: 0.0
argument path: simplify_mdata/fp/resources/strategy/ratio_unfinished

The ratio of *jobs* that can be unfinished.

para_deg:

type: int, optional, default: 1
argument path: simplify_mdata/fp/resources/para_deg

Decide how many tasks will be run in parallel.

source_list:

type: list, optional, default: []
argument path: simplify_mdata/fp/resources/source_list

The env file to be sourced before the command execution.

module_purge:

type: bool, optional, default: False

argument path: simplify_mdata/fp/resources/module_purge

Remove all modules on HPC system before module load (module_list)

module_unload_list:

type: list, optional, default: []

argument path: simplify_mdata/fp/resources/module_unload_list

The modules to be unloaded on HPC system before submitting jobs

module_list:

type: list, optional, default: []

argument path: simplify_mdata/fp/resources/module_list

The modules to be loaded on HPC system before submitting jobs

envs:

type: dict, optional, default: {}

argument path: simplify_mdata/fp/resources/envs

The environment variables to be exported on before submitting jobs

wait_time:

type: int | float, optional, default: 0

argument path: simplify_mdata/fp/resources/wait_time

The waitting time in second after a single *task* submitted

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

batch_type:

type: str (flag key)

argument path: simplify_mdata/fp/resources/batch_type

possible choices: *Shell*, *Slurm*, *PBS*, *Torque*, *SlurmJobArray*, *DpCloudServer*, *LSF*, *Lebesgue*, *DistributedShell*

The batch job system type loaded from machine/batch_type.

When batch_type is set to Shell (or its alias shell):

kwargs:

type: dict, optional
argument path: simplify_mdata/fp/resources[Shell]/kwargs

This field is empty for this batch.

When batch_type is set to Slurm (or its alias slurm):

kwargs:

type: dict, optional
argument path: simplify_mdata/fp/resources[Slurm]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None
argument path: simplify_mdata/fp/resources[Slurm]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to PBS (or its alias pbs):

kwargs:

type: dict, optional
argument path: simplify_mdata/fp/resources[PBS]/kwargs

This field is empty for this batch.

When batch_type is set to Torque (or its alias torque):

kwargs:

type: dict, optional
argument path: simplify_mdata/fp/resources[Torque]/kwargs

This field is empty for this batch.

When batch_type is set to SlurmJobArray (or its alias slurmjobarray):

kwargs:

type: dict, optional
argument path: simplify_mdata/fp/resources[SlurmJobArray]/kwargs

Extra arguments.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/fp/resources[SlurmJobArray]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #SBATCH

When batch_type is set to DpCloudServer (or its alias dpcloudserver):

kwargs:

type: dict, optional

argument path: simplify_mdata/fp/resources[DpCloudServer]/kwargs

This field is empty for this batch.

When batch_type is set to LSF (or its alias lsf):

kwargs:

type: dict

argument path: simplify_mdata/fp/resources[LSF]/kwargs

Extra arguments.

gpu_usage:

type: bool, optional, default: False

argument path: simplify_mdata/fp/resources[LSF]/kwargs/gpu_usage

Choosing if GPU is used in the calculation step.

gpu_new_syntax:

type: bool, optional, default: False

argument path: simplify_mdata/fp/resources[LSF]/kwargs/gpu_new_syntax

For LFS >= 10.1.0.3, new option -gpu for #BSUB could be used. If False, and old syntax would be used.

gpu_exclusive:

type: bool, optional, default: True

argument path: simplify_mdata/fp/resources[LSF]/kwargs/gpu_exclusive

Only take effect when new syntax enabled. Control whether submit tasks in exclusive way for GPU.

custom_gpu_line:

type: NoneType | str, optional, default: None

argument path: simplify_mdata/fp/resources[LSF]/kwargs/custom_gpu_line

Custom GPU configuration, starting with #BSUB

When batch_type is set to Lebesgue (or its alias lebesgue):

kwargs:

type: dict, optional

argument path: simplify_mdata/fp/resources[Lebesgue]/kwargs

This field is empty for this batch.

When batch_type is set to DistributedShell (or its alias distributedshell):

kwargs:

type: dict, optional

argument path: simplify_mdata/fp/resources[DistributedShell]/kwargs

This field is empty for this batch.

user_forward_files:

type: list, optional

argument path: simplify_mdata/fp/user_forward_files

Files to be forwarded to the remote machine.

user_backward_files:

type: list, optional

argument path: simplify_mdata/fp/user_backward_files

Files to be backwarded from the remote machine.

CHAPTER

SEVEN

AUTO TEST

7.1 Autotest Overview: Autotest for Deep Generator

Suppose that we have a potential (can be DFT, DP, MEAM . . .), `autotest` helps us automatically calculate M properties on N configurations. The folder where the `autotest` runs is called the working directory of `autotest`. Different potentials should be tested in different working directories.

A property is tested in three steps: `make`, `run` and `post`. `make` prepares all computational tasks that are needed to calculate the property. For example to calculate EOS, `make` prepares a series of tasks, each of which has a scaled configuration with certain volume, and all necessary input files necessary for starting a VASP, ABACUS, or LAMMPS calculations. `run` sends all the computational tasks to remote computational resources defined in a machine configuration file like `machine.json`, and automatically collects the results when remote calculations finish. `post` calculates the desired property from the collected results.

7.1.1 Relaxation

The relaxation of a structure should be carried out before calculating all other properties:

```
dpgen autotest make relax.json
dpgen autotest run relax.json machine.json
dpgen autotest post relax.json
```

If, for some reasons, the main program terminated at stage `run`, one can easily restart with the same command. `relax.json` is the parameter file. An example for `deepmd` relaxation is given as:

```
{
        "structures":   "confs/mp-*",
        "interaction": {
                "type":         "deepmd",
                "model":        "frozen_model.pb",
                "type_map":     {"Al": 0, "Mg": 1}
        },
        "relaxation": {}
}
```

where the key `structures` provides the structures to relax. `interaction` is provided with `deepmd`, and other options are `vasp`, `abacus`, `meam`. . .

7.1.2 Task type

There are now six task types implemented in the package: `vasp`, `abacus`, `deepmd`, `meam`, `eam_fs`, and `eam_alloy`. An `inter.json` file in json format containing the interaction parameters will be written in the directory of each task after `make`. We give input examples of the `interaction` part for each type below:

VASP:

The default of `potcar_prefix` is "".

```
        "interaction": {
                "type":                 "vasp",
                "incar":        "vasp_input/INCAR",
                "potcar_prefix":"vasp_input",
                "potcars":        {"Al": "POTCAR.al", "Mg": "POTCAR.mg"}
        }
```

ABACUS:

The default of `potcar_prefix` is "". The path of potcars/orb_files/deepks_desc is `potcar_prefix` + `potcars/orb_files/deepks_desc`.

```
        "interaction": {
                "type":                 "abacus",
                "incar":        "abacus_input/INPUT",
                "potcar_prefix":"abacus_input",
                "potcars":        {"Al": "pseudo_potential.al", "Mg": "pseudo_potential.
↪mg"},
                "orb_files": {"Al": "numerical_orb.al", "Mg": "numerical_orb.mg"},
                "atom_masses": {"Al": 26.9815, "Mg":24.305},
                "deepks_desc": "jle.orb"
        }
```

deepmd:

Only 1 model can be used in autotest in one working directory.

```
        "interaction": {
                "type":                 "deepmd",
                "model":        "frozen_model.pb",
                "type_map":     {"Al": 0, "Mg": 1}
        }
```

meam:

Please make sure the USER-MEAMC package has already been installed in LAMMPS.

```
        "interaction": {
                "type":                 "meam",
                "model":        ["meam.lib","AlMg.meam"],
                "type_map":     {"Al": 1, "Mg": 2}
        }
```

eam_fs & eam_alloy:

Please make sure the MANYBODY package has already been installed in LAMMPS

```
        "interaction": {
                "type":                 "eam_fs (eam_alloy)",
                "model":            "AlMg.eam.fs (AlMg.eam.alloy)",
                "type_map":         {"Al": 1, "Mg": 2}
        }
```

7.1.3 Property type

Now the supported property types are `eos`, `elastic`, `vacancy`, `interstitial`, `surface`, and `gamma`. Before property tests, `relaxation` should be done first or the relaxation results should be present in the corresponding directory `confs/mp-*/relaxation/relax_task`. A file named `task.json` in json format containing the property parameter will be written in the directory of each task after `make` step. Multiple property tests can be performed simultaneously.

7.2 Make run and post

There are three operations in auto test package, namely `make`, `run`, and `post`. Here we take `eos` property as an example for property type.

7.2.1 Make

The `INCAR`, `POSCAR`, `POTCAR` input files for VASP or `in.lammps`, `conf.lmp`, and the interatomic potential files for LAMMPS will be generated in the directory `confs/mp-*/relaxation/relax_task` for relaxation or `confs/mp-*/eos_00/task.[0-9]*[0-9]` for EOS. The `machine.json` file is not needed for `make`. Example:

```
dpgen autotest make relaxation.json
```

7.2.2 Run

The jobs would be dispatched according to the parameter in `machine.json` file and the calculation results would be sent back. Example:

```
dpgen autotest run relaxation.json machine.json
```

7.2.3 Post

The post process of calculation results would be performed. `result.json` in json format will be generated in `confs/mp-*/relaxation/relax_task` for relaxation and `result.json` in json format and `result.out` in txt format in `confs/mp-*/eos_00` for EOS. The `machine.json` file is also not needed for `post`. Example:

```
dpgen autotest post relaxation.json
```

7.3 Relaxation

7.3.1 Relaxation make

The list of the directories storing structures are `["confs/std-*"]` in the previous example. For single element system, if `POSCAR` doesn't exist in the directories: `std-fcc`, `std-hcp`, `std-dhcp`, `std-bcc`, `std-diamond`, and `std-sc`, the package will automatically generate the standard crystal structures **`fcc`**, **`hcp`**, **`dhcp`**, **`bcc`**, **`diamond`**, and **`sc`** in the corresponding directories, respectively. In other conditions and for multi-component system (more than 1), if `POSCAR` doesn't exist, the package will terminate and print the error **"no configuration for autotest"**.

VASP relaxation

Take the input example of Al in the previous section, when we do `make` as follows:

```
dpgen autotest make relaxation.json
```

the following files would be generated:

```
tree confs/std-fcc/relaxation/
```

```
confs/std-fcc/relaxation/
|-- INCAR
|-- POTCAR
`-- relax_task
    |-- INCAR -> ../INCAR
    |-- inter.json
    |-- KPOINTS
    |-- POSCAR -> ../../POSCAR
    |-- POTCAR -> ../POTCAR
    `-- task.json
```

`inter.json` records the information in the `interaction` dictionary and `task.json` records the information in the `relaxation` dictionary.

LAMMPS relaxation

```
dpgen autotest make relaxation.json
tree confs/std-fcc/
```

the output would be:

```
confs/std-fcc/
|-- POSCAR
`-- relaxation
    |-- frozen_model.pb -> ../../../frozen_model.pb
    |-- in.lammps
    `-- relax_task
        |-- conf.lmp
        |-- frozen_model.pb -> ../frozen_model.pb
        |-- in.lammps -> ../in.lammps
        |-- inter.json
```

(continues on next page)

(continued from previous page)

```
        |-- POSCAR -> ../../POSCAR
        `-- task.json
```

the `conf.lmp` is the input configuration and `in.lammps` is the input command file for lammps.

in.lammps: the package would generate the file `confs/mp-*/relaxation/in.lammps` as follows and we refer the user to the further information of fix box/relax function in lammps:

```
clear
units                  metal
dimension          3
boundary           p p p
atom_style           atomic
box                tilt large
read_data          conf.lmp
mass               1 26.982
neigh_modify       every 1 delay 0 check no
pair_style deepmd frozen_model.pb
pair_coeff
compute            mype all pe
thermo             100
thermo_style       custom step pe pxx pyy pzz pxy pxz pyz lx ly lz vol c_mype
dump               1 all custom 100 dump.relax id type xs ys zs fx fy fz
min_style          cg
fix                1 all box/relax iso 0.0
minimize           0 1.000000e-10 5000 500000
fix                1 all box/relax aniso 0.0
minimize           0 1.000000e-10 5000 500000
variable           N equal count(all)
variable           V equal vol
variable           E equal "c_mype"
variable           tmplx equal lx
variable           tmply equal ly
variable           Pxx equal pxx
variable           Pyy equal pyy
variable           Pzz equal pzz
variable           Pxy equal pxy
variable           Pxz equal pxz
variable           Pyz equal pyz
variable           Epa equal ${E}/${N}
variable           Vpa equal ${V}/${N}
variable           AA equal (${tmplx}*${tmply})
print "All done"
print "Total number of atoms = ${N}"
print "Final energy per atoms = ${Epa}"
print "Final volume per atoms = ${Vpa}"
print "Final Base area = ${AA}"
print "Final Stress (xx yy zz xy xz yz) = ${Pxx} ${Pyy} ${Pzz} ${Pxy} ${Pxz} ${Pyz}"
```

If user provides lammps input command file `in.lammps`, the `thermo_style` and `dump` commands should be the same as the above file.

interatomic potential model: the `frozen_model.pb` in `confs/mp-*/relaxation` would link to the `frozen_model.pb` file given in the input.

7.3.2 Relaxation run

The work path of each task should be in the form like `confs/mp-*/relaxation` and all task is in the form like `confs/mp-*/relaxation/relax_task`.

The `machine.json` file should be applied in this process and the machine parameters (eg. GPU or CPU) are determined according to the task type (VASP or LAMMPS). Then in each work path, the corresponding tasks would be submitted and the results would be sent back through make_dispatcher.

Take `deepmd` run for example:

```
nohup dpgen autotest run relaxation.json machine-ali.json > run.result 2>&1 &
tree confs/std-fcc/relaxation/
```

the output would be:

```
confs/std-fcc/relaxation/
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
|-- jr.json
`-- relax_task
    |-- conf.lmp
    |-- dump.relax
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- log.lammps
    |-- outlog
    |-- POSCAR -> ../../POSCAR
    `-- task.json
```

`dump.relax` is the file storing configurations and `log.lammps` is the output file for lammps.

7.3.3 Relaxation post

Take `deepmd` post for example:

```
dpgen autotest post relaxation.json
tree confs/std-fcc/relaxation/
```

the output will be:

```
confs/std-fcc/relaxation/
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
|-- jr.json
`-- relax_task
    |-- conf.lmp
    |-- CONTCAR
    |-- dump.relax
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- log.lammps
```

(continues on next page)

(continued from previous page)

```
    |-- outlog
    |-- POSCAR -> ../../POSCAR
    |-- result.json
    `-- task.json
```

`result.json` stores the box cell, coordinates, energy, force, virial,... information of each frame in the relaxation trajectory and `CONTCAR` is the final equilibrium configuration.

`result.json`:


```
{
    "@module": "dpdata.system",
    "@class": "LabeledSystem",
    "data": {
        "atom_numbs": [
            1
        ],
        "atom_names": [
            "Al"
        ],
        "atom_types": {
            "@module": "numpy",
            "@class": "array",
            "dtype": "int64",
            "data": [
                0
            ]
        },
        "orig": {
            "@module": "numpy",
            "@class": "array",
            "dtype": "int64",
            "data": [
                0,
                0,
                0
            ]
        },
        "cells": {
            "@module": "numpy",
            "@class": "array",
            "dtype": "float64",
            "data": [
                [
                    [
                        2.8637824638,
                        0.0,
                        0.0
                    ],
                    [
                        1.4318912319,
                        2.4801083646,
                        0.0
```


(continues on next page)

(continued from previous page)


```
                    ],
                    [
                        1.4318912319,
                        0.8267027882,
                        2.3382685902
                    ]
                ],
                [
                    [
                        2.8549207998018438,
                        0.0,
                        0.0
                    ],
                    [
                        1.4274603999009239,
                        2.472433938457684,
                        0.0
                    ],
                    [
                        1.4274603999009212,
                        0.8241446461525599,
                        2.331033071844216
                    ]
                ],
                [
                    [
                        2.854920788303194,
                        0.0,
                        0.0
                    ],
                    [
                        1.427460394144466,
                        2.472433928487206,
                        0.0
                    ],
                    [
                        1.427460394154763,
                        0.8241446428350139,
                        2.331033062460779
                    ]
                ]
            ]
        },
        "coords": {
            "@module": "numpy",
            "@class": "array",
            "dtype": "float64",
            "data": [
                [
                    [
                        0.0,
                        0.0,
```


(continues on next page)

(continued from previous page)


```
                    0.0
                ]
            ],
            [
                [
                    5.709841595683707e-25,
                    -4.3367974740910857e-19,
                    0.0
                ]
            ],
            [
                [
                    -8.673606219968035e-19,
                    8.673619637565944e-19,
                    8.673610853102186e-19
                ]
            ]
        ]
    },
    "energies": {
        "@module": "numpy",
        "@class": "array",
        "dtype": "float64",
        "data": [
            -3.745029,
            -3.7453815,
            -3.7453815
        ]
    },
    "forces": {
        "@module": "numpy",
        "@class": "array",
        "dtype": "float64",
        "data": [
            [
                [
                    0.0,
                    -6.93889e-18,
                    -3.46945e-18
                ]
            ],
            [
                [
                    1.38778e-17,
                    6.93889e-18,
                    -1.73472e-17
                ]
            ],
            [
                [
                    1.38778e-17,
                    1.73472e-17,
```


(continues on next page)

(continued from previous page)


```
                    -4.51028e-17
                ]
            ]
        ]
    },
    "virials": {
        "@module": "numpy",
        "@class": "array",
        "dtype": "float64",
        "data": [
            [
                [
                    -0.07534992071654338,
                    1.2156615579052586e-17,
                    1.3904892126132796e-17
                ],
                [
                    1.2156615579052586e-17,
                    -0.07534992071654338,
                    4.61571024026576e-12
                ],
                [
                    1.3904892126132796e-17,
                    4.61571024026576e-12,
                    -0.07534992071654338
                ]
            ],
            [
                [
                    -9.978994290457664e-08,
                    -3.396452753975288e-15,
                    8.785831629151552e-16
                ],
                [
                    -3.396452753975288e-15,
                    -9.991375413666671e-08,
                    5.4790751628409565e-12
                ],
                [
                    8.785831629151552e-16,
                    5.4790751628409565e-12,
                    -9.973497959053003e-08
                ]
            ],
            [
                [
                    1.506940521266962e-11,
                    1.1152016233536118e-11,
                    -8.231900529157644e-12
                ],
                [
                    1.1152016233536118e-11,
```


(continues on next page)

(continued from previous page)


```
                    -6.517665029355618e-11,
                    -6.33706710415926e-12
                ],
                [
                    -8.231900529157644e-12,
                    -6.33706710415926e-12,
                    5.0011471096530724e-11
                ]
            ]
        ]
    },
    "stress": {
        "@module": "numpy",
        "@class": "array",
        "dtype": "float64",
        "data": [
            [
                [
                    -7.2692250000000005,
                    1.1727839e-15,
                    1.3414452e-15
                ],
                [
                    1.1727839e-15,
                    -7.2692250000000005,
                    4.4529093000000003e-10
                ],
                [
                    1.3414452e-15,
                    4.4529093000000003e-10,
                    -7.2692250000000005
                ]
            ],
            [
                [
                    -9.71695e-06,
                    -3.3072633e-13,
                    8.5551193e-14
                ],
                [
                    -3.3072633e-13,
                    -9.729006000000001e-06,
                    5.3351969e-10
                ],
                [
                    8.5551193e-14,
                    5.3351969e-10,
                    -9.711598e-06
                ]
            ],
            [
                [
```


(continues on next page)

(continued from previous page)

```
                        1.4673689e-09,
                        1.0859169e-09,
                        -8.0157343e-10
                    ],
                    [
                        1.0859169e-09,
                        -6.3465139e-09,
                        -6.1706584e-10
                    ],
                    [
                        -8.0157343e-10,
                        -6.1706584e-10,
                        4.8698191e-09
                    ]
                ]
            ]
        }
    }
}
```

7.4 Property

7.4.1 Property get started and input examples

Here we take deepmd for example and the input file for other task types is similar.

```
{
    "structures":       ["confs/std-*"],
    "interaction": {
        "type":          "deepmd",
        "model":         "frozen_model.pb",
        "type_map":     {"Al": 0}
    },
    "properties": [
        {
         "type":         "eos",
         "vol_start":    0.9,
         "vol_end":      1.1,
         "vol_step":     0.01
        },
        {
         "type":         "elastic",
         "norm_deform":  1e-2,
         "shear_deform": 1e-2
        },
        {
         "type":              "vacancy",
         "supercell":         [3, 3, 3],
         "start_confs_path": "../vasp/confs"
        },
```

(continues on next page)

(continued from previous page)

```
        {
         "type":          "interstitial",
         "supercell":    [3, 3, 3],
         "insert_ele":   ["Al"],
         "conf_filters":{"min_dist": 1.5},
         "cal_setting": {"input_prop": "lammps_input/lammps_high"}
        },
        {
         "type":            "surface",
         "min_slab_size":   10,
         "min_vacuum_size":11,
         "max_miller":      2,
         "cal_type":        "static"
        },
        {
         "type": "gamma",
         "lattice_type": "fcc",
         "miller_index": [1, 1, 1],
         "displace_direction": [1, 1, 0],
         "supercell_size": [1, 1, 10],
         "min_vacuum_size": 10,
         "add_fix": ["true", "true", "false"],
         "n_steps": 20
        }
        ]
}
```

Universal key words for properties

Key words	data structure	example	description
type	String	"eos"	property type
skip	Boolean	true	whether to skip current property or not
start_confs_path	String	"../vasp/confs"	start from the equilibrium configuration in other path only for the current property type
cal_setting["input_prop"]	String	"lammps_input/lammps_high"	input commands file
cal_setting["overwrite_interaction"]	Dict		overwrite the interaction in the `interaction` part only for the current property type

other parameters in `cal_setting` and `cal_type` in `relaxation` also apply in `property`.

Key words for **EOS**

Key words	data structure	example	description
vol_start	Float	0.9	the starting volume related to the equilibrium structure
vol_end	Float	1.1	the biggest volume related to the equilibrium structure
vol_step	Float	0.01	the volume increment related to the equilibrium structure
vol_abs	Boolean	false	whether to treat vol_start, vol_end and vol_step as absolute volume or not (as relative volume), default = false

Key words for **Elastic**

Key words	data structure	example	description
norm_deform	Float	1e-2	deformation in xx, yy, zz, default = 1e-2
shear_deform	Float	1e-2	deformation in other directions, default = 1e-2

Key words for **Vacancy**

Key words	data structure	example	description
supercell	List of Int	[3,3,3]	the supercell to be constructed, default = [1,1,1]

Key words for **Interstitial**

Key words	data structure	example	description
insert_ele	List of String	["Al"]	the element to be inserted
supercell	List of Int	[3,3,3]	the supercell to be constructed, default = [1,1,1]
conf_filters	Dict	"min_dist": 1.5	filter out the undesirable configuration
bcc_self	Boolean	false	whether to do the self-interstitial calculations for bcc structures, default = false

Key words for **Surface**

Key words	data structure	example	description
min_slab_size	Int	10	minimum size of slab thickness
min_vacuum_size	Int	11	minimum size of vacuum width
pert_xz	Float	0.01	perturbation through xz direction used to compute surface energy, default = 0.01
max_miller	Int	2	the maximum miller index, default = 2

Key words for **Gamma**

Key words	data structure	example	description
lattice_type	String	"fcc"	"bcc" or "fcc" at this stage
miller_index	List of Int	[1,1,1]	slip plane for gamma-line calculation
displace_direction	List of Int	[1,1,0]	slip direction for gamma-line calculation
supercell_size	List of Int	[1,1,10]	the supercell to be constructed, default = [1,1,5]
min_vacuum_size	Int or Float	10	minimum size of vacuum width, default = 20
add_fix	List of String	['true','true','false']	whether to fix atoms in the direction, default = ['true','true','false'] (standard method)
n_steps	Int	20	Number of points for gamma-line calculation, default = 10

7.4.2 Property make

```
dpgen autotest make property.json
```

EOS output:

```
confs/std-fcc/eos_00/
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- task.000000
|   |-- conf.lmp
|   |-- eos.json
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps
|   |-- inter.json
|   |-- POSCAR
|   |-- POSCAR.orig -> ../../relaxation/relax_task/CONTCAR
|   `-- task.json
|-- task.000001
|   |-- conf.lmp
|   |-- eos.json
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps
|   |-- inter.json
|   |-- POSCAR
|   |-- POSCAR.orig -> ../../relaxation/relax_task/CONTCAR
|   `-- task.json
...
`-- task.000019
    |-- conf.lmp
    |-- eos.json
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps
    |-- inter.json
    |-- POSCAR
    |-- POSCAR.orig -> ../../relaxation/relax_task/CONTCAR
    `-- task.json
```

`eos.json` records the `volume` and `scale` of the corresponding task.

Elastic output:

```
confs/std-fcc/elastic_00/
|-- equi.stress.json
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
|-- POSCAR -> ../relaxation/relax_task/CONTCAR
|-- task.000000
|   |-- conf.lmp
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps -> ../in.lammps
|   |-- inter.json
|   |-- POSCAR
|   |-- strain.json
|   `-- task.json
```

(continues on next page)

(continued from previous page)

```
|-- task.000001
|   |-- conf.lmp
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps -> ../in.lammps
|   |-- inter.json
|   |-- POSCAR
|   |-- strain.json
|   `-- task.json
...
`-- task.000023
    |-- conf.lmp
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- POSCAR
    |-- strain.json
    `-- task.json
```

`equi.stress.json` records the stress information of the equilibrium task and `strain.json` records the deformation information of the corresponding task.

Vacancy output:

```
confs/std-fcc/vacancy_00/
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
|-- POSCAR -> ../relaxation/relax_task/CONTCAR
`-- task.000000
    |-- conf.lmp
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- POSCAR
    |-- supercell.json
    `-- task.json
```

`supercell.json` records the supercell size information of the corresponding task.

Interstitial output:

```
confs/std-fcc/interstitial_00/
|-- element.out
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
|-- POSCAR -> ../relaxation/relax_task/CONTCAR
|-- task.000000
|   |-- conf.lmp
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps -> ../in.lammps
|   |-- inter.json
|   |-- POSCAR
|   |-- supercell.json
|   `-- task.json
```

(continues on next page)

(continued from previous page)

```
`-- task.000001
    |-- conf.lmp
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- POSCAR
    |-- supercell.json
    `-- task.json
```

`element.out` records the inserted element type of each task and `supercell.json` records the supercell size information of the corresponding task.

Surface output:

```
confs/std-fcc/surface_00/
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
|-- POSCAR -> ../relaxation/relax_task/CONTCAR
|-- task.000000
|   |-- conf.lmp
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps -> ../in.lammps
|   |-- inter.json
|   |-- miller.json
|   |-- POSCAR
|   |-- POSCAR.tmp
|   `-- task.json
|-- task.000001
|   |-- conf.lmp
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps -> ../in.lammps
|   |-- inter.json
|   |-- miller.json
|   |-- POSCAR
|   |-- POSCAR.tmp
|   `-- task.json
...
`-- task.000008
    |-- conf.lmp
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- miller.json
    |-- POSCAR
    |-- POSCAR.tmp
    `-- task.json
```

`miller.json` records the miller index of the corresponding task.

7.4.3 Property run

```
nohup dpgen autotest run property.json machine-ali.json > run.result 2>&1 &
```

the result file `log.lammps`, `dump.relax`, and `outlog` would be sent back.

7.4.4 Property-post

Use command

```
dpgen autotest post property.json
```

to post results as `result.json` and `result.out` in each property's path.

7.4.5 Properties

EOS get started and input examples

Equation of State (EOS) here calculates the energies of the most stable structures as a function of volume. Users can refer to Figure 4 of the dpgen CPC paper for more information of EOS.

An example of the input file for EOS by VASP:

```
{
        "structures":         ["confs/mp-*","confs/std-*","confs/test-*"],
        "interaction": {
                "type":                 "vasp",
                "incar":         "vasp_input/INCAR",
                "potcar_prefix":"vasp_input",
                "potcars":         {"Al": "POTCAR.al", "Mg": "POTCAR.mg"}
        },
        "properties": [
        {
         "type":          "eos",
         "vol_start":     0.9,
         "vol_end":       1.1,
         "vol_step":      0.01
        }
        ]
}
```

`vol_start` is the starting volume relative to the equilibrium structure, `vol_step` is the volume increment step relative to the equilibrium structure, and the biggest relative volume is smaller than `vol_end`.

EOS make

Step 1. Before `make` in EOS, the equilibrium configuration `CONTCAR` must be present in `confs/mp-*/relaxation`.

Step 2. For the input example in the previous section, when we do `make`, 40 tasks would be generated as `confs/mp-*/eos_00/task.000000, confs/mp-*/eos_00/task.000001, ... , confs/mp-*/eos_00/task.000039`. The suffix `00` is used for possible `refine` later.

Step 3. If the task directory, for example `confs/mp-*/eos_00/task.000000` is not empty, the old input files in it including `INCAR`, `POSCAR`, `POTCAR`, `conf.lmp`, `in.lammps` would be deleted.

Step 4. In each task directory, `POSCAR.orig` would link to `confs/mp-*/relaxation/CONTCAR`. Then the `scale` parameter can be calculated as:

```
scale = (vol_current / vol_equi) ** (1. / 3.)
```

`vol_current` is the corresponding volume per atom of the current task and `vol_equi` is the volume per atom of the equilibrium configuration. Then the `poscar_scale` function in `dpgen.auto_test.lib.vasp` module would help to generate `POSCAR` file with `vol_current` in `confs/mp-*/eos_00/task.[0-9]*[0-9]`.

Step 5. According to the task type, the input file including `INCAR`, `POTCAR` or `conf.lmp`, `in.lammps` would be written in every `confs/mp-*/eos_00/task.[0-9]*[0-9]`.

EOS run

The work path of each task should be in the form like `confs/mp-*/eos_00` and all task is in the form like `confs/mp-*/eos_00/task.[0-9]*[0-9]`.

When we dispatch tasks, we would go through every individual work path in the list `confs/mp-*/eos_00`, and then submit `task.[0-9]*[0-9]` in each work path.

EOS post

The post processing of EOS would go to every directory in `confs/mp-*/eos_00` and do the post processing. Let's suppose we are now in `confs/mp-100/eos_00` and there are `task.000000, task.000001,..., task.000039` in this directory. By reading `inter.json` file in every task directory, the task type can be determined and the energy and force information of every task can further be obtained. By appending the `dict` of energy and force into a list, an example of the list with 1 atom is given as:

```
[
    {"energy": E1, "force": [fx1, fy1, fz1]},
    {"energy": E2, "force": [fx2, fy2, fz2]},
    ...
    {"energy": E40, "force": [fx40, fy40, fz40]}
]
```

Then the volume can be calculated from the task id and the corresponding energy can be obtained from the list above. Finally, there would be `result.json` in json format and `result.out` in txt format in `confs/mp-100/eos_00` containing the EOS results.

An example of `result.json` is give as:

```
{
    "14.808453313267595": -3.7194474,
    "14.972991683415014": -3.7242038,
```

(continues on next page)

(continued from previous page)

```
        ...
    "17.934682346068534": -3.7087655
}
```

An example of `result.out` is given below:

```
onf_dir: /root/auto_test_example/deepmd/confs/std-fcc/eos_00
 VpA(A^3)  EpA(eV)
 14.808   -3.7194
 14.973   -3.7242
   ...      ...
 17.935   -3.7088
```

Elastic get started and input examples

Here we calculate the mechanical properties which include elastic constants (C11 to C66), bulk modulus Bv, shear modulus Gv, Youngs modulus Ev, and Poission ratio Uv of a certain crystal structure.

An example of the input file for Elastic by deepmd:

```
{
        "structures":        ["confs/mp-*","confs/std-*","confs/test-*"],
        "interaction": {
                "type": "deepmd",
        "model": "frozen_model.pb",
                "type_map":        {"Al": 0, "Mg": 1}
        },
        "properties": [
            {
                "type": "elastic",
                "norm_deform": 1e-2,
                    "shear_deform": 1e-2
                }
        ]
}
```

Here the default values of `norm_deform` and `shear_deform` are **1e-2** and **1e-2**, respectively. A list of `norm_strains` and `shear_strains` would be generated as below:

```
[-norm_def, -0.5 * norm_def, 0.5 * norm_def, norm_def]
[-shear_def, -0.5 * shear_def, 0.5 * shear_def, shear_def]
```

Elastic make

Step 1. The `DeformedStructureSet` module in pymatgen.analysis.elasticity.strain is used to generate a set of independently deformed structures. `equi.stress.out` file is written to record the equilibrium stress in the Elastic directory. For the example in the previous section, `equi.stress.out` should be in `confs/mp-*/elastic_00`.

Step 2. If there are `init_from_suffix` and `output_suffix` parameter in the `properties` part, the *refine process* follows. Else, the deformed structure (`POSCAR`) and strain information (`strain.out`) are written in the task directory, for example, in `confs/mp-*/elastic_00/task.000000`.

Step 3. When doing `elastic` by VASP, `ISIF=2`. When doing by LAMMPS, the following `in.lammps` would be written.

```
units         metal
dimension        3
boundary        p        p    p
atom_style        atomic
box             tilt large
read_data       conf.lmp
mass            1 1
mass            2 1
neigh_modify    every 1 delay 0 check no
pair_style deepmd frozen_model.pb
pair_coeff
compute         mype all pe
thermo          100
thermo_style    custom step pe pxx pyy pzz pxy pxz pyz lx ly lz vol c_mype
dump            1 all custom 100 dump.relax id type xs ys zs fx fy fz
min_style       cg
minimize        0 1.000000e-10 5000 500000
variable        N equal count(all)
variable        V equal vol
variable        E equal "c_mype"
variable        Pxx equal pxx
variable        Pyy equal pyy
variable        Pzz equal pzz
variable        Pxy equal pxy
variable        Pxz equal pxz
variable        Pyz equal pyz
variable        Epa equal ${E}/${N}
variable        Vpa equal ${V}/${N}
print "All done"
print "Total number of atoms = ${N}"
print "Final energy per atoms = ${Epa}"
print "Final volume per atoms = ${Vpa}"
print "Final Stress (xx yy zz xy xz yz) = ${Pxx} ${Pyy} ${Pzz} ${Pxy} ${Pxz} ${Pyz}"
```

Elastic run

Very similar to the `run` operation of `EOS` except for in different directories. Now the work path of each task should be in the form like `confs/mp-*/elastic_00` and all task is in the form like `confs/mp-*/elastic_00/task.[0-9]*[0-9]`.

Elastic post

The `ElasticTensor` module in pymatgen.analysis.elasticity.elastic is used to get the elastic tensor, Bv, and Gv. The mechanical properties of a crystal structure would be written in `result.json` in json format and `result.out` in txt format. The example of the output file is give below.

result.json


```
{
    "elastic_tensor": [
        134.90955999999997,
        54.329958699999985,
        51.802386099999985,
        3.5745279599999993,
        -1.3886325999999648e-05,
        -1.9638233999999486e-05,
        54.55840299999999,
        134.59654699999996,
        51.7972336,
        -3.53972684,
        1.839568799999963e-05,
        8.756799399999951e-05,
        51.91324859999999,
        51.913292199999994,
        137.01763799999998,
        -5.090339399999969e-05,
        6.99251629999996e-05,
        3.736478699999946e-05,
        3.8780564440000007,
        -3.770445632,
        -1.2766205999999956,
        35.41343199999999,
        2.2479590800000023e-05,
        1.3837692000000172e-06,
        -4.959999999495933e-06,
        2.5800000003918792e-06,
        1.4800000030874965e-06,
        2.9000000008417968e-06,
        35.375960199999994,
        3.8608356,
        0.0,
        0.0,
        0.0,
        0.0,
        4.02554856,
```


(continues on next page)

(continued from previous page)

```
        38.375018399999995
    ],
    "BV": 80.3153630222222,
    "GV": 38.40582656,
    "EV": 99.37716395728943,
    "uV": 0.2937771799031088
}
```

The order of `elastic_tensor` is C11, C12, ..., C16, C21, C22, ..., C26, ..., C66 and the unit of Bv, Gv, Ev, and uv is GPa.

result.out

```
/root/auto_test_example/deepmd/confs/std-fcc/elastic_00
 134.91   54.33   51.80    3.57   -0.00   -0.00
  54.56  134.60   51.80   -3.54    0.00    0.00
  51.91   51.91  137.02   -0.00    0.00    0.00
   3.88   -3.77   -1.28   35.41    0.00    0.00
  -0.00    0.00    0.00    0.00   35.38    3.86
   0.00    0.00    0.00    0.00    4.03   38.38
# Bulk   Modulus BV = 80.32 GPa
# Shear  Modulus GV = 38.41 GPa
# Youngs Modulus EV = 99.38 GPa
# Poission Ratio uV = 0.29
```

Vacancy get started and input examples

`Vacancy` calculates the energy difference when removing an atom from the crystal structure. We only need to give the information of `supercell` to help calculate the vacancy energy and the default value of `supercell` is [1, 1, 1].

An example of the input file for Vacancy by deepmd:

```
{
        "structures":        "confs/mp-*",
        "interaction": {
                "type":               "deepmd",
                "model":        "frozen_model.pb",
                "type_map":        {"Al": 0, "Mg": 1}
        },
        "properties": [
            {
                "type":         "vacancy",
                "supercell":        [1, 1, 1]
            }
        ]
}
```

Vacancy make

Step 1. The `VacancyGenerator` module in pymatgen.analysis.defects.generators is used to generate a set of structures with vacancy.

Step 2. If there are `init_from_suffix` and `output_suffix` parameter in the `properties` part, the *refine process* follows. If reproduce is evoked, the *reproduce process* follows. Otherwise, the vacancy structure (`POSCAR`) and supercell information (`supercell.out`) are written in the task directory, for example, in `confs/mp-*/vacancy_00/task.000000` with the check and possible removing of the old input files like before.

Step 3. When doing `vacancy` by VASP, `ISIF = 3`. When doing `vacancy` by LAMMPS, the same `in.lammps` as that in *EOS (change_box is True)* would be generated with `scale` set to one.

Vacancy run

Very similar to the `run` operation of `EOS` except for in different directories. Now the work path of each task should be in the form like `confs/mp-*/vacancy_00` and all task is in the form like `confs/mp-*/vacancy_00/task.[0-9]*[0-9]`.

Vacancy post

For `Vacancy`, we need to calculate the energy difference between a crystal structure with and without a vacancy. The examples of the output files `result.json` in json format and `result.out` in txt format are given below.

result.json

```
{
    "[3, 3, 3]-task.000000": [
        0.7352769999999964,
        -96.644642,
        -97.379919
    ]
}
```

result.out

```
/root/auto_test_example/deepmd/confs/std-fcc/vacancy_00
Structure:          Vac_E(eV)  E(eV) equi_E(eV)
[3, 3, 3]-task.000000:   0.735  -96.645 -97.380
```

Interstitial get started and input examples

`Interstitial` calculates the energy difference when adding an atom into the crystal structure. We need to give the information of `supercell` (default value is [1, 1, 1]) and `insert_ele` list for the element types of the atoms added in.

An example of the input file for Interstitial by deepmd:

```
{
        "structures":         "confs/mp-*",
        "interaction": {
                "type":                 "deepmd",
                "model":          "frozen_model.pb",
                "type_map":          {"Al": 0, "Mg": 1}
        },
        "properties": [
            {
                "type":          "interstitial",
                "supercell":    [3, 3, 3],
                "insert_ele":   ["Al"],
                "conf_filters":{"min_dist": 1.5},
                "cal_setting": {"input_prop": "lammps_input/lammps_high"}
            }
        ]
}
```

We add a `conf_filters` parameter in `properties` part and this parameter can help to eliminate undesirable structure which can render rather difficult convergence in calculations. In the example above, **"min_dist": 1.5** means if the smallest atomic distance in the structure is less than 1.5 angstrom, the configuration would be eliminated and not used in calculations.

Interstitial make

Step 1. For each element in `insert_ele` list, `InterstitialGenerator` module in pymatgen.analysis.defects.generators would help to generate interstitial structure. The structure would be appended into a list if it can meet the requirements in `conf_filters`.

Step 2. If `refine` is `True`, we do *refine process*. If `reprod-opt` is `True` (the default is **False**), we do *reproduce process*. Else, the vacancy structure (`POSCAR`) and supercell information (`supercell.out`) are written in the task directory, for example, in `confs/mp-*/interstitial_00/task.000000` with the check and possible removing of the old input files like before.

Step 3. In `interstitial` by VASP, `ISIF = 3`. In `interstitial` by LAMMPS, the same `in.lammps` as that in *EOS (change_box is True)* would be generated with `scale` set to one.

Interstitial run

Very similar to the `run` operation of `EOS` except for in different directories. Now the work path of each task should be in the form like `confs/mp-*/interstitial_00` and all task is in the form like `confs/mp-*/interstitial_00/task.[0-9]*[0-9]`.

Interstitial post

For `Interstitial`, we need to calculate the energy difference between a crystal structure with and without atom added in. The examples of the output files `result.json` in json format and `result.out` in txt format are given below.

result.json

```
{
    "Al-[3, 3, 3]-task.000000": [
        4.022952000000004,
        -100.84773,
        -104.870682
    ],
    "Al-[3, 3, 3]-task.000001": [
        2.7829520000000088,
        -102.08773,
        -104.870682
    ]
}
```

result.out

```
/root/auto_test_example/deepmd/confs/std-fcc/interstitial_00
Insert_ele-Struct: Inter_E(eV)  E(eV) equi_E(eV)
Al-[3, 3, 3]-task.000000:   4.023  -100.848 -104.871
Al-[3, 3, 3]-task.000001:   2.783  -102.088 -104.871
```

Surface get started and input examples

`Surface` calculates the surface energy. We need to give the information of `min_slab_size`, `min_vacuum_size`, `max_miller` (default value is 2), and `pert_xz` which means perturbations in xz and will help work around vasp bug.

An example of the input file for Surface by deepmd:

```
{
        "structures":         "confs/mp-*",
        "interaction": {
                "type":                 "deepmd",
                "model":          "frozen_model.pb",
                "type_map":          {"Al": 0, "Mg": 1}
        },
```

(continues on next page)

(continued from previous page)

```
        "properties": [
            {
                "type":           "surface",
                "min_slab_size":  10,
                "min_vacuum_size":11,
                "max_miller":     2,
                "cal_type":       "static"
            }
        ]
}
```

Surface make

Step 1. Based on the equilibrium configuration, `generate_all_slabs` module in pymatgen.core.surface would help to generate surface structure list with using `max_miller`, `min_slab_size`, and `min_vacuum_size` parameters.

Step 2. If `refine` is True, we do *refine process*. If `reprod-opt` is True (the default is False), we do *reproduce process*. Otherwise, the surface structure (`POSCAR`) with perturbations in xz and miller index information (`miller.out`) are written in the task directory, for example, in `confs/mp-*/interstitial_00/task.000000` with the check and possible removing of the old input files like before.

Surface run

Very similar to the `run` operation of `EOS` except for in different directories. Now the work path of each task should be in the form like `confs/mp-*/surface_00` and all task is in the form like `confs/mp-*/surface_00/task.[0-9]*[0-9]`.

Surface post

For `Surface`, we need to calculate the energy difference between a crystal structure with and without a surface with a certain miller index divided by the surface area.

The examples of the output files `result.json` in json format and `result.out` in txt format are given below.

result.json

```
{
    "[1, 1, 1]-task.000000": [
        0.8051037974207992,
        -3.6035018,
        -3.7453815
    ],
    "[2, 2, 1]-task.000001": [
        0.9913881928811771,
        -3.5781115999999997,
        -3.7453815
    ],
    "[1, 1, 0]-task.000002": [
        0.9457333586026173,
```

(continues on next page)

(continued from previous page)


```
        -3.5529366000000002,
        -3.7453815
    ],
    "[2, 2, -1]-task.000003": [
        0.9868013100872397,
        -3.5590607142857142,
        -3.7453815
    ],
    "[2, 1, 1]-task.000004": [
        1.0138239046484236,
        -3.563035875,
        -3.7453815
    ],
    "[2, 1, -1]-task.000005": [
        1.0661817319108005,
        -3.5432459166666668,
        -3.7453815
    ],
    "[2, 1, -2]-task.000006": [
        1.034003253044026,
        -3.550884125,
        -3.7453815
    ],
    "[2, 0, -1]-task.000007": [
        0.9569958287615818,
        -3.5685403333333334,
        -3.7453815
    ],
    "[2, -1, -1]-task.000008": [
        0.9432935501134583,
        -3.5774615714285716,
        -3.7453815
    ]
}
```


result.out

```
/root/auto_test_example/deepmd/confs/std-fcc/surface_00
Miller_Indices:         Surf_E(J/m^2) EpA(eV) equi_EpA(eV)
[1, 1, 1]-task.000000:          0.805      -3.604   -3.745
[2, 2, 1]-task.000001:          0.991      -3.578   -3.745
[1, 1, 0]-task.000002:          0.946      -3.553   -3.745
[2, 2, -1]-task.000003:         0.987      -3.559   -3.745
[2, 1, 1]-task.000004:          1.014      -3.563   -3.745
[2, 1, -1]-task.000005:         1.066      -3.543   -3.745
[2, 1, -2]-task.000006:         1.034      -3.551   -3.745
[2, 0, -1]-task.000007:         0.957      -3.569   -3.745
[2, -1, -1]-task.000008:        0.943      -3.577   -3.745
```

7.5 Refine

7.5.1 Refine get started and input examples

Sometimes we want to refine the calculation of a property from previous results. For example, when higher convergence criteria `EDIFF` and `EDIFFG` are necessary in VASP, the new VASP calculation is desired to start from the previous output configuration, rather than starting from scratch.

An example of the input file `refine.json` is given below:

```
{
    "structures":       ["confs/std-*"],
    "interaction": {
        "type":          "deepmd",
        "model":         "frozen_model.pb",
        "type_map":     {"Al": 0}
    },
    "properties": [
        {
        "type":              "vacancy",
        "init_from_suffix": "00",
        "output_suffix":    "01",
        "cal_setting":     {"input_prop":  "lammps_input/lammps_high"}
        }
        ]
}
```

In this example, `refine` would output the results to `vacancy_01` based on the previous results in `vacancy_00` by using a different input commands file for lammps.

7.5.2 Refine make

```
dpgen autotest make refine.json
tree confs/std-fcc/vacancy_01/
```

the output will be:

```
confs/std-fcc/vacancy_01/
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
`-- task.000000
    |-- conf.lmp
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- POSCAR -> ../../vacancy_00/task.000000/CONTCAR
    |-- supercell.json -> ../../vacancy_00/task.000000/supercell.json
    `-- task.json
```

an new directory `vacancy_01` would be established and the starting configuration links to previous results.

7.5.3 Refine run

```
nohup dpgen autotest run refine.json machine-ali.json > run.result 2>&1 &
```

the run process of `refine` is similar to before.

7.5.4 Refine post

```
dpgen autotest post refine.json
```

the post process of `refine` is similar to the corresponding property.

7.6 Reproduce

7.6.1 Reproduce get started and input examples

Sometimes we want to reproduce the initial results with the same configurations for cross validation. This version of autotest package can accomplish this successfully in all property types except for `Elastic`. An input example for using `deepmd` to reproduce the `VASP` Interstitial results is given below:

```
{
    "structures":       ["confs/std-*"],
    "interaction": {
        "type":          "deepmd",
        "model":         "frozen_model.pb",
        "type_map":     {"Al": 0}
    },
    "properties": [
        {
        "type":             "interstitial",
        "reproduce":        true,
        "init_from_suffix": "00",
        "init_data_path":   "../vasp/confs",
        "reprod_last_frame":       false
        }
        ]
}
```

`reproduce` denotes whether to do `reproduce` or not and the default value is False.

`init_data_path` is the path of VASP or LAMMPS initial data to be reproduced. `init_from_suffix` is the suffix of the initial data and the default value is "00". In this case, the VASP Interstitial results are stored in `../vasp/confs/std-*/interstitial_00` and the reproduced Interstitial results would be in `deepmd/confs/std-*/interstitial_reprod`.

`reprod_last_frame` denotes if only the last frame is used in reproduce. The default value is True for eos and surface, but is False for vacancy and interstitial.

7.6.2 Reproduce make

```
dpgen autotest make reproduce.json
tree confs/std-fcc/interstitial_reprod/
```

the output will be:

```
confs/std-fcc/interstitial_reprod/
|-- frozen_model.pb -> ../../../frozen_model.pb
|-- in.lammps
|-- task.000000
|   |-- conf.lmp
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps -> ../in.lammps
|   |-- inter.json
|   |-- POSCAR
|   `-- task.json
|-- task.000001
|   |-- conf.lmp
|   |-- frozen_model.pb -> ../frozen_model.pb
|   |-- in.lammps -> ../in.lammps
|   |-- inter.json
|   |-- POSCAR
|   `-- task.json
...
`-- task.000038
    |-- conf.lmp
    |-- frozen_model.pb -> ../frozen_model.pb
    |-- in.lammps -> ../in.lammps
    |-- inter.json
    |-- POSCAR
    `-- task.json
```

every singe frame in the initial data is split into each task and the following `in.lammps` would help to do the `static` calculation:

```
clear
units                  metal
dimension          3
boundary           p p p
atom_style           atomic
box                tilt large
read_data          conf.lmp
mass               1 26.982
neigh_modify       every 1 delay 0 check no
pair_style deepmd frozen_model.pb
pair_coeff
compute            mype all pe
thermo             100
thermo_style       custom step pe pxx pyy pzz pxy pxz pyz lx ly lz vol c_mype
dump               1 all custom 100 dump.relax id type xs ys zs fx fy fz
run                0
variable           N equal count(all)
```

(continues on next page)

(continued from previous page)

```
variable          V equal vol
variable          E equal "c_mype"
variable          tmplx equal lx
variable          tmply equal ly
variable          Pxx equal pxx
variable          Pyy equal pyy
variable          Pzz equal pzz
variable          Pxy equal pxy
variable          Pxz equal pxz
variable          Pyz equal pyz
variable          Epa equal ${E}/${N}
variable          Vpa equal ${V}/${N}
variable          AA equal (${tmplx}*${tmply})
print "All done"
print "Total number of atoms = ${N}"
print "Final energy per atoms = ${Epa}"
print "Final volume per atoms = ${Vpa}"
print "Final Base area = ${AA}"
print "Final Stress (xx yy zz xy xz yz) = ${Pxx} ${Pyy} ${Pzz} ${Pxy} ${Pxz} ${Pyz}"
```

7.6.3 Reproduce run

```
nohup dpgen autotest run reproduce.json machine-ali.json > run.result 2>&1 &
```

the run process of `reproduce` is similar to before.

7.6.4 Reproduce post

```
dpgen autotest post reproduce.json
```

the output will be:

`result.out`:

```
/root/auto_test_example/deepmd/confs/std-fcc/interstitial_reprod
Reproduce: Initial_path Init_E(eV/atom)  Reprod_E(eV/atom)  Difference(eV/atom)
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.020   -3.240   -0.220
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.539   -3.541   -0.002
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.582   -3.582   -0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.582   -3.581    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.594   -3.593    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.594   -3.594    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.598   -3.597    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.600   -3.600    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.600   -3.600    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.601   -3.600    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.602   -3.601    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.603   -3.602    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.603   -3.602    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.603   -3.602    0.001
```

(continues on next page)

(continued from previous page)

```
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.603   -3.602    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.603   -3.602    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.603   -3.602    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000000  -3.603   -3.602    0.001
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.345   -3.372   -0.027
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.546   -3.556   -0.009
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.587   -3.593   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.593   -3.599   -0.006
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.600   -3.606   -0.006
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.600   -3.606   -0.006
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.624   -3.631   -0.006
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.634   -3.640   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.637   -3.644   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.637   -3.644   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.638   -3.645   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.638   -3.645   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
.../vasp/confs/std-fcc/interstitial_00/task.000001  -3.639   -3.646   -0.007
```

the comparison of the initial and reproduced results as well as the absolute path of the initial data is recorded.

`result.json`:

```json
{
    "/root/auto_test_example/vasp/confs/std-fcc/interstitial_00/task.000000": {
        "nframes": 18,
        "error": 0.0009738182472213228
    },
    "/root/auto_test_example/vasp/confs/std-fcc/interstitial_00/task.000001": {
        "nframes": 21,
        "error": 0.0006417039154057605
    }
}
```

the error analysis corresponding to the initial data is recorded and the error of the first frame is disregarded when all the frames are considered in reproduce.

CHAPTER

EIGHT

USER GUIDE

This part aims to show you how to get the community's help. Some frequently asked questions are listed in troubleshooting, and the explanation of errors that often occur is listed in common errors. If other unexpected problems occur, you're welcome to contact us for help.

8.1 Discussions

Welcome everyone to participate in the discussion about DP-GEN in the discussion module. You can ask for help, share an idea or anything to discuss here. Note: before you raise a question, please check TUTORIAL/FAQs and search history discussions to find solutions.

8.2 Issue

If you want to make a bug report or a request for new features, you can make an issue in the issue module.

Here are the types you can choose. A proper type can help developer figure out what you need. Also, you can assign yourself to solve the issue. Your contribution is welcome!

Note: before you raise a question, please check TUTORIAL/FAQs and search history issues to find solutions.

8.3 Tutorials

Tutorials can be found here.

8.4 Example for parameters

If you have no idea how to prepare a `PARAM` for your task, you can find examples of PARAM for different tasks in examples.

For example, if you want to set specific template for LAMMPS, you can find an example here

If you want to learn more about Machine parameters, please check docs for dpdispatcher

8.5 Pull requests - How to contribute

8.6 Troubleshooting

1. The most common problem is whether two settings correspond with each other, including:
    - The order of elements in `type_map` and `mass_map` and **`fp_pp_files`**.
    - Size of `init_data_sys` and `init_batch_size`.
    - Size of `sys_configs` and `sys_batch_size`.
    - Size of `sel_a` and actual types of atoms in your system.
    - Index of `sys_configs` and `sys_idx`.
2. Please verify the directories of `sys_configs`. If there isn't any POSCAR for `01.model_devi` in one iteration, it may happen that you write the false path of `sys_configs`.
3. Correct format of JSON file.
4. The frames of one system should be larger than `batch_size` and `numb_test` in `default_training_param`. It happens that one iteration adds only a few structures and causes error in next iteration's training. In this condition, you may let `fp_task_min` be larger than `numb_test`.

8.7 Common Errors

(Errors are sorted alphabetically)

8.7.1 dargs.dargs.ArgumentTypeError: [at root location] key `xxx` gets wrong value type, requires but gets

Please check your parameters with DPGEN's Document. Maybe youhave superfluous parentheses in your parameter file.

8.7.2 Dargs: xxx is not allowed in strict mode.

Strict format check has been applied since version 0.10.7. To avoid misleading users, some older-version keys that are already ignored or absorbed into default settings are not allowed to be present. And the expected structure of the dictionary in the param.json also differs from those before version 0.10.7. This error will occur when format check finds older-fashion keys in the json file. Please try deleting or annotating these keys, or correspondingly modulate the json file. Example files in the newest format could be found in examples.

8.7.3 FileNotFoundError: [Errno 2] No such file or directory: '.../01.model_devi/graph.xxx.pb'

If you find this error occurs, please check your initial data. Your model will not be generated if the initial data is incorrect.

8.7.4 json.decoder.JSONDecodeError

Your `.json` file is incorrect. It may be a mistake in syntax or a missing comma.

8.7.5 RuntimeError: job:xxxxxxx failed 3 times

```
RuntimeError: job:xxxxxxx failed 3 times

......

RuntimeError: Meet errors will handle unexpected submission state.
Debug information: remote_root==xxxxxx
Debug information: submission_hash==xxxxxx
Please check the dirs and scripts in remote_root. The job information mentioned above␣
↪may help.
```

If a user finds an error like this, he or she is advised to check the files on the remote server. It shows that your job has failed 3 times, but has not shown the reason.

To find the reason, you can check the log on the remote root. For example, you can check train.log, which is generated by DeePMD-kit. It can tell you more details. If it doesn't help, you can manually run the `.sub` script, whose path is shown in `Debug information: remote_root==xxxxxx`

Some common reasons are as follows:

1. Two or more jobs are submitted manually or automatically at the same time, and their hash value collide. This bug will be fixed in dpdispatcher.
2. You may have something wrong in your input files, which causes the process to fail.

8.7.6 RuntimeError: find too many unsuccessfully terminated jobs.

The ratio of failed jobs is larger than ratio_failure. You can set a high value for ratio_failure or check if there is something wrong with your input files.

8.7.7 ValueError: Cannot load file containing picked data when allow_picked=False

Please ensure that you write the correct path of the dataset with no excess files.

CHAPTER

NINE

CONTRIBUTING GUIDE

9.1 Contributing Guide

The way to make contributions is through making pull requests(PR for short). After your PR is merged, the changes you make can be applied by other users.

Firstly, fork in DP-GEN repository. Then you can clone the repository, build a new branch, make changes and then make a pull request.

9.1.1 How to contribute to DP-GEN

Welcome to the repository of DP-GEN

DP-GEN adopts the same convention as other software in DeepModeling Community.

You can first refer to DeePMD-kit's Contributing guide and Developer guide.

You can also read relative chapters on Github Docs.

If you have no idea how to fix your problem or where to find the relative source code, please check *Code Structure* of the DP-GEN repository on this website.

Use command line

You can use git with the command line, or open the repository on Github Desktop. Here is a video as a demo of making changes to DP-GEN and publishing it with command line.

> If you have never used Github before, remember to generate your ssh key and configure the public key in Github Settings. If you can't configure your username and password, please use token. The explanation from Github see Github Blog: token authentication requirements for git operations. A discussion on StaskOverflow can solve this problem.

Use Github Desktop

Also, you can use Github Desktop to make PR. The following shows the steps to clone the repository and add your doc to tutorials. If it is your first time using Github, Open with Github Desktop is recommended. Github Desktop is a software, which can make your operations on branches visually.

After you clone it to your PC, you can open it with Github Desktop.

Firstly, create your new branch based on devel branch.

Secondly, add your doc to the certain directory in your local repository, and add its name into index.

Here is an example. Remember to add the filename of your doc into index!

Thirdly, select the changes that you what to push, and commit to it. Press "Publish branch" to push your origin repository to the remote branch.

Finally, you can check it on github and make a pull request. Press "Compare & pull request" to make a PR.

(Note: please commit pr to the devel branch)

9.1.2 How to contribute to DP-GEN tutorials and documents

Welcome to the documents of DP-GEN

- If you want to add the documentation of a toy model, simply put your file in the directory doc/toymodels/ and push;
- If you want to add a new directory for a new category of instructions, make a new directory and add it in doc/index.rst.

Also welcome to Tutorials repository You can find the structure of tutorials and preparations before writing a document in Writing Tips.

The latest page of DP-GEN Docs

Examples of contributions

- Example 1
- Example 2 (a simple one for beginner)

1. Push your doc

2. Add the directory in index.rst

3. Build and check it

As mentioned in "How to build the website to check if the modification works".

4. Make pull request to dpgen

9.1.3 Find how a parameter is used in the code

It is strongly recommended that you use the `find in files` function of Visual Studio software, `Search` function of Visual Studio Code, or similar functions of other software. Type in the name of the parameter you are looking for, and you will see where it is read in and used in the procedure. Of course, you can also search for the relevant code according to the above guide.

9.1.4 Want to modify a function?

If you have special requirements, you can make personalized modifications in the code corresponding to the function. If you think your modification can benefit the public, and it does not conflict with the current DP-GEN function; or if you fix a bug, please make a pull request to contribute the optimization to the DP-GEN repository.

9.1.5 DP-GEN dependencies

dpdispatcher and dpdata are dependencies of DP-GEN. dpdispatcher is related to task submission, monitoring and recovery, and dpdata is related to data processing. If you encounter an error and want to find the reason, please judge whether the problem comes from DP-GEN, dpdispatcher or dpdata according to the last line of `Traceback`.

9.1.6 About the update of the parameter file

You may have noticed that there are arginfo.py files in many folders. This is a file used to generate parameter documentation. If you add or modify a parameter in DP-GEN and intend to export it to the main repository, please sync your changes in arginfo.

9.1.7 Tips

1. Please try to submit a PR after finishing all the changes
2. Please briefly describe what you do with `git commit -m "<conclude-the-change-you-make>"`! "No description provided." will make the maintainer feel confused.
3. It is not recommended to make changes directly in the `devel` branch. It is recommended to pull a branch from devel: `git checkout -b <new-branch-name>`
4. When switching branches, remember to check if you want to bring the changes to the next branch!
5. Please fix the errors reported by the unit test. You can firstly test on your local machine before pushing commits. Hint: The way to test the code is to go from the main directory to the tests directory, and use the command `python3 -m unittest`. You can watch the demo video for review. Sometimes you may fail unit tests due to your local circumstance. You can check whether the error reported is related to the part you modified to eliminate this problem. After submitting, as long as there is a green check mark after the PR title on the webpage, it means that the test has been passed.
6. Pay attention to whether there are comments under your PR. If there is a change request, you need to check and modify the code. If there are conflicts, you need to solve them manually.

After successfully making a PR, developers will check it and give comments. It will be merged after everything done. Then CONGRATULATIONS! You become a first-time contributor to DP-GEN!

How to get help from the community

CHAPTER

TEN

DP-GEN API

10.1 dpgen package

dpgen.**info**()

10.1.1 Subpackages

dpgen.auto_test package

Subpackages

dpgen.auto_test.lib package

Submodules

dpgen.auto_test.lib.BatchJob module

class dpgen.auto_test.lib.BatchJob.**BatchJob**(*job_dir=''*, *job_script=''*, *job_finish_tag='tag_finished'*, *job_id_file='tag_jobid'*)

Bases: object

Abstract class of a batch job It submit a job (leave the id in file tag_jobid) It check the status of the job (return JobStatus) NOTICE: I assume that when a job finishes, a tag file named tag_finished should be touched by the user. TYPICAL USAGE: job = DERIVED_BatchJob (dir, script) job.submit () stat = job.check_status ()

Methods

submit_command()	submission is $ [command] [script]

check_status	
get_job_id	
submit	

check_status()

get_job_id()

submit()

submit_command()

submission is $ [command] [script]

class dpgen.auto_test.lib.BatchJob.**JobStatus**(*value*)

Bases: Enum

An enumeration.

finished = 5

running = 3

terminated = 4

unknow = 100

unsubmitted = 1

waiting = 2

dpgen.auto_test.lib.RemoteJob module

class dpgen.auto_test.lib.RemoteJob.**CloudMachineJob**(*ssh_session*, *local_root*)

Bases: *RemoteJob*

Methods

block_call	
block_checkcall	
check_status	
clean	
download	
get_job_root	
submit	
upload	

check_status()

submit(*job_dirs*, *cmd*, *args=None*, *resources=None*)

class dpgen.auto_test.lib.RemoteJob.**JobStatus**(*value*)

Bases: Enum

An enumeration.

finished = 5

running = 3

terminated = 4

unknow = 100

unsubmitted = 1

waiting = 2

class dpgen.auto_test.lib.RemoteJob.**PBSJob**(*ssh_session*, *local_root*)

Bases: *RemoteJob*

Methods

block_call	
block_checkcall	
check_status	
clean	
download	
get_job_root	
submit	
upload	

check_status()

submit(*job_dirs*, *cmd*, *args=None*, *resources=None*)

class dpgen.auto_test.lib.RemoteJob.**RemoteJob**(*ssh_session*, *local_root*)

Bases: object

Methods

block_call	
block_checkcall	
clean	
download	
get_job_root	
upload	

block_call(*cmd*)

block_checkcall(*cmd*)

clean()

download(*job_dirs*, *remote_down_files*)

get_job_root()

upload(*job_dirs*, *local_up_files*, *dereference=True*)

class dpgen.auto_test.lib.RemoteJob.**SSHSession**(*jdata*)

Bases: object

Methods

close	
get_session_root	
get_ssh_client	

close()

get_session_root()

get_ssh_client()

class dpgen.auto_test.lib.RemoteJob.**SlurmJob**(*ssh_session*, *local_root*)

Bases: *RemoteJob*

Methods

block_call	
block_checkcall	
check_status	
clean	
download	
get_job_root	
submit	
upload	

check_status()

submit(*job_dirs*, *cmd*, *args=None*, *resources=None*)

dpgen.auto_test.lib.SlurmJob module

class dpgen.auto_test.lib.SlurmJob.**SlurmJob**(*job_dir=''*, *job_script=''*, *job_finish_tag='tag_finished'*, *job_id_file='tag_jobid'*)

Bases: *BatchJob*

Methods

submit_command()	submission is $ [command] [script]

check_status	
get_job_id	
submit	

check_status()

submit_command()

submission is $ [command] [script]

dpgen.auto_test.lib.abacus module

dpgen.auto_test.lib.abacus.**check_finished**(*fname*)

dpgen.auto_test.lib.abacus.**check_stru_fixed**(*struf*, *fixed*)

dpgen.auto_test.lib.abacus.**final_stru**(*abacus_path*)

dpgen.auto_test.lib.abacus.**make_kspacing_kpt**(*struf*, *kspacing*)

dpgen.auto_test.lib.abacus.**modify_stru_path**(*strucf*, *tpath*)

dpgen.auto_test.lib.abacus.**poscar2stru**(*poscar*, *inter_param*, *stru*)

- poscar: POSCAR for input
- **inter_param: dictionary of 'interaction' from param.json**
  some key words for ABACUS are:
  - atom_masses: a dictionary of atoms' masses
  - orb_files: a dictionary of orbital files
  - deepks_desc: a string of deepks descriptor file
- stru: output filename, usally is 'STRU'

dpgen.auto_test.lib.abacus.**stru2Structure**(*struf*)

dpgen.auto_test.lib.abacus.**stru_fix_atom**(*struf*, *fix_atom=[True, True, True]*)

… ATOMIC_POSITIONS Cartesian #Cartesian(Unit is LATTICE_CONSTANT) Si #Name of element 0.0 #Magnetic for this element. 2 #Number of atoms 0.00 0.00 0.00 0 0 0 #x,y,z, move_x, move_y, move_z 0.25 0.25 0.25 0 0 0

dpgen.auto_test.lib.abacus.**stru_scale**(*stru_in*, *stru_out*, *scale*)

dpgen.auto_test.lib.abacus.**write_input**(*inputf*, *inputdict*)

dpgen.auto_test.lib.abacus.**write_kpt**(*kptf*, *kptlist*)

dpgen.auto_test.lib.crys module

dpgen.auto_test.lib.crys.**bcc**(*ele_name='ele'*, *a=3.2144871302356037*)

dpgen.auto_test.lib.crys.**dhcp**(*ele_name='ele'*, *a=2.863782463805517*, *c=9.353074360871936*)

dpgen.auto_test.lib.crys.**diamond**(*ele_name='ele'*, *a=2.551340126037118*)

dpgen.auto_test.lib.crys.**fcc**(*ele_name='ele'*, *a=4.05*)

dpgen.auto_test.lib.crys.**fcc1**(*ele_name='ele'*, *a=4.05*)

dpgen.auto_test.lib.crys.**hcp**(*ele_name='ele'*, *a=2.863782463805517*, *c=4.676537180435968*)

dpgen.auto_test.lib.crys.**sc**(*ele_name='ele'*, *a=2.551340126037118*)

dpgen.auto_test.lib.lammps module

dpgen.auto_test.lib.lammps.**apply_type_map**(*conf_file*, *deepmd_type_map*, *ptypes*)

apply type map. conf_file: conf file converted from POSCAR deepmd_type_map: deepmd atom type map ptypes: atom types defined in POSCAR

dpgen.auto_test.lib.lammps.**check_finished**(*fname*)

dpgen.auto_test.lib.lammps.**check_finished_new**(*fname*, *keyword*)

dpgen.auto_test.lib.lammps.**cvt_lammps_conf**(*fin*, *fout*, *type_map*, *ofmt='lammps/data'*)

Format convert from fin to fout, specify the output format by ofmt Imcomplete situation

dpgen.auto_test.lib.lammps.**element_list**(*type_map*)

dpgen.auto_test.lib.lammps.**get_base_area**(*log*)

get base area

dpgen.auto_test.lib.lammps.**get_nev**(*log*)

get natoms, energy_per_atom and volume_per_atom from lammps log

dpgen.auto_test.lib.lammps.**get_stress**(*log*)

get stress from lammps log

dpgen.auto_test.lib.lammps.**inter_deepmd**(*param*)

dpgen.auto_test.lib.lammps.**inter_eam_alloy**(*param*)

dpgen.auto_test.lib.lammps.**inter_eam_fs**(*param*)

dpgen.auto_test.lib.lammps.**inter_meam**(*param*)

dpgen.auto_test.lib.lammps.**make_lammps_elastic**(*conf*, *type_map*, *interaction*, *param*, *etol=0*, *ftol=1e-10*, *maxiter=5000*, *maxeval=500000*)

dpgen.auto_test.lib.lammps.**make_lammps_equi**(*conf*, *type_map*, *interaction*, *param*, *etol=0*, *ftol=1e-10*, *maxiter=5000*, *maxeval=500000*, *change_box=True*)

dpgen.auto_test.lib.lammps.**make_lammps_eval**(*conf*, *type_map*, *interaction*, *param*)

dpgen.auto_test.lib.lammps.**make_lammps_phonon**(*conf*, *masses*, *interaction*, *param*, *etol=0*, *ftol=1e-10*, *maxiter=5000*, *maxeval=500000*)

make lammps input for elastic calculation

dpgen.auto_test.lib.lammps.**make_lammps_press_relax**(*conf*, *type_map*, *scale2equi*, *interaction*, *param*, *B0=70*, *bp=0*, *etol=0*, *ftol=1e-10*, *maxiter=5000*, *maxeval=500000*)

dpgen.auto_test.lib.lammps.**poscar_from_last_dump**(*dump*, *poscar_out*, *deepmd_type_map*)

get poscar from the last frame of a lammps MD traj (dump format)

dpgen.auto_test.lib.lmp module

dpgen.auto_test.lib.lmp.**box2lmpbox**(*orig*, *box*)

dpgen.auto_test.lib.lmp.**from_system_data**(*system*)

dpgen.auto_test.lib.lmp.**get_atoms**(*lines*)

dpgen.auto_test.lib.lmp.**get_atype**(*lines*)

dpgen.auto_test.lib.lmp.**get_lmpbox**(*lines*)

dpgen.auto_test.lib.lmp.**get_natoms**(*lines*)

dpgen.auto_test.lib.lmp.**get_natoms_vec**(*lines*)

dpgen.auto_test.lib.lmp.**get_natomtypes**(*lines*)

dpgen.auto_test.lib.lmp.**get_posi**(*lines*)

dpgen.auto_test.lib.lmp.**lmpbox2box**(*lohi*, *tilt*)

dpgen.auto_test.lib.lmp.**system_data**(*lines*)

dpgen.auto_test.lib.lmp.**to_system_data**(*lines*)

dpgen.auto_test.lib.mfp_eosfit module

dpgen.auto_test.lib.mfp_eosfit.**BM4**(*vol*, *pars*)
Birch-Murnaghan 4 pars equation from PRB 70, 224107, 3-order

dpgen.auto_test.lib.mfp_eosfit.**BM5**(*vol*, *pars*)
Birch-Murnaghan 5 pars equation from PRB 70, 224107, 4-Order

dpgen.auto_test.lib.mfp_eosfit.**LOG4**(*vol*, *pars*)
Natrual strain (Poirier-Tarantola)EOS with 4 paramters Seems only work in near-equillibrium range.

dpgen.auto_test.lib.mfp_eosfit.**LOG5**(*vol*, *parameters*)
Natrual strain (Poirier-Tarantola)EOS with 5 paramters

dpgen.auto_test.lib.mfp_eosfit.**Li4p**(*V*, *parameters*)
Li JH, APL, 87, 194111 (2005)

dpgen.auto_test.lib.mfp_eosfit.**SJX_5p**(*vol*, *par*)
SJX_5p's five parameters EOS, Physica B: Condens Mater, 2011, 406: 1276-1282

dpgen.auto_test.lib.mfp_eosfit.**SJX_v2**(*vol*, *par*)
Sun Jiuxun, et al. J phys Chem Solids, 2005, 66: 773-782. They said it is satified for the limiting condition at high pressure.

dpgen.auto_test.lib.mfp_eosfit.**TEOS**(*v*, *par*)
Holland, et al, Journal of Metamorphic Geology, 2011, 29(3): 333-383 Modified Tait equation of Huang & Chow

dpgen.auto_test.lib.mfp_eosfit.**birch**(*v*, *parameters*)
From Intermetallic compounds: Principles and Practice, Vol. I: Princples Chapter 9 pages 195-210 by M. Mehl. B. Klein, D. Papaconstantopoulos paper downloaded from Web

case where n=0

dpgen.auto_test.lib.mfp_eosfit.**calc_props_BM4**(*pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_LOG4**(*pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_SJX_5p**(*par*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_mBM4**(*pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_mBM4poly**(*pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_mBM5poly**(*pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_morse**(*pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_morse_6p**(*par*)

dpgen.auto_test.lib.mfp_eosfit.**calc_props_vinet**(*pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_v0_mBM4poly**(*x*, *pars*)

dpgen.auto_test.lib.mfp_eosfit.**calc_v0_mBM5poly**(*x*, *pars*)

dpgen.auto_test.lib.mfp_eosfit.**ext_splint**(*xp*, *yp*, *order=3*, *method='unispl'*)

dpgen.auto_test.lib.mfp_eosfit.**ext_vec**(*func*, *fin*, *p0*, *fs*, *fe*, *vols=None*, *vole=None*, *ndata=101*, *refit=0*, *show_fig=False*)

extrapolate the data points for E-V based on the fitted parameters in small or very large volume range.

dpgen.auto_test.lib.mfp_eosfit.**ext_velp**(*fin*, *fstart*, *fend*, *vols*, *vole*, *ndata*, *order=3*, *method='unispl'*, *fout='ext_velp.dat'*, *show_fig=False*)

extrapolate the lattice parameters based on input data

dpgen.auto_test.lib.mfp_eosfit.**get_eos_list**()

dpgen.auto_test.lib.mfp_eosfit.**get_eos_list_3p**()

dpgen.auto_test.lib.mfp_eosfit.**get_eos_list_4p**()

dpgen.auto_test.lib.mfp_eosfit.**get_eos_list_5p**()

dpgen.auto_test.lib.mfp_eosfit.**get_eos_list_6p**()

dpgen.auto_test.lib.mfp_eosfit.**init_guess**(*fin*)

dpgen.auto_test.lib.mfp_eosfit.**lsqfit_eos**(*func*, *fin*, *par*, *fstart*, *fend*, *show_fig=False*, *fout='EoSfit.out'*, *refit=-1*)

dpgen.auto_test.lib.mfp_eosfit.**mBM4**(*vol*, *pars*)

Birch-Murnaghan 4 pars equation from PRB 70, 224107, 3-order BM

dpgen.auto_test.lib.mfp_eosfit.**mBM4poly**(*vol*, *parameters*)

modified BM5 EOS, Shang SL comput mater sci, 2010: 1040-1048, original expressions.

dpgen.auto_test.lib.mfp_eosfit.**mBM5**(*vol*, *pars*)

modified BM5 EOS, Shang SL comput mater sci, 2010: 1040-1048

dpgen.auto_test.lib.mfp_eosfit.**mBM5poly**(*vol*, *pars*)

modified BM5 EOS, Shang SL comput mater sci, 2010: 1040-1048, original expressions.

dpgen.auto_test.lib.mfp_eosfit.**mie**(*v*, *p*)

Mie model for song's FVT

dpgen.auto_test.lib.mfp_eosfit.**mie_simple**(*v*, *p*)

Mie_simple model for song's FVT

dpgen.auto_test.lib.mfp_eosfit.**morse**(*v*, *pars*)

Reproduce from ShunliShang's matlab script.

dpgen.auto_test.lib.mfp_eosfit.**morse_3p**(*volume*, *p*)

morse_AB EOS formula from Song's FVT souces A= 0.5*B

dpgen.auto_test.lib.mfp_eosfit.**morse_6p**(*vol*, *par*)

Generalized Morse EOS proposed by Qin, see: Qin et al. Phys Rev B, 2008, 78, 214108. Qin et al. Phys Rev B, 2008, 77, 220103(R).

dpgen.auto_test.lib.mfp_eosfit.**morse_AB**(*volume*, *p*)

morse_AB EOS formula from Song's FVT souces

dpgen.auto_test.lib.mfp_eosfit.**murnaghan**(*vol*, *pars*)

Four-parameters murnaghan EOS. From PRB 28,5480 (1983)

dpgen.auto_test.lib.mfp_eosfit.**parse_argument**()

dpgen.auto_test.lib.mfp_eosfit.**rBM4**(*vol*, *pars*)

Implementions as Alberto Otero-de-la-Roza, i.e. rBM4 is used here Comput Physics Comm, 2011, 182: 1708-1720

dpgen.auto_test.lib.mfp_eosfit.**rBM4_pv**(*vol*, *pars*)

Implementions as Alberto Otero-de-la-Roza, i.e. rBM4 is used here Comput Physics Comm, 2011, 182: 1708-1720 Fit for V-P relations

dpgen.auto_test.lib.mfp_eosfit.**rBM5**(*vol*, *pars*)

Implementions as Alberto Otero-de-la-Roza, i.e. rBM5 is used here Comput Physics Comm, 2011, 182: 1708-1720

dpgen.auto_test.lib.mfp_eosfit.**rBM5_pv**(*vol*, *pars*)

Implementions as Alberto Otero-de-la-Roza, i.e. rBM5 is used here Comput Physics Comm, 2011, 182: 1708-1720 Fit for V-P relations

dpgen.auto_test.lib.mfp_eosfit.**rPT4**(*vol*, *pars*)

Natrual strain EOS with 4 paramters Seems only work in near-equillibrium range. Implementions as Alberto Otero-de-la-Roza, i.e. rPT4 is used here Comput Physics Comm, 2011, 182: 1708-1720, in their article, labeled as PT3 (3-order), however, we mention it as rPT4 for 4-parameters EOS.

dpgen.auto_test.lib.mfp_eosfit.**rPT4_pv**(*vol*, *pars*)

Natrual strain (Poirier-Tarantola)EOS with 4 paramters Seems only work in near-equillibrium range. Implementions as Alberto Otero-de-la-Roza, i.e. rPT4 is used here Comput Physics Comm, 2011, 182: 1708-1720, in their article, labeled as PT3 (3-order), however, we mention it as rPT4 for 4-parameters EOS.

dpgen.auto_test.lib.mfp_eosfit.**rPT5**(*vol*, *pars*)

Natrual strain EOS with 4 paramters Seems only work in near-equillibrium range. Implementions as Alberto Otero-de-la-Roza, i.e. rPT5 is used here Comput Physics Comm, 2011, 182: 1708-1720, in their article, labeled as PT3 (3-order), however, we mention it as rPT5 for 4-parameters EOS.

dpgen.auto_test.lib.mfp_eosfit.**rPT5_pv**(*vol*, *pars*)

Natrual strain (Poirier-Tarantola)EOS with 5 paramters Implementions as Alberto Otero-de-la-Roza, i.e. rPT5 is used here Comput Physics Comm, 2011, 182: 1708-1720, in their article, labeled as PT3 (3-order), however, we mention it as rPT5 for 4-parameters EOS.

dpgen.auto_test.lib.mfp_eosfit.**read_ve**(*fin*)

dpgen.auto_test.lib.mfp_eosfit.**read_velp**(*fin*, *fstart*, *fend*)

dpgen.auto_test.lib.mfp_eosfit.**read_vlp**(*fin*, *fstart*, *fend*)

dpgen.auto_test.lib.mfp_eosfit.**repro_ve**(*func*, *vol_i*, *p*)

dpgen.auto_test.lib.mfp_eosfit.**repro_vp**(*func*, *vol_i*, *pars*)

dpgen.auto_test.lib.mfp_eosfit.**res_BM4**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_BM5**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_LOG4**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_LOG5**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_Li4p**(*p*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_SJX_5p**(*p*, *e*, *v*)

dpgen.auto_test.lib.mfp_eosfit.**res_SJX_v2**(*p*, *e*, *v*)

dpgen.auto_test.lib.mfp_eosfit.**res_TEOS**(*p*, *e*, *v*)

dpgen.auto_test.lib.mfp_eosfit.**res_birch**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_mBM4**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_mBM4poly**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_mBM5**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_mBM5poly**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_mie**(*p*, *e*, *v*)

dpgen.auto_test.lib.mfp_eosfit.**res_mie_simple**(*p*, *e*, *v*)

dpgen.auto_test.lib.mfp_eosfit.**res_morse**(*p*, *en*, *volume*)

dpgen.auto_test.lib.mfp_eosfit.**res_morse_3p**(*p*, *en*, *volume*)

dpgen.auto_test.lib.mfp_eosfit.**res_morse_6p**(*p*, *en*, *volume*)

dpgen.auto_test.lib.mfp_eosfit.**res_morse_AB**(*p*, *en*, *volume*)

dpgen.auto_test.lib.mfp_eosfit.**res_murnaghan**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rBM4**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rBM4_pv**(*par*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rBM5**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rBM5_pv**(*par*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rPT4**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rPT4_pv**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rPT5**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_rPT5_pv**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_universal**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_vinet**(*pars*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**res_vinet_pv**(*par*, *y*, *x*)

dpgen.auto_test.lib.mfp_eosfit.**universal**(*vol*, *parameters*)

Universal equation of state(Vinet P et al., J. Phys.: Condens. Matter 1, p1941 (1989))

dpgen.auto_test.lib.mfp_eosfit.**vinet**(*vol*, *pars*)

Vinet equation from PRB 70, 224107 Following, Shang Shunli et al., comput mater sci, 2010: 1040-1048, original expressions.

dpgen.auto_test.lib.mfp_eosfit.**vinet_pv**(*vol*, *pars*)

dpgen.auto_test.lib.pwscf module

dpgen.auto_test.lib.pwscf.**make_pwscf_input**(*sys_data*, *fp_pp_files*, *fp_params*)

dpgen.auto_test.lib.siesta module

dpgen.auto_test.lib.siesta.**make_siesta_input**(*sys_data*, *fp_pp_files*, *fp_params*)

dpgen.auto_test.lib.util module

dpgen.auto_test.lib.util.**collect_task**(*all_task*, *task_type*)

dpgen.auto_test.lib.util.**get_machine_info**(*mdata*, *task_type*)

dpgen.auto_test.lib.util.**insert_data**(*task*, *task_type*, *username*, *file_name*)

dpgen.auto_test.lib.util.**make_work_path**(*jdata*, *task*, *reprod_opt*, *static*, *user*)

dpgen.auto_test.lib.util.**voigt_to_stress**(*inpt*)

dpgen.auto_test.lib.utils module

dpgen.auto_test.lib.utils.**cmd_append_log**(*cmd*, *log_file*)

dpgen.auto_test.lib.utils.**copy_file_list**(*file_list*, *from_path*, *to_path*)

dpgen.auto_test.lib.utils.**create_path**(*path*)

dpgen.auto_test.lib.utils.**log_iter**(*task*, *ii*, *jj*)

dpgen.auto_test.lib.utils.**log_task**(*message*)

dpgen.auto_test.lib.utils.**make_iter_name**(*iter_index*)

dpgen.auto_test.lib.utils.**record_iter**(*record*, *confs*, *ii*, *jj*)

dpgen.auto_test.lib.utils.**repeat_to_length**(*string_to_expand*, *length*)

dpgen.auto_test.lib.utils.**replace**(*file_name*, *pattern*, *subst*)

dpgen.auto_test.lib.vasp module

exception dpgen.auto_test.lib.vasp.**OutcarItemError**

Bases: Exception

dpgen.auto_test.lib.vasp.**check_finished**(*fname*)

dpgen.auto_test.lib.vasp.**get_boxes**(*fname*)

dpgen.auto_test.lib.vasp.**get_energies**(*fname*)

dpgen.auto_test.lib.vasp.**get_nev**(*fname*)

dpgen.auto_test.lib.vasp.**get_poscar_natoms**(*fname*)

dpgen.auto_test.lib.vasp.**get_poscar_types**(*fname*)

dpgen.auto_test.lib.vasp.**get_stress**(*fname*)

dpgen.auto_test.lib.vasp.**make_kspacing_kpoints**(*poscar*, *kspacing*, *kgamma*)

dpgen.auto_test.lib.vasp.**make_vasp_kpoints**(*kpoints*, *kgamma=False*)

dpgen.auto_test.lib.vasp.**make_vasp_kpoints_from_incar**(*work_dir*, *jdata*)

dpgen.auto_test.lib.vasp.**make_vasp_phonon_incar**(*ecut*, *ediff*, *npar*, *kpar*, *kspacing=0.5*, *kgamma=True*, *ismear=1*, *sigma=0.2*)

dpgen.auto_test.lib.vasp.**make_vasp_relax_incar**(*ecut*, *ediff*, *relax_ion*, *relax_shape*, *relax_volume*, *npar*, *kpar*, *kspacing=0.5*, *kgamma=True*, *ismear=1*, *sigma=0.22*)

dpgen.auto_test.lib.vasp.**make_vasp_static_incar**(*ecut*, *ediff*, *npar*, *kpar*, *kspacing=0.5*, *kgamma=True*, *ismear=1*, *sigma=0.2*)

dpgen.auto_test.lib.vasp.**perturb_xz**(*poscar_in*, *poscar_out*, *pert=0.01*)

dpgen.auto_test.lib.vasp.**poscar_natoms**(*poscar_in*)

dpgen.auto_test.lib.vasp.**poscar_scale**(*poscar_in*, *poscar_out*, *scale*)

dpgen.auto_test.lib.vasp.**poscar_vol**(*poscar_in*)

dpgen.auto_test.lib.vasp.**reciprocal_box**(*box*)

dpgen.auto_test.lib.vasp.**regulate_poscar**(*poscar_in*, *poscar_out*)

dpgen.auto_test.lib.vasp.**sort_poscar**(*poscar_in*, *poscar_out*, *new_names*)

Submodules

dpgen.auto_test.ABACUS module

class dpgen.auto_test.ABACUS.**ABACUS**(*inter_parameter*, *path_to_poscar*)

Bases: *Task*

Methods

backward_files([property_type])	staticmethod(function) -> method
compute(output_dir)	Compute output of the task.
forward_common_files([property_type])	staticmethod(function) -> method
forward_files([property_type])	staticmethod(function) -> method
make_input_file(output_dir, task_type, ...)	Prepare input files for a computational task For example, the VASP prepares INCAR.
make_potential_files(output_dir)	Prepare potential files for a computational task.

modify_input	

backward_files(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

compute(*output_dir*)

Compute output of the task. IMPORTANT: The output configuration should be converted and stored in a CONTCAR file.

Parameters

output_dir

[str] The directory storing the input and output files.

Returns

result_dict: dict

A dict that storing the result. For example: { “energy”: xxx, “force”: [xxx] }

`forward_common_files`(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

`forward_files`(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

`make_input_file`(*output_dir*, *task_type*, *task_param*)

Prepare input files for a computational task For example, the VASP prepares INCAR. LAMMPS (including DeePMD, MEAM...) prepares in.lammps.

Parameters

output_dir

[str] The directory storing the input files.

task_type

[str] Can be - “relaxation:”: structure relaxation - “static”: static computation calculates the energy, force... of a strcture

task_parame: dict

The parameters of the task. For example the VASP interaction can be provided with { “ediff”: 1e-6, “ediffg”: 1e-5 }

`make_potential_files`(*output_dir*)

Prepare potential files for a computational task. For example, the VASP prepares POTCAR. DeePMD prepares frozen model(s). IMPORTANT: Interaction should be stored in output_dir/inter.json

Parameters

output_dir

[str] The directory storing the potential files.

Outputs

——-

inter.json: output file

The task information is stored in *output_dir/inter.json*

modify_input(*incar*, *x*, *y*)

dpgen.auto_test.EOS module

class dpgen.auto_test.EOS.**EOS**(*parameter*, *inter_param=None*)

Bases: *Property*

Methods

compute(output_file, print_file, path_to_work)	Postprocess the finished tasks to compute the property.
make_confs(path_to_work, path_to_equi[, refine])	Make configurations needed to compute the property.
post_process(task_list)	post_process the KPOINTS file in elastic.
task_param()	Return the parameter of each computational task, for example, {'ediffg': 1e-4}
task_type()	Return the type of each computational task, for example, 'relaxation', 'static'....

make_confs(*path_to_work*, *path_to_equi*, *refine=False*)

Make configurations needed to compute the property. The tasks directory will be named as path_to_work/task.xxxxxx IMPORTANT: handel the case when the directory exists.

Parameters

path_to_work

[str] The path where the tasks for the property are located

path_to_equi

[str] -refine == False: The path to the directory that equilibrated the configuration.
-refine == True: The path to the directory that has property confs.

refine: str

To refine existing property confs or generate property confs from a equilibrated conf

Returns

task_list: list of str

The list of task directories.

post_process(*task_list*)

post_process the KPOINTS file in elastic.

task_param()

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

task_type()

Return the type of each computational task, for example, 'relaxation', 'static'....

dpgen.auto_test.Elastic module

class dpgen.auto_test.Elastic.**Elastic**(*parameter*, *inter_param=None*)

Bases: *Property*

Methods

compute(output_file, print_file, path_to_work)	Postprocess the finished tasks to compute the property.
make_confs(path_to_work, path_to_equi[, refine])	Make configurations needed to compute the property.
post_process(task_list)	post_process the KPOINTS file in elastic.
task_param()	Return the parameter of each computational task, for example, {'ediffg': 1e-4}
task_type()	Return the type of each computational task, for example, 'relaxation', 'static'....

make_confs(*path_to_work*, *path_to_equi*, *refine=False*)

Make configurations needed to compute the property. The tasks directory will be named as path_to_work/task.xxxxxx IMPORTANT: handel the case when the directory exists.

Parameters

path_to_work

[str] The path where the tasks for the property are located

path_to_equi

[str] -refine == False: The path to the directory that equilibrated the configuration. -refine == True: The path to the directory that has property confs.

refine: str

To refine existing property confs or generate property confs from a equilibrated conf

Returns

task_list: list of str

The list of task directories.

post_process(*task_list*)

post_process the KPOINTS file in elastic.

task_param()

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

task_type()

Return the type of each computational task, for example, 'relaxation', 'static'....

dpgen.auto_test.Gamma module

class dpgen.auto_test.Gamma.**Gamma**(*parameter*, *inter_param=None*)

Bases: *Property*

Calculation of common gamma lines for bcc and fcc

Methods

compute(output_file, print_file, path_to_work)	Postprocess the finished tasks to compute the property.
make_confs(path_to_work, path_to_equi[, refine])	Make configurations needed to compute the property.
post_process(task_list)	post_process the KPOINTS file in elastic.
task_param()	Return the parameter of each computational task, for example, {'ediffg': 1e-4}
task_type()	Return the type of each computational task, for example, 'relaxation', 'static'....

centralize_slab	
return_direction	

static centralize_slab(*slab*) → None

make_confs(*path_to_work*, *path_to_equi*, *refine=False*)

Make configurations needed to compute the property. The tasks directory will be named as path_to_work/task.xxxxxx IMPORTANT: handel the case when the directory exists.

Parameters

path_to_work
[str] The path where the tasks for the property are located

path_to_equi
[str] -refine == False: The path to the directory that equilibrated the configuration. -refine == True: The path to the directory that has property confs.

refine: str
To refine existing property confs or generate property confs from a equilibrated conf

Returns

task_list: list of str
The list of task directories.

post_process(*task_list*)

post_process the KPOINTS file in elastic.

return_direction()

task_param()

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

task_type()

Return the type of each computational task, for example, 'relaxation', 'static'....

dpgen.auto_test.Interstitial module

class dpgen.auto_test.Interstitial.**Interstitial**(*parameter*, *inter_param=None*)

Bases: *Property*

Methods

compute(output_file, print_file, path_to_work)	Postprocess the finished tasks to compute the property.
make_confs(path_to_work, path_to_equi[, refine])	Make configurations needed to compute the property.
post_process(task_list)	post_process the KPOINTS file in elastic.
task_param()	Return the parameter of each computational task, for example, {'ediffg': 1e-4}
task_type()	Return the type of each computational task, for example, 'relaxation', 'static'....

make_confs(*path_to_work*, *path_to_equi*, *refine=False*)

Make configurations needed to compute the property. The tasks directory will be named as path_to_work/task.xxxxxx IMPORTANT: handel the case when the directory exists.

Parameters

path_to_work

[str] The path where the tasks for the property are located

path_to_equi

[str] -refine == False: The path to the directory that equilibrated the configuration. -refine == True: The path to the directory that has property confs.

refine: str

To refine existing property confs or generate property confs from a equilibrated conf

Returns

task_list: list of str

The list of task directories.

post_process(*task_list*)

post_process the KPOINTS file in elastic.

task_param()

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

task_type()

Return the type of each computational task, for example, 'relaxation', 'static'....

dpgen.auto_test.Lammps module

class dpgen.auto_test.Lammps.**Lammps**(*inter_parameter*, *path_to_poscar*)

Bases: *Task*

Methods

backward_files([property_type])	staticmethod(function) -> method
compute(output_dir)	Compute output of the task.
forward_common_files([property_type])	staticmethod(function) -> method
forward_files([property_type])	staticmethod(function) -> method
make_input_file(output_dir, task_type, ...)	Prepare input files for a computational task For example, the VASP prepares INCAR.
make_potential_files(output_dir)	Prepare potential files for a computational task.

set_inter_type_func	
set_model_param	

backward_files(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

compute(*output_dir*)

Compute output of the task. IMPORTANT: The output configuration should be converted and stored in a CONTCAR file.

Parameters

output_dir

[str] The directory storing the input and output files.

Returns

result_dict: dict

A dict that storing the result. For example: { “energy”: xxx, “force”: [xxx] }

forward_common_files(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

forward_files(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

`make_input_file`(*output_dir*, *task_type*, *task_param*)

Prepare input files for a computational task For example, the VASP prepares INCAR. LAMMPS (including DeePMD, MEAM...) prepares in.lammps.

Parameters

output_dir

[str] The directory storing the input files.

task_type

[str] Can be - "relaxation:": structure relaxation - "static": static computation calculates the energy, force... of a strcture

task_parame: dict

The parameters of the task. For example the VASP interaction can be provided with { "ediff": 1e-6, "ediffg": 1e-5 }

`make_potential_files`(*output_dir*)

Prepare potential files for a computational task. For example, the VASP prepares POTCAR. DeePMD prepares frozen model(s). IMPORTANT: Interaction should be stored in output_dir/inter.json

Parameters

output_dir

[str] The directory storing the potential files.

Outputs

——-

inter.json: output file

The task information is stored in *output_dir/inter.json*

`set_inter_type_func`()

`set_model_param`()

dpgen.auto_test.Property module

`class` dpgen.auto_test.Property.`Property`(*parameter*)

Bases: ABC

Attributes

task_param

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

task_type

Return the type of each computational task, for example, 'relaxation', 'static'....

Methods

compute(output_file, print_file, path_to_work)	Postprocess the finished tasks to compute the property.
make_confs(path_to_work, path_to_equi[, refine])	Make configurations needed to compute the property.
post_process(task_list)	post_process the KPOINTS file in elastic.

compute(*output_file*, *print_file*, *path_to_work*)

Postprocess the finished tasks to compute the property. Output the result to a json database

Parameters

output_file:
The file to output the property in json format

print_file:
The file to output the property in txt format

path_to_work:
The working directory where the computational tasks locate.

abstract make_confs(*path_to_work*, *path_to_equi*, *refine=False*)

Make configurations needed to compute the property. The tasks directory will be named as path_to_work/task.xxxxxx IMPORTANT: handel the case when the directory exists.

Parameters

path_to_work
[str] The path where the tasks for the property are located

path_to_equi
[str] -refine == False: The path to the directory that equilibrated the configuration.
-refine == True: The path to the directory that has property confs.

refine: str
To refine existing property confs or generate property confs from a equilibrated conf

Returns

task_list: list of str
The list of task directories.

abstract post_process(*task_list*)

post_process the KPOINTS file in elastic.

abstract property task_param

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

abstract property task_type

Return the type of each computational task, for example, 'relaxation', 'static'....

dpgen.auto_test.Surface module

class dpgen.auto_test.Surface.**Surface**(*parameter*, *inter_param=None*)

Bases: *Property*

Methods

compute(output_file, print_file, path_to_work)	Postprocess the finished tasks to compute the property.
make_confs(path_to_work, path_to_equi[, refine])	Make configurations needed to compute the property.
post_process(task_list)	post_process the KPOINTS file in elastic.
task_param()	Return the parameter of each computational task, for example, {'ediffg': 1e-4}
task_type()	Return the type of each computational task, for example, 'relaxation', 'static'....

make_confs(*path_to_work*, *path_to_equi*, *refine=False*)

Make configurations needed to compute the property. The tasks directory will be named as path_to_work/task.xxxxxx IMPORTANT: handel the case when the directory exists.

Parameters

path_to_work

[str] The path where the tasks for the property are located

path_to_equi

[str] -refine == False: The path to the directory that equilibrated the configuration. -refine == True: The path to the directory that has property confs.

refine: str

To refine existing property confs or generate property confs from a equilibrated conf

Returns

task_list: list of str

The list of task directories.

post_process(*task_list*)

post_process the KPOINTS file in elastic.

task_param()

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

task_type()

Return the type of each computational task, for example, 'relaxation', 'static'....

dpgen.auto_test.Task module

class dpgen.auto_test.Task.**Task**(*inter_parameter*, *path_to_poscar*)

Bases: ABC

Attributes

backward_files

staticmethod(function) -> method

forward_common_files

staticmethod(function) -> method

forward_files

staticmethod(function) -> method

Methods

compute(output_dir)	Compute output of the task.
make_input_file(output_dir, task_type, ...)	Prepare input files for a computational task For example, the VASP prepares INCAR.
make_potential_files(output_dir)	Prepare potential files for a computational task.

abstract property backward_files

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

abstract compute(*output_dir*)

Compute output of the task. IMPORTANT: The output configuration should be converted and stored in a CONTCAR file.

Parameters

output_dir

[str] The directory storing the input and output files.

Returns

result_dict: dict

A dict that storing the result. For example: { “energy”: xxx, “force”: [xxx] }

abstract property forward_common_files

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

abstract property forward_files

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

abstract make_input_file(*output_dir*, *task_type*, *task_param*)

Prepare input files for a computational task For example, the VASP prepares INCAR. LAMMPS (including DeePMD, MEAM...) prepares in.lammps.

Parameters

output_dir

[str] The directory storing the input files.

task_type

[str] Can be - "relaxation:": structure relaxation - "static": static computation calculates the energy, force... of a strcture

task_parame: dict

The parameters of the task. For example the VASP interaction can be provided with { "ediff": 1e-6, "ediffg": 1e-5 }

abstract make_potential_files(*output_dir*)

Prepare potential files for a computational task. For example, the VASP prepares POTCAR. DeePMD prepares frozen model(s). IMPORTANT: Interaction should be stored in output_dir/inter.json

Parameters

output_dir

[str] The directory storing the potential files.

Outputs

——-

inter.json: output file

The task information is stored in *output_dir/inter.json*

dpgen.auto_test.VASP module

class dpgen.auto_test.VASP.**VASP**(*inter_parameter*, *path_to_poscar*)

Bases: *Task*

Methods

backward_files([property_type])	staticmethod(function) -> method
compute(output_dir)	Compute output of the task.
forward_common_files([property_type])	staticmethod(function) -> method
forward_files([property_type])	staticmethod(function) -> method
make_input_file(output_dir, task_type, ...)	Prepare input files for a computational task For example, the VASP prepares INCAR.
make_potential_files(output_dir)	Prepare potential files for a computational task.

backward_files(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

`compute`(*output_dir*)

Compute output of the task. IMPORTANT: The output configuration should be converted and stored in a CONTCAR file.

Parameters

output_dir

[str] The directory storing the input and output files.

Returns

result_dict: dict

A dict that storing the result. For example: { "energy": xxx, "force": [xxx] }

`forward_common_files`(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

`forward_files`(*property_type='relaxation'*)

staticmethod(function) -> method

Convert a function to be a static method.

A static method does not receive an implicit first argument. To declare a static method, use this idiom:

class C:

@staticmethod def f(arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class.

Static methods in Python are similar to those found in Java or C++. For a more advanced concept, see the classmethod builtin.

`make_input_file`(*output_dir*, *task_type*, *task_param*)

Prepare input files for a computational task For example, the VASP prepares INCAR. LAMMPS (including DeePMD, MEAM...) prepares in.lammps.

Parameters

output_dir

[str] The directory storing the input files.

task_type

[str] Can be - "relaxation:": structure relaxation - "static": static computation calculates the energy, force... of a strcture

task_parame: dict

The parameters of the task. For example the VASP interaction can be provided with { "ediff": 1e-6, "ediffg": 1e-5 }

make_potential_files(*output_dir*)

Prepare potential files for a computational task. For example, the VASP prepares POTCAR. DeePMD prepares frozen model(s). IMPORTANT: Interaction should be stored in output_dir/inter.json

Parameters

output_dir

[str] The directory storing the potential files.

Outputs

——-

inter.json: output file

The task information is stored in *output_dir/inter.json*

dpgen.auto_test.Vacancy module

class dpgen.auto_test.Vacancy.**Vacancy**(*parameter*, *inter_param=None*)

Bases: *Property*

Methods

compute(output_file, print_file, path_to_work)	Postprocess the finished tasks to compute the property.
make_confs(path_to_work, path_to_equi[, refine])	Make configurations needed to compute the property.
post_process(task_list)	post_process the KPOINTS file in elastic.
task_param()	Return the parameter of each computational task, for example, {'ediffg': 1e-4}
task_type()	Return the type of each computational task, for example, 'relaxation', 'static'....

make_confs(*path_to_work*, *path_to_equi*, *refine=False*)

Make configurations needed to compute the property. The tasks directory will be named as path_to_work/task.xxxxxx IMPORTANT: handel the case when the directory exists.

Parameters

path_to_work

[str] The path where the tasks for the property are located

path_to_equi

[str] -refine == False: The path to the directory that equilibrated the configuration. -refine == True: The path to the directory that has property confs.

refine: str

To refine existing property confs or generate property confs from a equilibrated conf

Returns

task_list: list of str

The list of task directories.

post_process(*task_list*)

post_process the KPOINTS file in elastic.

task_param()

Return the parameter of each computational task, for example, {'ediffg': 1e-4}

task_type()

Return the type of each computational task, for example, 'relaxation', 'static'....

dpgen.auto_test.calculator module

dpgen.auto_test.calculator.**make_calculator**(*inter_parameter*, *path_to_poscar*)
Make an instance of Task

dpgen.auto_test.common_equi module

dpgen.auto_test.common_equi.**make_equi**(*confs*, *inter_param*, *relax_param*)

dpgen.auto_test.common_equi.**post_equi**(*confs*, *inter_param*)

dpgen.auto_test.common_equi.**run_equi**(*confs*, *inter_param*, *mdata*)

dpgen.auto_test.common_prop module

dpgen.auto_test.common_prop.**make_property**(*confs*, *inter_param*, *property_list*)

dpgen.auto_test.common_prop.**make_property_instance**(*parameters*, *inter_param*)
Make an instance of Property

dpgen.auto_test.common_prop.**post_property**(*confs*, *inter_param*, *property_list*)

dpgen.auto_test.common_prop.**run_property**(*confs*, *inter_param*, *property_list*, *mdata*)

dpgen.auto_test.common_prop.**worker**(*work_path*, *all_task*, *forward_common_files*, *forward_files*, *backward_files*, *mdata*, *inter_type*)

dpgen.auto_test.gen_confs module

dpgen.auto_test.gen_confs.**gen_alloy**(*eles*, *key*)

dpgen.auto_test.gen_confs.**gen_ele_std**(*ele_name*, *ctype*)

dpgen.auto_test.gen_confs.**gen_element**(*ele_name*, *key*)

dpgen.auto_test.gen_confs.**gen_element_std**(*ele_name*)

dpgen.auto_test.gen_confs.**make_path_mp**(*ii*)

dpgen.auto_test.gen_confs.**test_fit**(*struct*, *data*)

dpgen.auto_test.mpdb module

dpgen.auto_test.mpdb.**check_apikey**()

dpgen.auto_test.mpdb.**get_structure**(*mp_id*)

dpgen.auto_test.refine module

dpgen.auto_test.refine.**make_refine**(*init_from_suffix*, *output_suffix*, *path_to_work*)

dpgen.auto_test.reproduce module

dpgen.auto_test.reproduce.**make_repro**(*inter_param*, *init_data_path*, *init_from_suffix*, *path_to_work*, *reprod_last_frame=True*)

dpgen.auto_test.reproduce.**post_repro**(*init_data_path*, *init_from_suffix*, *all_tasks*, *ptr_data*, *reprod_last_frame=True*)

dpgen.auto_test.run module

dpgen.auto_test.run.**gen_test**(*args*)

dpgen.auto_test.run.**run_task**(*step*, *param_file*, *machine_file=None*)

dpgen.collect package

Submodules

dpgen.collect.collect module

dpgen.collect.collect.**collect_data**(*target_folder*, *param_file*, *output*, *verbose=True*, *shuffle=True*, *merge=True*)

dpgen.collect.collect.**gen_collect**(*args*)

dpgen.data package

Subpackages

dpgen.data.tools package

Submodules

dpgen.data.tools.bcc module

dpgen.data.tools.bcc.**gen_box**()

dpgen.data.tools.bcc.**numb_atoms**()

dpgen.data.tools.bcc.**poscar_unit**(*latt*)

dpgen.data.tools.cessp2force_lin module

dpgen.data.tools.cessp2force_lin.**Parser**()

dpgen.data.tools.cessp2force_lin.**get_outcar_files**(*directory*, *recursive*)

dpgen.data.tools.cessp2force_lin.**process_outcar_file_v5_dev**(*outcars*, *data*, *numbers*, *types*, *max_types*, *elements=None*, *windex=None*, *fout='potfit.configs'*)

dpgen.data.tools.cessp2force_lin.**scan_outcar_file**(*file_handle*)

dpgen.data.tools.cessp2force_lin.**uniq**(*seq*)

dpgen.data.tools.create_random_disturb module

dpgen.data.tools.create_random_disturb.**RandomDisturbParser**()

dpgen.data.tools.create_random_disturb.**create_disturbs_abacus_dev**(*fin*, *nfile*, *dmax=1.0*, *etmax=0.1*, *ofmt='abacus'*, *dstyle='uniform'*, *write_d=False*, *diag=0*)

dpgen.data.tools.create_random_disturb.**create_disturbs_ase**(*fin*, *nfile*, *dmax=1.0*, *ofmt='lmp'*, *dstyle='uniform'*, *write_d=False*)

dpgen.data.tools.create_random_disturb.**create_disturbs_ase_dev**(*fin*, *nfile*, *dmax=1.0*, *etmax=0.1*, *ofmt='lmp'*, *dstyle='uniform'*, *write_d=False*, *diag=0*)

dpgen.data.tools.create_random_disturb.**create_disturbs_atomsk**(*fin*, *nfile*, *dmax=1.0*, *ofmt='lmp'*)

dpgen.data.tools.create_random_disturb.**create_random_alloys**(*fin*, *alloy_dist*, *ifmt='vasp'*, *ofmt='vasp'*)

In fact, atomsk also gives us the convinient tool to do this

dpgen.data.tools.create_random_disturb.**gen_random_disturb**(*dmax*, *a*, *b*, *dstyle='uniform'*)

dpgen.data.tools.create_random_disturb.**gen_random_emat**(*etmax*, *diag=0*)

dpgen.data.tools.create_random_disturb.**random_range**(*a*, *b*, *ndata=1*)

dpgen.data.tools.diamond module

dpgen.data.tools.diamond.**gen_box**()

dpgen.data.tools.diamond.**numb_atoms**()

dpgen.data.tools.diamond.**poscar_unit**(*latt*)

dpgen.data.tools.fcc module

dpgen.data.tools.fcc.**gen_box**()

dpgen.data.tools.fcc.**numb_atoms**()

dpgen.data.tools.fcc.**poscar_unit**(*latt*)

dpgen.data.tools.hcp module

dpgen.data.tools.hcp.**gen_box**()

dpgen.data.tools.hcp.**numb_atoms**()

dpgen.data.tools.hcp.**poscar_unit**(*latt*)

dpgen.data.tools.io_lammps module

ASE Atoms convert to LAMMPS configuration Some functions are adapted from ASE lammpsrun.py

dpgen.data.tools.io_lammps.**ase2lammpsdata**(*atoms*, *typeids=None*, *fout='out.lmp'*)

dpgen.data.tools.io_lammps.**car2dir**(*v*, *Ainv*)

Cartesian to direct coordinates

dpgen.data.tools.io_lammps.**convert_cell**(*ase_cell*)

Convert a parallel piped (forming right hand basis) to lower triangular matrix LAMMPS can accept. This function transposes cell matrix so the bases are column vectors

dpgen.data.tools.io_lammps.**convert_forces**(*forces0*, *cell0*, *cell_new*)

dpgen.data.tools.io_lammps.**convert_positions**(*pos0*, *cell0*, *cell_new*, *direct=False*)

dpgen.data.tools.io_lammps.**convert_stress**(*s6_0*, *cell0*, *cell_new*)

dpgen.data.tools.io_lammps.**dir2car**(*v*, *A*)

Direct to cartesian coordinates

dpgen.data.tools.io_lammps.**get_atoms_ntypes**(*atoms*)

dpgen.data.tools.io_lammps.**get_typeid**(*typeids*, *csymbol*)

dpgen.data.tools.io_lammps.**is_upper_triangular**(*mat*)

test if 3x3 matrix is upper triangular LAMMPS has a rule for cell matrix definition

dpgen.data.tools.io_lammps.**set_atoms_typeids**(*atoms*)

dpgen.data.tools.io_lammps.**set_atoms_typeids_with_atomic_numbers**(*atoms*)

dpgen.data.tools.io_lammps.**stress6_to_stress9**(*s6*)

dpgen.data.tools.io_lammps.**stress9_to_stress6**(*s9*)

dpgen.data.tools.ovito_file_convert module

dpgen.data.tools.poscar_copy module

dpgen.data.tools.sc module

dpgen.data.tools.sc.**gen_box**()

dpgen.data.tools.sc.**numb_atoms**()

dpgen.data.tools.sc.**poscar_unit**(*latt*)

Submodules

dpgen.data.arginfo module

dpgen.data.arginfo.**init_bulk_mdata_arginfo**() → Argument

Generate arginfo for dpgen init_bulk mdata.

Returns

Argument

arginfo

dpgen.data.arginfo.**init_reaction_jdata_arginfo**() → Argument

Generate arginfo for dpgen init_reaction jdata.

Returns

Argument

dpgen init_reaction jdata arginfo

dpgen.data.arginfo.**init_reaction_mdata_arginfo**() → Argument

Generate arginfo for dpgen init_reaction mdata.

Returns

Argument

arginfo

dpgen.data.arginfo.**init_surf_mdata_arginfo**() → Argument

Generate arginfo for dpgen init_surf mdata.

Returns

Argument

arginfo

dpgen.data.gen module

dpgen.data.gen.**class_cell_type**(*jdata*)

dpgen.data.gen.**coll_abacus_md**(*jdata*)

dpgen.data.gen.**coll_vasp_md**(*jdata*)

dpgen.data.gen.**create_path**(*path*, *back=False*)

dpgen.data.gen.**gen_init_bulk**(*args*)

dpgen.data.gen.**make_abacus_md**(*jdata*, *mdata*)

dpgen.data.gen.**make_abacus_relax**(*jdata*, *mdata*)

dpgen.data.gen.**make_combines**(*dim*, *natoms*)

dpgen.data.gen.**make_scale**(*jdata*)

dpgen.data.gen.**make_scale_ABACUS**(*jdata*)

dpgen.data.gen.**make_super_cell**(*jdata*)

dpgen.data.gen.**make_super_cell_ABACUS**(*jdata*, *stru_data*)

dpgen.data.gen.**make_super_cell_STRU**(*jdata*)

dpgen.data.gen.**make_super_cell_poscar**(*jdata*)

dpgen.data.gen.**make_unit_cell**(*jdata*)

dpgen.data.gen.**make_unit_cell_ABACUS**(*jdata*)

dpgen.data.gen.**make_vasp_md**(*jdata*, *mdata*)

dpgen.data.gen.**make_vasp_relax**(*jdata*, *mdata*)

dpgen.data.gen.**out_dir_name**(*jdata*)

dpgen.data.gen.**pert_scaled**(*jdata*)

dpgen.data.gen.**place_element**(*jdata*)

dpgen.data.gen.**place_element_ABACUS**(*jdata*, *supercell_stru*)

dpgen.data.gen.**poscar_ele**(*poscar_in*, *poscar_out*, *eles*, *natoms*)

dpgen.data.gen.**poscar_natoms**(*lines*)

dpgen.data.gen.**poscar_scale**(*poscar_in*, *poscar_out*, *scale*)

dpgen.data.gen.**poscar_scale_abacus**(*poscar_in*, *poscar_out*, *scale*, *jdata*)

dpgen.data.gen.**poscar_scale_cartesian**(*str_in*, *scale*)

dpgen.data.gen.**poscar_scale_direct**(*str_in*, *scale*)

dpgen.data.gen.**poscar_shuffle**(*poscar_in*, *poscar_out*)

dpgen.data.gen.**replace**(*file_name*, *pattern*, *subst*)

dpgen.data.gen.**run_abacus_md**(*jdata*, *mdata*)

dpgen.data.gen.**run_abacus_relax**(*jdata*, *mdata*)

dpgen.data.gen.**run_vasp_md**(*jdata*, *mdata*)

dpgen.data.gen.**run_vasp_relax**(*jdata*, *mdata*)

dpgen.data.gen.**shuffle_stru_data**(*supercell_stru*)

dpgen.data.gen.**stru_ele**(*supercell_stru*, *stru_out*, *eles*, *natoms*, *jdata*, *path_work*)

dpgen.data.reaction module

input: trajectory 00: ReaxFF MD (lammps) 01: build dataset (mddatasetbuilder) 02: fp (gaussian) 03: convert to deepmd data output: data

dpgen.data.reaction.**convert_data**(*jdata*)

dpgen.data.reaction.**gen_init_reaction**(*args*)

dpgen.data.reaction.**link_fp_input**()

dpgen.data.reaction.**link_reaxff**(*jdata*)

dpgen.data.reaction.**link_trj**(*jdata*)

link lammpstrj

dpgen.data.reaction.**make_lmp**(*jdata*)

dpgen.data.reaction.**run_build_dataset**(*jdata*, *mdata*, *log_file='build_log'*)

dpgen.data.reaction.**run_fp**(*jdata*, *mdata*, *log_file='output'*, *forward_common_files=[]*)

dpgen.data.reaction.**run_reaxff**(*jdata*, *mdata*, *log_file='reaxff_log'*)

dpgen.data.surf module

dpgen.data.surf.**class_cell_type**(*jdata*)

dpgen.data.surf.**create_path**(*path*)

dpgen.data.surf.**gen_init_surf**(*args*)

dpgen.data.surf.**make_combines**(*dim*, *natoms*)

dpgen.data.surf.**make_scale**(*jdata*)

dpgen.data.surf.**make_super_cell_pymatgen**(*jdata*)

dpgen.data.surf.**make_unit_cell**(*jdata*)

dpgen.data.surf.**make_vasp_relax**(*jdata*)

dpgen.data.surf.**out_dir_name**(*jdata*)

dpgen.data.surf.**pert_scaled**(*jdata*)

dpgen.data.surf.**place_element**(*jdata*)

dpgen.data.surf.**poscar_ele**(*poscar_in*, *poscar_out*, *eles*, *natoms*)

dpgen.data.surf.**poscar_elong**(*poscar_in*, *poscar_out*, *elong*, *shift_center=True*)

dpgen.data.surf.**poscar_natoms**(*poscar_in*)

dpgen.data.surf.**poscar_scale**(*poscar_in*, *poscar_out*, *scale*)

dpgen.data.surf.**poscar_scale_cartesian**(*str_in*, *scale*)

dpgen.data.surf.**poscar_scale_direct**(*str_in*, *scale*)

dpgen.data.surf.**poscar_shuffle**(*poscar_in*, *poscar_out*)

dpgen.data.surf.**replace**(*file_name*, *pattern*, *subst*)

dpgen.data.surf.**run_vasp_relax**(*jdata*, *mdata*)

dpgen.database package

Submodules

dpgen.database.entry module

class dpgen.database.entry.**Entry**(*composition*, *calculator*, *inputs*, *data*, *entry_id=None*, *attribute=None*, *tag=None*)

Bases: MSONable

An lightweight Entry object containing key computed data for storing purpose.

Attributes

number_element

Methods

as_dict()	A JSON serializable dict representation of an object.
from_dict(d)	**param d** Dict representation.
to_json()	Returns a json string representation of the MSONable object.
unsafe_hash()	Returns an hash of the current object.
validate_monty(v)	pydantic Validator for MSONable pattern

as_dict()

A JSON serializable dict representation of an object.

classmethod from_dict(*d*)

Parameters

d – Dict representation.

Returns

MSONable class.

property number_element

dpgen.database.run module

dpgen.database.run.**db_run**(*args*)

dpgen.database.run.**parsing_gaussian**(*path*, *output='dpgen_db.json'*)

dpgen.database.run.**parsing_pwscf**(*path*, *output='dpgen_db.json'*)

dpgen.database.run.**parsing_vasp**(*path*, *config_info_dict*, *skip_init*, *output='dpgen_db.json'*, *id_prefix=None*)

dpgen.database.vasp module

class dpgen.database.vasp.**DPPotcar**(*symbols=None*, *functional='PBE'*, *pp_file=None*, *pp_lists=None*)

Bases: MSONable

Methods

as_dict()	A JSON serializable dict representation of an object.
from_dict(d)	**param d** Dict representation.
to_json()	Returns a json string representation of the MSONable object.
unsafe_hash()	Returns an hash of the current object.
validate_monty(v)	pydantic Validator for MSONable pattern

from_file	
write_file	

as_dict()

A JSON serializable dict representation of an object.

classmethod from_dict(*d*)

Parameters

d – Dict representation.

Returns

MSONable class.

classmethod from_file(*filename*)

write_file(*filename*)

class dpgen.database.vasp.**VaspInput**(*incar*, *poscar*, *potcar*, *kpoints=None*, *optional_files=None*, ***kwargs*)

Bases: dict, MSONable

Class to contain a set of vasp input objects corresponding to a run.

Args:

incar: Incar object. kpoints: Kpoints object. poscar: Poscar object. potcar: Potcar object. optional_files: Other input files supplied as a dict of {

filename: object}. The object should follow standard pymatgen conventions in implementing a as_dict() and from_dict method.

Methods

as_dict()	A JSON serializable dict representation of an object.
clear()	
copy()	
from_dict(d)	**param d** Dict representation.
from_directory(input_dir[, optional_files])	Read in a set of VASP input from a directory.
fromkeys(iterable[, value])	Create a new dictionary with keys from iterable and values set to value.
get(key[, default])	Return the value for key if key is in the dictionary, else default.
items()	
keys()	
pop(k[,d])	If key is not found, d is returned if given, otherwise KeyError is raised
popitem()	2-tuple; but raise KeyError if D is empty.
setdefault(key[, default])	Insert key with a value of default if key is not in the dictionary.
to_json()	Returns a json string representation of the MSONable object.
unsafe_hash()	Returns an hash of the current object.
update([E,]**F)	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
validate_monty(v)	pydantic Validator for MSONable pattern
values()	
write_input([output_dir, ...])	Write VASP input to a directory.

as_dict()

A JSON serializable dict representation of an object.

classmethod from_dict(*d*)

Parameters
d – Dict representation.
Returns
MSONable class.

static from_directory(*input_dir*, *optional_files=None*)

Read in a set of VASP input from a directory. Note that only the standard INCAR, POSCAR, POTCAR and KPOINTS files are read unless optional_filenames is specified.

Args:

input_dir (str): Directory to read VASP input from. optional_files (dict): Optional files to read in as well as a

dict of {filename: Object type}. Object type must have a static method from_file.

write_input(*output_dir='.'*, *make_dir_if_not_present=True*)

Write VASP input to a directory.

Args:

output_dir (str): Directory to write to. Defaults to current directory (“.”).

make_dir_if_not_present (bool): Create the directory if not present. Defaults to True.

dpgen.dispatcher package

Submodules

dpgen.dispatcher.ALI module

dpgen.dispatcher.AWS module

class dpgen.dispatcher.AWS.**AWS**(*context*, *uuid_names=True*)

Bases: *Batch*

Attributes

job_id

Methods

AWS_check_status([job_id])	to aviod query jobStatus too often, set a time interval query_dict example: {job_id: JobStatus}
do_submit(job_dirs, cmd[, args, res, ...])	submit a single job, assuming that no job is running there.
sub_script(job_dirs, cmd, args, res, outlog, ...)	make submit script

check_finish_tag	
check_status	
default_resources	
map_aws_status_to_dpgen_status	
sub_script_cmd	
sub_script_head	
submit	

classmethod AWS_check_status(*job_id=''*)

to aviod query jobStatus too often, set a time interval query_dict example: {job_id: JobStatus}

{‘40fb24b2-d0ca-4443-8e3a-c0906ea03622’: <JobStatus.running: 3>,

‘41bda50c-0a23-4372-806c-87d16a680d85’: <JobStatus.waiting: 2>}

check_status()

default_resources(*res*)

do_submit(*job_dirs*, *cmd*, *args=None*, *res=None*, *outlog='log'*, *errlog='err'*)

submit a single job, assuming that no job is running there.

property job_id

static map_aws_status_to_dpgen_status(*aws_status*)

sub_script(*job_dirs*, *cmd*, *args*, *res*, *outlog*, *errlog*)

make submit script

job_dirs(list): directories of jobs. size: n_job cmd(list): commands to be executed. size: n_cmd args(list of list): args of commands. size of n_cmd x n_job
can be None
res(dict): resources available outlog(str): file name for output errlog(str): file name for error

dpgen.dispatcher.Batch module

class dpgen.dispatcher.Batch.**Batch**(*context*, *uuid_names=True*)

Bases: object

Methods

do_submit(job_dirs, cmd[, args, res, ...])	submit a single job, assuming that no job is running there.
sub_script(job_dirs, cmd[, args, res, ...])	make submit script

check_finish_tag	
check_status	
default_resources	
sub_script_cmd	
sub_script_head	
submit	

check_finish_tag()

check_status()

default_resources(*res*)

do_submit(*job_dirs*, *cmd*, *args=None*, *res=None*, *outlog='log'*, *errlog='err'*)

submit a single job, assuming that no job is running there.

sub_script(*job_dirs*, *cmd*, *args=None*, *res=None*, *outlog='log'*, *errlog='err'*)

make submit script

job_dirs(list): directories of jobs. size: n_job cmd(list): commands to be executed. size: n_cmd args(list of list): args of commands. size of n_cmd x n_job
can be None
res(dict): resources available outlog(str): file name for output errlog(str): file name for error

sub_script_cmd(*cmd*, *res*)

sub_script_head(*res*)

submit(*job_dirs*, *cmd*, *args=None*, *res=None*, *restart=False*, *outlog='log'*, *errlog='err'*)

dpgen.dispatcher.Dispatcher module

class dpgen.dispatcher.Dispatcher.**Dispatcher**(*remote_profile*, *context_type='local'*, *batch_type='slurm'*, *job_record='jr.json'*)

Bases: object

Methods

all_finished	
run_jobs	
submit_jobs	

all_finished(*job_handler*, *mark_failure*, *clean=True*)

run_jobs(*resources*, *command*, *work_path*, *tasks*, *group_size*, *forward_common_files*, *forward_task_files*, *backward_task_files*, *forward_task_deference=True*, *mark_failure=False*, *outlog='log'*, *errlog='err'*)

submit_jobs(*resources*, *command*, *work_path*, *tasks*, *group_size*, *forward_common_files*, *forward_task_files*, *backward_task_files*, *forward_task_deference=True*, *outlog='log'*, *errlog='err'*)

class dpgen.dispatcher.Dispatcher.**JobRecord**(*path*, *task_chunks*, *fname='job_record.json'*, *ip=None*)

Bases: object

Methods

check_all_finished	
check_finished	
check_nfail	
check_submitted	
dump	
get_uuid	
increase_nfail	
load	
record_finish	
record_remote_context	
valid_hash	

check_all_finished()

check_finished(*chunk_hash*)

check_nfail(*chunk_hash*)

check_submitted(*chunk_hash*)

dump()

get_uuid(*chunk_hash*)

increase_nfail(*chunk_hash*)

load()

record_finish(*chunk_hash*)

record_remote_context(*chunk_hash*, *local_root*, *remote_root*, *job_uuid*, *ip=None*, *instance_id=None*)

valid_hash(*chunk_hash*)

dpgen.dispatcher.Dispatcher.**make_dispatcher**(*mdata*, *mdata_resource=None*, *work_path=None*, *run_tasks=None*, *group_size=None*)

dpgen.dispatcher.Dispatcher.**make_submission**(*mdata_machine*, *mdata_resources*, *commands*, *work_path*, *run_tasks*, *group_size*, *forward_common_files*, *forward_files*, *backward_files*, *outlog*, *errlog*)

dpgen.dispatcher.Dispatcher.**make_submission_compat**(*machine: dict*, *resources: dict*, *commands: List[str]*, *work_path: str*, *run_tasks: List[str]*, *group_size: int*, *forward_common_files: List[str]*, *forward_files: List[str]*, *backward_files: List[str]*, *outlog: str = 'log'*, *errlog: str = 'err'*, *api_version: str = '0.9'*) → None

Make submission with compatibility of both dispatcher API v0 and v1.

If *api_version* is less than 1.0, use *make_dispatcher*. If *api_version* is large than 1.0, use *make_submission*.

Parameters

machine
: [dict] machine dict

resources
: [dict] resource dict

commands
: [list[str]] list of commands

work_path
: [str] working directory

run_tasks
: [list[str]] list of paths to running tasks

group_size
: [int] group size

forward_common_files
: [list[str]] forwarded common files shared for all tasks

forward_files
: [list[str]] forwarded files for each task

backward_files
: [list[str]] backwarded files for each task

outlog
: [str, default=log] path to log from stdout

errlog
: [str, default=err] path to log from stderr

api_version
: [str, default=0.9] API version. 1.0 is recommended

dpgen.dispatcher.Dispatcher.**mdata_arginfo**() → List[Argument]

This method generates arginfo for a single mdata.

A submission requires the following keys: command, machine, and resources.

Returns

list[Argument]
: arginfo

dpgen.dispatcher.DispatcherList module

class dpgen.dispatcher.DispatcherList.**DispatcherList**(*mdata_machine*, *mdata_resources*, *work_path*, *run_tasks*, *group_size*, *cloud_resources=None*)

Bases: object

Methods

catch_dispatcher_exception(ii)	everything is okay: return 0 ssh not active : return 1 machine callback : return 2
check_all_dispatchers_finished(ratio_failure)	
check_dispatcher_status(ii[, allow_failure])	catch running dispatcher exception if no exception occured, check finished
clean()	
create(ii)	case1: use existed machine(finished) to make_dispatcher case2: create one machine, then make_dispatcher, change status from unallocated to unsubmitted
delete(ii)	delete one machine if entity is none, means this machine is used by another dispatcher, shouldn't be deleted
exception_handling(ratio_failure)	
init()	
make_dispatcher(ii)	
run_jobs(resources, command, work_path, ...)	
update()	

catch_dispatcher_exception(*ii*)

everything is okay: return 0 ssh not active : return 1 machine callback : return 2

check_all_dispatchers_finished(*ratio_failure*)

check_dispatcher_status(*ii*, *allow_failure=False*)

catch running dispatcher exception if no exception occured, check finished

clean()

create(*ii*)

case1: use existed machine(finished) to make_dispatcher case2: create one machine, then make_dispatcher, change status from unallocated to unsubmitted

delete(*ii*)

delete one machine if entity is none, means this machine is used by another dispatcher, shouldn't be deleted

exception_handling(*ratio_failure*)

init()

make_dispatcher(*ii*)

run_jobs(*resources*, *command*, *work_path*, *tasks*, *group_size*, *forward_common_files*, *forward_task_files*, *backward_task_files*, *forward_task_deference=True*, *mark_failure=False*, *outlog='log'*, *errlog='err'*)

update()

class dpgen.dispatcher.DispatcherList.**Entity**(*ip*, *instance_id*, *job_record=None*, *job_handler=None*)

Bases: object

dpgen.dispatcher.JobStatus module

class dpgen.dispatcher.JobStatus.**JobStatus**(*value*)

Bases: Enum

An enumeration.

completing = 6

finished = 5

running = 3

terminated = 4

unknown = 100

unsubmitted = 1

waiting = 2

dpgen.dispatcher.LSF module

class dpgen.dispatcher.LSF.**LSF**(*context*, *uuid_names=True*)

Bases: *Batch*

Methods

default_resources(res_)	set default value if a key in **res_** is not fhound
do_submit(job_dirs, cmd[, args, res, ...])	submit a single job, assuming that no job is running there.
sub_script(job_dirs, cmd[, args, res, ...])	make submit script

check_finish_tag	
check_status	
sub_script_cmd	
sub_script_head	
submit	

check_status()

default_resources(*res_*)

set default value if a key in **res_** is not fhound

do_submit(*job_dirs*, *cmd*, *args=None*, *res=None*, *outlog='log'*, *errlog='err'*)

submit a single job, assuming that no job is running there.

sub_script_cmd(*cmd*, *arg*, *res*)

sub_script_head(*res*)

dpgen.dispatcher.LazyLocalContext module

class dpgen.dispatcher.LazyLocalContext.**LazyLocalContext**(*local_root*, *work_profile=None*, *job_uuid=None*)

Bases: object

Methods

block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
download	
get_job_root	
get_return	
kill	
read_file	
upload	
write_file	

block_call(*cmd*)

block_checkcall(*cmd*)

call(*cmd*)

check_file_exists(*fname*)

check_finish(*proc*)

clean()

download(*job_dirs*, *remote_down_files*, *check_exists=False*, *mark_failure=True*, *back_error=False*)

get_job_root()

get_return(*proc*)

kill(*proc*)

read_file(*fname*)

upload(*job_dirs*, *local_up_files*, *dereference=True*)

write_file(*fname*, *write_str*)

class dpgen.dispatcher.LazyLocalContext.**SPRetObj**(*ret*)

Bases: object

Methods

read	
readlines	

read()

readlines()

dpgen.dispatcher.LocalContext module

class dpgen.dispatcher.LocalContext.**LocalContext**(*local_root*, *work_profile*, *job_uuid=None*)

Bases: object

Methods

block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
download	
get_job_root	
get_return	
kill	
read_file	
upload	
write_file	

block_call(*cmd*)

block_checkcall(*cmd*)

call(*cmd*)

check_file_exists(*fname*)

check_finish(*proc*)

clean()

download(*job_dirs*, *remote_down_files*, *check_exists=False*, *mark_failure=True*, *back_error=False*)

get_job_root()

get_return(*proc*)

kill(*proc*)

read_file(*fname*)

upload(*job_dirs*, *local_up_files*, *dereference=True*)

write_file(*fname*, *write_str*)

class dpgen.dispatcher.LocalContext.**LocalSession**(*jdata*)

Bases: object

Methods

get_work_root	

get_work_root()

class dpgen.dispatcher.LocalContext.**SPRetObj**(*ret*)

Bases: object

Methods

read	
readlines	

read()

readlines()

dpgen.dispatcher.PBS module

class dpgen.dispatcher.PBS.**PBS**(*context*, *uuid_names=True*)

Bases: *Batch*

Methods

default_resources(res_)	set default value if a key in **res_** is not fhound
do_submit(job_dirs, cmd[, args, res, ...])	submit a single job, assuming that no job is running there.
sub_script(job_dirs, cmd[, args, res, ...])	make submit script

check_finish_tag	
check_status	
sub_script_cmd	
sub_script_head	
submit	

check_status()

default_resources(*res_*)

set default value if a key in **res_** is not fhound

do_submit(*job_dirs*, *cmd*, *args=None*, *res=None*, *outlog='log'*, *errlog='err'*)

submit a single job, assuming that no job is running there.

sub_script_cmd(*cmd*, *arg*, *res*)

sub_script_head(*res*)

dpgen.dispatcher.SSHContext module

class dpgen.dispatcher.SSHContext.**SSHContext**(*local_root*, *ssh_session*, *job_uuid=None*)

Bases: object

Attributes

sftp

ssh

Methods

block_call	
block_checkcall	
call	
check_file_exists	
check_finish	
clean	
close	
download	
get_job_root	
get_return	
kill	
read_file	
upload	
write_file	

block_call(*cmd*)

block_checkcall(*cmd*, *retry=0*)

call(*cmd*)

check_file_exists(*fname*)

check_finish(*cmd_pipes*)

clean()

close()

download(*job_dirs*, *remote_down_files*, *check_exists=False*, *mark_failure=True*, *back_error=False*)

get_job_root()

get_return(*cmd_pipes*)

kill(*cmd_pipes*)

read_file(*fname*)

property sftp

property ssh

upload(*job_dirs*, *local_up_files*, *dereference=True*)

write_file(*fname*, *write_str*)

class dpgen.dispatcher.SSHContext.**SSHSession**(*jdata*)

Bases: object

Attributes

sftp

Returns sftp.

Methods

exec_command(cmd[, retry])	Calling self.ssh.exec_command but has an exception check.

close	
ensure_alive	
get_session_root	
get_ssh_client	

close()

ensure_alive(*max_check=10*, *sleep_time=10*)

exec_command(*cmd*, *retry=0*)

Calling self.ssh.exec_command but has an exception check.

get_session_root()

get_ssh_client()

property sftp

Returns sftp. Open a new one if not existing.

dpgen.dispatcher.Shell module

class dpgen.dispatcher.Shell.**Shell**(*context*, *uuid_names=True*)

Bases: *Batch*

Methods

do_submit(job_dirs, cmd[, args, res, ...])	submit a single job, assuming that no job is running there.
sub_script(job_dirs, cmd[, args, res, ...])	make submit script

check_finish_tag
check_running
check_status
default_resources
sub_script_cmd
sub_script_head
submit

check_running()

check_status()

default_resources(*res_*)

do_submit(*job_dirs*, *cmd*, *args=None*, *res=None*, *outlog='log'*, *errlog='err'*)

submit a single job, assuming that no job is running there.

sub_script_cmd(*cmd*, *arg*, *res*)

sub_script_head(*resources*)

dpgen.dispatcher.Slurm module

class dpgen.dispatcher.Slurm.**Slurm**(*context*, *uuid_names=True*)

Bases: *Batch*

Methods

check_status()	check the status of a job
default_resources(res_)	set default value if a key in **res_** is not fhound
do_submit(job_dirs, cmd[, args, res, ...])	submit a single job, assuming that no job is running there.
sub_script(job_dirs, cmd[, args, res, ...])	make submit script

check_finish_tag
sub_script_cmd
sub_script_head
submit

check_status()
check the status of a job

default_resources(*res_*)
set default value if a key in **res_** is not fhound

do_submit(*job_dirs*, *cmd*, *args=None*, *res=None*, *outlog='log'*, *errlog='err'*)
submit a single job, assuming that no job is running there.

sub_script_cmd(*cmd*, *arg*, *res*)

sub_script_head(*res*)

dpgen.generator package

Subpackages

dpgen.generator.lib package

Submodules

dpgen.generator.lib.abacus_scf module

dpgen.generator.lib.abacus_scf.**get_abacus_STRU**(*STRU*, *INPUT=None*, *n_ele=None*)

dpgen.generator.lib.abacus_scf.**get_abacus_input_parameters**(*INPUT*)

dpgen.generator.lib.abacus_scf.**get_additional_from_STRU**(*geometry_inlines*, *nele*)

dpgen.generator.lib.abacus_scf.**get_mass_from_STRU**(*geometry_inlines*, *inlines*, *atom_names*)

dpgen.generator.lib.abacus_scf.**get_natoms_from_stru**(*geometry_inlines*)

dpgen.generator.lib.abacus_scf.**make_abacus_scf_input**(*fp_params*)

dpgen.generator.lib.abacus_scf.**make_abacus_scf_kpt**(*fp_params*)

dpgen.generator.lib.abacus_scf.**make_abacus_scf_stru**(*sys_data*, *fp_pp_files*, *fp_orb_files=None*, *fp_dpks_descriptor=None*, *fp_params=None*)

dpgen.generator.lib.abacus_scf.**make_kspacing_kpoints_stru**(*stru*, *kspacing*)

dpgen.generator.lib.abacus_scf.**make_supercell_abacus**(*from_struct*, *super_cell*)

dpgen.generator.lib.calypso_check_outcar module

dpgen.generator.lib.calypso_run_model_devi module

dpgen.generator.lib.calypso_run_opt module

dpgen.generator.lib.cp2k module

dpgen.generator.lib.cp2k.**iterdict**(*d*, *out_list*, *flag=None*)

Doc
a recursive expansion of dictionary into cp2k input
K
current key
V
current value
D
current dictionary under expansion
Flag
used to record dictionary state. if flag is None,
it means we are in top level dict. flag is a string.

dpgen.generator.lib.cp2k.**make_cp2k_input**(*sys_data*, *fp_params*)

dpgen.generator.lib.cp2k.**make_cp2k_input_from_external**(*sys_data*, *exinput_path*)

dpgen.generator.lib.cp2k.**make_cp2k_xyz**(*sys_data*)

dpgen.generator.lib.cp2k.**update_dict**(*old_d*, *update_d*)
a method to recursive update dict :old_d: old dictionary :update_d: some update value written in dictionary form

dpgen.generator.lib.cvasp module

dpgen.generator.lib.cvasp.**runvasp**(*cmd*, *opt=False*, *max_errors=3*, *backup=False*, *auto_gamma=False*, *auto_npar=False*, *ediffg=-0.05*)
cmd example: cmd=[‘mpirun’, ‘-np’, ‘32’ , ‘-machinefile’, ‘hosts’,’vasp_std’]

dpgen.generator.lib.ele_temp module

class dpgen.generator.lib.ele_temp.**NBandsEsti**(*test_list*)
Bases: `object`

Methods

predict	
save	

predict(*target_dir*, *tolerance=0.5*)

save(*fname*)

dpgen.generator.lib.gaussian module

dpgen.generator.lib.gaussian.**detect_multiplicity**(*symbols*)

dpgen.generator.lib.gaussian.**make_gaussian_input**(*sys_data*, *fp_params*)

dpgen.generator.lib.gaussian.**take_cluster**(*old_conf_name*, *type_map*, *idx*, *jdata*)

dpgen.generator.lib.lammps module

dpgen.generator.lib.lammps.**get_all_dumped_forces**(*file_name*)

dpgen.generator.lib.lammps.**get_dumped_forces**(*file_name*)

dpgen.generator.lib.lammps.**make_lammps_input**(*ensemble*, *conf_file*, *graphs*, *nsteps*, *dt*, *neidelay*, *trj_freq*, *mass_map*, *temp*, *jdata*, *tau_t=0.1*, *pres=None*, *tau_p=0.5*, *pka_e=None*, *ele_temp_f=None*, *ele_temp_a=None*, *max_seed=1000000*, *nopbc=False*, *deepmd_version='0.1'*)

dpgen.generator.lib.make_calypso module

dpgen.generator.lib.make_calypso.**make_calypso_input**(*nameofatoms*, *numberofatoms*, *numberofformula*, *volume*, *distanceofion*, *psoratio*, *popsize*, *maxstep*, *icode*, *split*, *vsc*, *maxnumatom*, *ctrlrange*, *pstress*, *fmax*)

dpgen.generator.lib.make_calypso.**write_model_devi_out**(*devi*, *fname*)

dpgen.generator.lib.parse_calypso module

dpgen.generator.lib.pwmat module

dpgen.generator.lib.pwmat.**input_upper**(*dinput*)

dpgen.generator.lib.pwmat.**make_pwmat_input_dict**(*node1*, *node2*, *atom_config*, *ecut*, *e_error*, *rho_error*, *icmix=None*, *smearing=None*, *sigma=None*, *kspacing=0.5*, *flag_symm=None*)

dpgen.generator.lib.pwmat.**make_pwmat_input_user_dict**(*fp_params*)

dpgen.generator.lib.pwmat.**write_input_dict**(*input_dict*)

dpgen.generator.lib.pwscf module

dpgen.generator.lib.pwscf.**cvt_1frame**(*fin*, *fout*)

dpgen.generator.lib.pwscf.**get_atom_types**(*lines*)

dpgen.generator.lib.pwscf.**get_block**(*lines*, *keyword*, *skip=0*)

dpgen.generator.lib.pwscf.**get_cell**(*lines*)

dpgen.generator.lib.pwscf.**get_coords**(*lines*)

dpgen.generator.lib.pwscf.**get_energy**(*lines*)

dpgen.generator.lib.pwscf.**get_force**(*lines*)

dpgen.generator.lib.pwscf.**get_natoms**(*lines*)

dpgen.generator.lib.pwscf.**get_stress**(*lines*, *cells*)

dpgen.generator.lib.pwscf.**get_types**(*lines*)

dpgen.generator.lib.pwscf.**make_pwscf_01_runctrl_dict**(*sys_data*, *idict*)

dpgen.generator.lib.pwscf.**make_pwscf_input**(*sys_data*, *fp_pp_files*, *fp_params*, *user_input=True*)

dpgen.generator.lib.run_calypso module

calypso as model devi engine:

1. gen_structures
2. analysis
3. model devi

dpgen.generator.lib.run_calypso.**analysis**(*iter_index*, *jdata*, *calypso_model_devi_path*)

dpgen.generator.lib.run_calypso.**gen_main**(*iter_index*, *jdata*, *mdata*, *caly_run_opt_list*, *gen_idx*)

dpgen.generator.lib.run_calypso.**gen_structures**(*iter_index*, *jdata*, *mdata*, *caly_run_path*, *current_idx*, *length_of_caly_runopt_list*)

dpgen.generator.lib.run_calypso.**run_calypso_model_devi**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.lib.siesta module

dpgen.generator.lib.siesta.**make_siesta_input**(*sys_data*, *fp_pp_files*, *fp_params*)

dpgen.generator.lib.utils module

dpgen.generator.lib.utils.**cmd_append_log**(*cmd*, *log_file*)

dpgen.generator.lib.utils.**copy_file_list**(*file_list*, *from_path*, *to_path*)

dpgen.generator.lib.utils.**create_path**(*path*)

dpgen.generator.lib.utils.**log_iter**(*task*, *ii*, *jj*)

dpgen.generator.lib.utils.**log_task**(*message*)

dpgen.generator.lib.utils.**make_iter_name**(*iter_index*)

dpgen.generator.lib.utils.**record_iter**(*record*, *ii*, *jj*)

dpgen.generator.lib.utils.**repeat_to_length**(*string_to_expand*, *length*)

dpgen.generator.lib.utils.**replace**(*file_name*, *pattern*, *subst*)

dpgen.generator.lib.utils.**symlink_user_forward_files**(*mdata*, *task_type*, *work_path*, *task_format=None*)

Symlink user-defined forward_common_files Current path should be work_path, such as 00.train

mdata
: [dict] machine parameters

task_type: str
: task_type, such as "train"

work_path
: [str] work_path, such as "iter.000001/00.train"

None

dpgen.generator.lib.vasp module

dpgen.generator.lib.vasp.**incar_upper**(*dincar*)

dpgen.generator.lib.vasp.**make_vasp_incar_user_dict**(*fp_params*)

dpgen.generator.lib.vasp.**write_incar_dict**(*incar_dict*)

Submodules

dpgen.generator.arginfo module

dpgen.generator.arginfo.**basic_args**() → List[Argument]

dpgen.generator.arginfo.**data_args**() → List[Argument]

dpgen.generator.arginfo.**fp_args**() → List[Argument]

dpgen.generator.arginfo.**fp_style_abacus_args**() → List[Argument]

dpgen.generator.arginfo.**fp_style_amber_diff_args**() → List[Argument]

Arguments for FP style amber/diff.

Returns

list[dargs.Argument]

list of Gaussian fp style arguments

dpgen.generator.arginfo.**fp_style_cp2k_args**() → List[Argument]

dpgen.generator.arginfo.**fp_style_gaussian_args**() → List[Argument]

Gaussian fp style arguments.

Returns

list[dargs.Argument]

list of Gaussian fp style arguments

dpgen.generator.arginfo.**fp_style_siesta_args**() → List[Argument]

dpgen.generator.arginfo.**fp_style_variant_type_args**() → Variant

dpgen.generator.arginfo.**fp_style_vasp_args**() → List[Argument]

dpgen.generator.arginfo.**model_devi_amber_args**() → List[Argument]

Amber engine arguments.

dpgen.generator.arginfo.**model_devi_args**() → List[Variant]

dpgen.generator.arginfo.**model_devi_jobs_args**() → List[Argument]

dpgen.generator.arginfo.**model_devi_lmp_args**() → List[Argument]

dpgen.generator.arginfo.**run_jdata_arginfo**() → Argument

Argument information for dpgen run mdata.

Returns

Argument

argument information

dpgen.generator.arginfo.**run_mdata_arginfo**() → Argument

Generate arginfo for dpgen run mdata.

Returns

Argument

arginfo

dpgen.generator.arginfo.**training_args**() → List[Argument]

Traning arguments.

Returns

list[dargs.Argument]

List of training arguments.

dpgen.generator.run module

init: data iter:

00.train 01.model_devi 02.vasp 03.data

dpgen.generator.run.**check_bad_box**(*conf_name*, *criteria*, *fmt='lammps/dump'*)

dpgen.generator.run.**check_cluster**(*conf_name*, *fp_cluster_vacuum*, *fmt='lammps/dump'*)

dpgen.generator.run.**copy_model**(*numb_model*, *prv_iter_index*, *cur_iter_index*)

dpgen.generator.run.**detect_batch_size**(*batch_size*, *system=None*)

dpgen.generator.run.**dump_to_deepmd_raw**(*dump*, *deepmd_raw*, *type_map*, *fmt='gromacs/gro'*, *charge=None*)

dpgen.generator.run.**expand_idx**(*in_list*)

dpgen.generator.run.**expand_matrix_values**(*target_list*, *cur_idx=0*)

dpgen.generator.run.**find_only_one_key**(*lmp_lines*, *key*)

dpgen.generator.run.**gen_run**(*args*)

dpgen.generator.run.**get_atomic_masses**(*atom*)

dpgen.generator.run.**get_job_names**(*jdata*)

dpgen.generator.run.**get_sys_index**(*task*)

dpgen.generator.run.**make_fp**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**make_fp_abacus_scf**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_amber_diff**(*iter_index: int*, *jdata: dict*)

Run amber twice to calculate high-level and low-level potential, and then generate difference between them.

Besides AMBER, one needs to install *dpamber* package, which is avaiable at https://github.com/njzjz/dpamber

Currently, it should be used with the AMBER model_devi driver.

Parameters

iter_index

[int] iter index

jdata

[dict]

Run parameters. The following parameters are used in this method:

mdin_prefix

[str] The path prefix to AMBER mdin files

qm_region

[list[str]] AMBER mask of the QM region. Each mask maps to a system.

qm_charge

[list[int]] Charge of the QM region. Each charge maps to a system.

high_level

[str] high level method

low_level

[str] low level method

fp_params

[dict]

This parameters includes:

high_level_mdin

[str] High-level AMBER mdin file. %qm_theory%, %qm_region%, and %qm_charge% will be replace.

low_level_mdin
[str] Low-level AMBER mdin file. %qm_theory%, %qm_region%, and %qm_charge% will be replace.

parm7_prefix
[str] The path prefix to AMBER PARM7 files

parm7
[list[str]] List of paths to AMBER PARM7 files. Each file maps to a system.

References

[1]

dpgen.generator.run.**make_fp_cp2k**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_gaussian**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_pwmat**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_pwscf**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_siesta**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_task_name**(*sys_idx*, *counter*)

dpgen.generator.run.**make_fp_vasp**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_vasp_cp_cvasp**(*iter_index*, *jdata*)

dpgen.generator.run.**make_fp_vasp_incar**(*iter_index*, *jdata*, *nbands_esti=None*)

dpgen.generator.run.**make_fp_vasp_kp**(*iter_index*, *jdata*)

dpgen.generator.run.**make_model_devi**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**make_model_devi_conf_name**(*sys_idx*, *conf_idx*)

dpgen.generator.run.**make_model_devi_task_name**(*sys_idx*, *task_idx*)

dpgen.generator.run.**make_pwmat_input**(*jdata*, *filename*)

dpgen.generator.run.**make_train**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**make_vasp_incar**(*jdata*, *filename*)

dpgen.generator.run.**make_vasp_incar_ele_temp**(*jdata*, *filename*, *ele_temp*, *nbands_esti=None*)

dpgen.generator.run.**parse_cur_job**(*cur_job*)

dpgen.generator.run.**parse_cur_job_revmat**(*cur_job*, *use_plm=False*)

dpgen.generator.run.**parse_cur_job_sys_revmat**(*cur_job*, *sys_idx*, *use_plm=False*)

dpgen.generator.run.**poscar_natoms**(*lines*)

dpgen.generator.run.**poscar_shuffle**(*poscar_in*, *poscar_out*)

dpgen.generator.run.**poscar_to_conf**(*poscar*, *conf*)

dpgen.generator.run.**post_fp**(*iter_index*, *jdata*)

dpgen.generator.run.**post_fp_abacus_scf**(*iter_index*, *jdata*)

dpgen.generator.run.**post_fp_amber_diff**(*iter_index*, *jdata*)

dpgen.generator.run.**post_fp_check_fail**(*iter_index*, *jdata*, *rfailed=None*)

dpgen.generator.run.**post_fp_cp2k**(*iter_index*, *jdata*, *rfailed=None*)

dpgen.generator.run.**post_fp_gaussian**(*iter_index*, *jdata*)

dpgen.generator.run.**post_fp_pwmat**(*iter_index*, *jdata*, *rfailed=None*)

dpgen.generator.run.**post_fp_pwscf**(*iter_index*, *jdata*)

dpgen.generator.run.**post_fp_siesta**(*iter_index*, *jdata*)

dpgen.generator.run.**post_fp_vasp**(*iter_index*, *jdata*, *rfailed=None*)

dpgen.generator.run.**post_model_devi**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**post_train**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**revise_by_keys**(*lmp_lines*, *keys*, *values*)

dpgen.generator.run.**revise_lmp_input_dump**(*lmp_lines*, *trj_freq*)

dpgen.generator.run.**revise_lmp_input_model**(*lmp_lines*, *task_model_list*, *trj_freq*, *deepmd_version='1'*)

dpgen.generator.run.**revise_lmp_input_plm**(*lmp_lines*, *in_plm*, *out_plm='output.plumed'*)

dpgen.generator.run.**run_fp**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**run_fp_inner**(*iter_index*, *jdata*, *mdata*, *forward_files*, *backward_files*, *check_fin*, *log_file='fp.log'*, *forward_common_files=[]*)

dpgen.generator.run.**run_iter**(*param_file*, *machine_file*)

dpgen.generator.run.**run_md_model_devi**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**run_model_devi**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**run_train**(*iter_index*, *jdata*, *mdata*)

dpgen.generator.run.**set_version**(*mdata*)

dpgen.generator.run.**sys_link_fp_vasp_pp**(*iter_index*, *jdata*)

dpgen.generator.run.**update_mass_map**(*jdata*)

dpgen.remote package

Submodules

dpgen.remote.RemoteJob module

class dpgen.remote.RemoteJob.**CloudMachineJob**(*ssh_session*, *local_root*, *job_uuid=None*)

Bases: *RemoteJob*

Methods

block_call	
block_checkcall	
check_status	
clean	
download	
get_job_root	
submit	
upload	

check_status()

submit(*job_dirs*, *cmd*, *args=None*, *resources=None*)

class dpgen.remote.RemoteJob.**JobStatus**(*value*)

Bases: Enum

An enumeration.

finished = 5

running = 3

terminated = 4

unknown = 100

unsubmitted = 1

waiting = 2

class dpgen.remote.RemoteJob.**LSFJob**(*ssh_session*, *local_root*, *job_uuid=None*)

Bases: *RemoteJob*

Methods

block_call	
block_checkcall	
check_limit	
check_status	
clean	
download	
get_job_root	
submit	
upload	

check_limit(*task_max*)

check_status()

submit(*job_dirs*, *cmd*, *args=None*, *resources=None*, *restart=False*)

class dpgen.remote.RemoteJob.**PBSJob**(*ssh_session*, *local_root*, *job_uuid=None*)

Bases: *RemoteJob*

Methods

block_call	
block_checkcall	
check_status	
clean	
download	
get_job_root	
submit	
upload	

check_status()

submit(*job_dirs*, *cmd*, *args=None*, *resources=None*)

class dpgen.remote.RemoteJob.**RemoteJob**(*ssh_session*, *local_root*, *job_uuid=None*)

Bases: object

Methods

block_call	
block_checkcall	
clean	
download	
get_job_root	
upload	

block_call(*cmd*)

block_checkcall(*cmd*)

clean()

download(*job_dirs*, *remote_down_files*, *back_error=False*)

get_job_root()

upload(*job_dirs*, *local_up_files*, *dereference=True*)

class dpgen.remote.RemoteJob.**SSHSession**(*jdata*)

Bases: object

Methods

close	
get_session_root	
get_ssh_client	

close()

get_session_root()

get_ssh_client()

class dpgen.remote.RemoteJob.**SlurmJob**(*ssh_session*, *local_root*, *job_uuid=None*)

Bases: *RemoteJob*

Methods

block_call	
block_checkcall	
check_status	
clean	
download	
get_job_root	
submit	
upload	

check_status()

submit(*job_dirs*, *cmd*, *args=None*, *resources=None*, *restart=False*)

class dpgen.remote.RemoteJob.**awsMachineJob**(*remote_root*, *work_path*, *job_uuid=None*)

Bases: object

Methods

download	
upload	

download(*job_dir*, *remote_down_files*, *dereference=True*)

upload(*job_dir*, *local_up_files*, *dereference=True*)

dpgen.remote.decide_machine module

dpgen.remote.decide_machine.**convert_mdata**(*mdata*, *task_types=['train', 'model_devi', 'fp']*)

Convert mdata for DP-GEN main process. New convension is like mdata["fp"]["machine"], DP-GEN needs mdata["fp_machine"]

Notice that we deprecate the function which can automatically select one most avalaible machine, since this function was only used by Angus, and only supports for Slurm. In the future this can be implemented.

Parameters

mdata

[dict] Machine parameters to be converted.

task_types

[list of string] Type of tasks, default is ["train", "model_devi", "fp"]

Returns

dict

mdata converted

dpgen.remote.group_jobs module

class dpgen.remote.group_jobs.**PMap**(*path*, *fname='pmap.json'*)

Bases: object

Path map class to operate {read,write,delte} the pmap.json file

Methods

delete	
dump	
load	

delete()

dump(*pmap*, *indent=4*)

load()

dpgen.remote.group_jobs.**aws_submit_jobs**(*machine*, *resources*, *command*, *work_path*, *tasks*, *group_size*, *forward_common_files*, *forward_task_files*, *backward_task_files*, *forward_task_deference=True*)

dpgen.remote.group_jobs.**group_local_jobs**(*ssh_sess*, *resources*, *command*, *work_path*, *tasks*, *group_size*, *forward_common_files*, *forward_task_files*, *backward_task_files*, *forward_task_deference=True*)

dpgen.remote.group_jobs.**group_slurm_jobs**(*ssh_sess*, *resources*, *command*, *work_path*, *tasks*, *group_size*, *forward_common_files*, *forward_task_files*, *backward_task_files*, *remote_job=<class 'dpgen.remote.RemoteJob.SlurmJob'>*, *forward_task_deference=True*)

dpgen.remote.group_jobs.**ucloud_submit_jobs**(*machine*, *resources*, *command*, *work_path*, *tasks*, *group_size*, *forward_common_files*, *forward_task_files*, *backward_task_files*, *forward_task_deference=True*)

dpgen.simplify package

Submodules

dpgen.simplify.arginfo module

dpgen.simplify.arginfo.**fp_args**() → List[Argument]

Generate arginfo for fp.

Returns

List[Argument]

arginfo

dpgen.simplify.arginfo.**fp_style_variant_type_args**() → Variant

Generate variant for fp style variant type.

Returns

Variant

variant for fp style

dpgen.simplify.arginfo.**general_simplify_arginfo**() → Argument

General simplify arginfo.

Returns

Argument

arginfo

dpgen.simplify.arginfo.**simplify_jdata_arginfo**() → Argument

Generate arginfo for dpgen simplify jdata.

Returns

Argument

arginfo

dpgen.simplify.arginfo.**simplify_mdata_arginfo**() → Argument

Generate arginfo for dpgen simplify mdata.

Returns

Argument

arginfo

dpgen.simplify.simplify module

Simplify dataset (minimize the dataset size).

Init: pick up init data from dataset randomly

Iter: 00: train models (same as generator) 01: calculate model deviations of the rest dataset, pick up data with proper model deviaiton 02: fp (optional, if the original dataset do not have fp data, same as generator)

dpgen.simplify.simplify.**gen_simplify**(*args*)

dpgen.simplify.simplify.**get_multi_system**(*path*, *jdata*)

dpgen.simplify.simplify.**get_system_cls**(*jdata*)

dpgen.simplify.simplify.**init_model**(*iter_index*, *jdata*, *mdata*)

dpgen.simplify.simplify.**init_pick**(*iter_index*, *jdata*, *mdata*)
pick up init data from dataset randomly

dpgen.simplify.simplify.**make_fp**(*iter_index*, *jdata*, *mdata*)

dpgen.simplify.simplify.**make_fp_calculation**(*iter_index*, *jdata*)

dpgen.simplify.simplify.**make_fp_configs**(*iter_index*, *jdata*)

dpgen.simplify.simplify.**make_fp_gaussian**(*iter_index*, *jdata*)

dpgen.simplify.simplify.**make_fp_labeled**(*iter_index*, *jdata*)

dpgen.simplify.simplify.**make_fp_vasp**(*iter_index*, *jdata*)

dpgen.simplify.simplify.**make_model_devi**(*iter_index*, *jdata*, *mdata*)
calculate the model deviation of the rest idx

dpgen.simplify.simplify.**post_model_devi**(*iter_index*, *jdata*, *mdata*)
calculate the model deviation

dpgen.simplify.simplify.**run_iter**(*param_file*, *machine_file*)
init (iter 0): init_pick

tasks (iter > 0): 00 make_train (same as generator) 01 run_train (same as generator) 02 post_train (same as generator) 03 make_model_devi 04 run_model_devi 05 post_model_devi 06 make_fp 07 run_fp (same as generator) 08 post_fp (same as generator)

dpgen.simplify.simplify.**run_model_devi**(*iter_index*, *jdata*, *mdata*)
submit dp test tasks

dpgen.tools package

Submodules

dpgen.tools.auto_gen_param module

class dpgen.tools.auto_gen_param.**Iteration**(*temps*, *nsteps_list=[500, 500, 1000, 1000, 3000, 3000, 6000, 6000]*, *sub_iteration_num=8*, *ensemble='npt'*, *press=[1.0, 10.0, 100.0, 1000.0, 5000.0, 10000.0, 20000.0, 50000.0]*, *trj_freq=10*)

Bases: object

Attributes

index_iteration

Methods

gen_sub_iter	
register_iteration	
register_sub_iteartion	

current_num_of_itearation = 0

current_num_of_sub_itearation = 0

gen_sub_iter(*system_list*)

property index_iteration

classmethod register_iteration()

classmethod register_sub_iteartion()

class dpgen.tools.auto_gen_param.**System**(*system_prefix=''*)

Bases: object

Attributes

index_system

Methods

add_sub_system	
get_sub_system	
register_sub_system	
register_system	

add_sub_system(*idx2*, *files_list*)

current_num_of_sub_systems = 0

current_num_of_system = 0

get_sub_system()

property index_system

classmethod register_sub_system()

classmethod register_system()

dpgen.tools.auto_gen_param.**auto_gen_param**(*args*)

dpgen.tools.auto_gen_param.**default_map_generator**(*map_list=[1, 1, 2, 2, 2, 4, 4, 4]*, *data_list=None*)

dpgen.tools.auto_gen_param.**default_temps_generator**(*melt_point*, *temps_intervel=0.1*, *num_temps=5*)

dpgen.tools.auto_gen_param.**get_basic_param_json**(*melt_point*, *out_param_filename='param_basic.json'*, *scan_dir='./'*, *file_name='POSCAR'*, *init_file_name='type.raw'*, *min_allow_files_num=16*, *map_list=[1, 1, 2, 2, 2, 4, 4, 4]*, *meta_iter_num=4*, *sub_iteration_num=8*, *map_iterator=None*, *nsteps_list=[500, 500, 1000, 1000, 3000, 3000, 6000, 6000]*, *press=[1.0, 10.0, 100.0, 1000.0, 5000.0, 10000.0, 20000.0, 50000.0]*, *temps_iterator=None*, *ensemble='npt'*, *trj_freq=10*, *temps_intervel=0.1*, *num_temps=5*)

dpgen.tools.auto_gen_param.**get_init_data_sys**(*scan_dir='./'*, *init_file_name='type.raw'*)

dpgen.tools.auto_gen_param.**get_model_devi_jobs**(*melt_point*, *system_list*, *nsteps_list=[500, 500, 1000, 1000, 3000, 3000, 6000, 6000]*, *press=[1.0, 10.0, 100.0, 1000.0, 5000.0, 10000.0, 20000.0, 50000.0]*, *meta_iter_num=4*, *sub_iteration_num=8*, *temps_iterator=None*, *ensemble='npt'*, *trj_freq=10*, *temps_intervel=0.1*, *num_temps=5*)

dpgen.tools.auto_gen_param.**get_sys_configs**(*system_list*)

dpgen.tools.auto_gen_param.**get_system_list**(*system_dict*, *map_list=[1, 1, 2, 2, 2, 4, 4, 4]*, *meta_iter_num=4*, *sub_iteration_num=8*, *map_iterator=None*, *file_name='POSCAR'*)

:Exmaple [[‘000000’, ‘000001’,], [‘00000[2-9]’,], [‘00001?’, ‘000020’,],]

dpgen.tools.auto_gen_param.**scan_files**(*scan_dir='./'*, *file_name='POSCAR'*, *min_allow_files_num=20*)

dpgen.tools.collect_data module

dpgen.tools.collect_data.**collect_data**(*target_folder*, *param_file*, *output*, *verbose=True*)

dpgen.tools.collect_data.**file_len**(*fname*)

dpgen.tools.relabel module

dpgen.tools.relabel.**copy_pp_files**(*tdir*, *fp_pp_path*, *fp_pp_files*)

dpgen.tools.relabel.**create_init_tasks**(*target_folder*, *param_file*, *output*, *fp_json*, *verbose=True*)

dpgen.tools.relabel.**create_tasks**(*target_folder*, *param_file*, *output*, *fp_json*, *verbose=True*, *numb_iter=-1*)

dpgen.tools.relabel.**get_lmp_info**(*input_file*)

dpgen.tools.relabel.**link_pp_files**(*tdir*, *fp_pp_path*, *fp_pp_files*)

dpgen.tools.relabel.**make_pwscf**(*tdir*, *fp_params*, *mass_map*, *fp_pp_path*, *fp_pp_files*, *user_input*)

dpgen.tools.relabel.**make_siesta**(*tdir*, *fp_params*, *fp_pp_path*, *fp_pp_files*)

dpgen.tools.relabel.**make_vasp**(*tdir*, *fp_params*)

dpgen.tools.relabel.**make_vasp_incar**(*tdir*, *fp_incar*)

dpgen.tools.run_report module

dpgen.tools.run_report.**run_report**(*args*)

dpgen.tools.stat_iter module

dpgen.tools.stat_iter.**stat_iter**(*target_folder*, *param_file='param.json'*, *verbose=True*, *mute=False*)

dpgen.tools.stat_sys module

dpgen.tools.stat_sys.**ascii_hist**(*count*)

dpgen.tools.stat_sys.**run_report**(*args*)

dpgen.tools.stat_sys.**stat_sys**(*target_folder*, *param_file='param.json'*, *verbose=True*, *mute=False*)

dpgen.tools.stat_time module

dpgen.tools.stat_time.**stat_time**(*target_folder*, *param_file='param.json'*, *verbose=True*, *mute=False*)

10.1.2 Submodules

10.1.3 dpgen.arginfo module

dpgen.arginfo.**general_mdata_arginfo**(*name: str*, *tasks: Tuple[str]*) → Argument

Generate arginfo for general mdata.

Parameters

name

[str] mdata name

tasks

[tuple[str]] tuple of task keys, e.g. (“train”, “model_devi”, “fp”)

Returns

Argument

arginfo

10.1.4 dpgen.main module

dpgen.main.**main**()

dpgen.main.**main_parser**() → ArgumentParser

Returns parser for *dpgen* command.

Returns

argparse.ArgumentParser

parser for *dpgen* command

10.1.5 dpgen.util module

dpgen.util.**box_center**(*ch=''*, *fill=' '*, *sp='|'*)

put the string at the center of | |

dpgen.util.**expand_sys_str**(*root_dir: Union[str, Path]*) → List[str]

Recursively iterate over directories taking those that contain *type.raw* file.

Parameters

root_dir

[Union[str, Path]] starting directory

Returns

List[str]

list of string pointing to system directories

dpgen.util.**normalize**(*arginfo: Argument*, *data: dict*, *strict_check: bool = True*) → dict

Normalize and check input data.

Parameters

arginfo

[dargs.Argument] argument information

data

[dict] input data

strict_check

[bool, default=True] strict check data or not

Returns

dict

normalized data

dpgen.util.**sepline**(*ch='-'*, *sp='-'*, *screen=False*)

seperate the output by '-'

CHAPTER

ELEVEN

AUTHORS

- AnguseZhang
- Anopaul
- BaozCWJ
- Cloudac7
- EC2 Default User
- Ericwang6
- Futaki Haduki
- Futaki Hatuki
- Han Wang
- HuangJiameng
- Jinzh Zeng
- Jinzhe Zeng
- Kick-H
- LiangWenshuo1118
- Liu Renxi
- Liu-RX
- LiuGroupHNU
- Manyi Yang
- Pan Xiang
- Pinchen Xie
- Silvia-liu
- TaipingHu
- Tongqi Wen
- TongqiWen
- Waikit Chan
- Wanrun Jiang
- Yingze Wang

- Yixiao Chen
- Yongbin Zhuang
- Yuan Fengbo
- Yuan Fengbo ()
- Yunfan Xu
- Yunpei Liu
- Yuzhi Zhang
- Zhiwei Zhang
- baihuyu12
- cherushui
- cyFortneu
- deepmodeling
- dingzhaohan
- dinngzhaohan
- felix5572
- fqgong
- haidi
- hongriTianqi
- jameswind
- pee8379
- pxlxingliang
- robinzhuang
- robinzyb
- root
- shazj99
- tianhongzhen
- tuoping
- unknown
- yuzhi
- zhang yuzhi
- zhangbei07
- zhaohan
- zhengming-HIT
- zhenyu
- ziqi-hu
-

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Development of Range-Corrected Deep Learning Potentials for Fast, Accurate Quantum Mechanical/Molecular Mechanical Simulations of Chemical Reactions in Solution, Jinzhe Zeng, Timothy J. Giese, Şölen Ekesan, and Darrin M. York, Journal of Chemical Theory and Computation 2021 17 (11), 6993-7009

PYTHON MODULE INDEX

d

INDEX

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W