



Workflows Manual

Amsterdam Modeling Suite 2025.1

www.scm.com

Mar 25, 2025

CONTENTS

1	ChemTraYzer2	1
1.1	New in ChemTraYzer2-2023	1
1.2	Important information for using ChemTraYzer2	1
1.3	The ChemTraYzer2 algorithm	2
1.4	Distinguishing reactions with ChemTraYzer2	3
1.5	Using ChemTraYzer2 from the GUI	4
1.6	Tips for getting the most out of ChemTraYzer2	4
1.6.1	The MD simulation	4
1.6.2	ChemTraYzer2 Settings	4
1.7	Minimal input	4
1.8	Input options	5
1.9	Output	9
1.9.1	Summarizing reactions	9
	reaction_events.csv	9
	reactions.csv	9
1.9.2	Reaction frequency	10
	reaction_events_per_time.csv	10
	bond_change_events_per_time.csv	10
1.9.3	Molecular population analysis	10
	mol_statistics.csv	10
	mol_population.csv	10
1.9.4	Geometry output	11
1.9.5	Additional output files	11
1.10	References	11
2	OLED Deposition and Properties	13
2.1	General	13
2.1.1	New in AMS2025.1	14
2.1.2	New in AMS2023.1	15
2.2	Deposition	15
2.2.1	Basic input	16
2.2.2	Output	20
2.2.3	Deposition of host-guest materials	21
2.2.4	Deposition of interfaces	22
2.2.5	Restarting	24
2.2.6	LAMMPS offload	25
2.3	Properties	26
2.3.1	Basic input	27
2.3.2	Output	28
	Results directory	28

	Data on the HDF5 file	29
	Accessing the HDF5 file	31
	Summarizing the results	31
2.3.3	Additional settings	32
2.3.4	Parallelization	37
2.3.5	Restarting	37
2.4	Material database	38
2.4.1	Pure materials	39
2.4.2	Host-guest systems	40
3	Reactions Discovery	41
3.1	General	41
3.1.1	What's new in AMS2024?	41
3.2	Overview of workflow	42
3.3	Quickstart guide with example input file	45
3.4	Command to execute, parallelization	47
3.5	Engine settings	47
3.6	Molecular dynamics	48
3.6.1	Nanoreactor	50
3.6.2	Lattice deformation	52
3.6.3	Build the initial system	54
3.6.4	Fixed MD settings	56
3.6.5	Molecular dynamics restart	56
	Example: restart MD simulations after they exceeded walltime limit	57
	Example: Continue MD simulations for more steps	58
3.7	Network Extraction	58
3.7.1	Initial Network from ChemTrayzer2	58
3.7.2	Geometry Optimization	59
3.7.3	Molecular Charge Assignment	59
3.7.4	Manual specification of MD trajectories	59
3.8	Product Ranking	61
3.8.1	Reaction Energies	62
3.8.2	Product Cost	62
3.8.3	Product Stability	62
3.8.4	Reaction Balance	62
3.8.5	Example: ProductRanking from finished NetworkExtraction	62
3.9	Output	64
3.9.1	Reactants, products, unstable	64
3.9.2	KF output files	65
3.10	Reactions Discovery in Python (PLAMS)	75
3.11	Frequently Asked Questions	79
3.11.1	There are no reactions	79
3.11.2	There are too many reactions	79
3.11.3	The MD simulations are too slow	79
3.11.4	How should I set the density and compression factor?	80
3.11.5	The simulation explodes	80
3.11.6	How do I use computing resources efficiently?	80
4	Simple Active Learning	81
4.1	General	81
4.1.1	Licensing	83
4.1.2	What's new in Simple Active Learning?	83
	AMS2025	83
	AMS2024	83

4.2	Quickstart guide	84
4.3	Input	85
4.3.1	Overview	86
4.3.2	Initial reference data	87
	Generate initial reference data	88
	Load initial reference data	88
	Initial reference data input	88
4.3.3	When to run reference calculations (step sequence type)	90
	Step Type Geometric (default)	91
	Step Type Linear	92
	Step Type List	92
	Steps input	93
4.3.4	Success criteria	96
	Energy: total and relative	96
	Forces (gradients)	96
	Success criteria input	98
4.3.5	Reasonable simulation criteria (uncertainties, temperature, ...)	101
4.3.6	From scratch training	104
4.3.7	Output to save	106
4.3.8	At workflow end: retrain model, rerun simulation	107
	Retrain model	107
	Rerun simulation (final production simulation)	108
	AtEnd input	108
4.4	Output	109
4.5	Python Examples	109
4.5.1	Single molecule: setup and run	109
	Complete Python code	110
4.5.2	Single molecule: access results	111
	Complete Python code	111
4.5.3	Single molecule: Compare to M3GNet-UP-2022	114
	Complete Python code	114
4.5.4	Single molecule: Production simulation with retrained ML potential	116
	Complete Python code	116
4.5.5	Continue active learning with a new system or new simulation settings	118
	Complete Python code	119
4.5.6	Liquid water: diffusion coefficient, radial distribution function, density	120
	Complete Python code	122
4.5.7	Conformers: Active learning with CREST metadynamics and custom addition of data points	131
	Complete Python code	133
4.5.8	Li-vacancy diffusion in a solid electrolyte	143
	Complete Python code	144
4.5.9	Active Learning with uncertainties predicted from committee models	153
	Complete Python code	155
4.5.10	Ru/H introduction	161
4.5.11	Ru/H Part 1: Initial reference data from lattice optimization, volume scan, bond scan	162
	Complete Python code	163
4.5.12	Ru/H Part 2: Initial reference data from cartesian coordinate scans and bond scans	165
	Complete Python code	165
4.5.13	Ru/H Part 3: Initial reference data MD simulation + single-point replays	169
	Complete Python code	169
4.5.14	Ru/H Part 4: Initial training	171
	Complete Python code	171
4.5.15	Ru/H Part 5: Active learning for molecule gun MD	173
	Complete Python code	173

4.6	Python API	176
4.6.1	SimpleActiveLearningJob	177
4.6.2	SimpleActiveLearningResults	179
4.7	Frequently Asked Questions	180
4.7.1	What kind of properties can I fit?	180
4.7.2	Can I run on the GPU?	181
4.7.3	What kinds of MD simulations can I run?	181
4.7.4	Can VASP be used with Simple Active Learning?	182
	Index	183

CHEMTRAYZER2

ChemTraYzer2 (CT2) is a tool for post-processing reactive [molecular dynamics](#) (MD) trajectories. The purpose of CT2 is to detect and distinguish the reactive events that occur, construct a database of unique reactions from these events, and then calculate aggregate kinetic and population properties for the trajectory. Practically speaking, CT2 is capable of greatly simplifying MD simulations into a set of useful values such as reaction rate constants¹, net fluxes for all chemical species, and occurrence counts for all reactions. ChemTraYzer2 is the successor of ChemTraYzer.

See also:

The GUI tutorial [Detecting reactions with ChemTraYzer 2: Hydrogen combustion with ReaxFF](#) will show you how to set up and perform a ChemTraYzer2 analysis using the Graphical User Interface.

1.1 New in ChemTraYzer2-2023

- trajectory population analysis
- support for trajectories with a non-constant number of atoms
- an improved reaction rates calculator
- additional output files for population statistics

1.2 Important information for using ChemTraYzer2

Bond orders are necessary for post-processing MD trajectories with ChemTraYzer2. CT2 does *not* estimate bond orders but instead uses those computed by the MD engine used to run the simulation. Though most AMS [engines](#) can compute bond orders, there are some that cannot (see [Summary of engine capabilities](#)). CT2 can still be used with these engines, but a bond guessing algorithm must be used to estimate the bond orders. This can be done by specifying the following settings in the MD input.

Important: When preparing MD simulations for use with CT2, it is recommended to set the `BondOrders` variable in the `Properties` block to `Yes`. This will ensure that bond orders are calculated and stored. Depending on the chosen engine's capabilities, either it will supply bond orders or a bond guessing algorithm will be used. More information on this setting can be found in [here](#)

The quality of the ChemTraYzer2's analysis depends partially on the quality of the bond orders provided, but it is more dependent on the connectivity information (i.e., whether or not there is a bond between two atoms).

¹ L.C. Kroeger et al., *Assessing Statistical Uncertainties of Rare Events in Reactive Molecular Dynamics Simulations*, *Journal of Chemical Theory and Computation* 13, 3955-3960 (2017) (<http://dx.doi.org/10.1021/acs.jctc.7b00524>)

1.3 The ChemTraYzer2 algorithm

The following is a summary of the steps taken by ChemTraYzer2 while post-processing a MD trajectory. All of these steps are automatically conducted by ChemTraYzer2, so it is not necessary to understand them in detail in order to use ChemTraYzer2. This section is simply intended to provide the interested user with more technical information about the algorithm.

(1) Identifying all bond breaking and bond forming events in the MD trajectory

Bond changes are fundamental to chemical reactions, and the first step of ChemTraYzer2 is to analyze the MD trajectory and detect all bond change events that occur. ChemTraYzer2 defines a bond change event as either of the following:

- *Bond formation* – this occurs when the bond order between 2 atoms crosses the `BondFormationThreshold` parameter between 2 MD frames. More specifically, this means the bond order between 2 atoms must be below the `BondFormationThreshold` in one frame and then above it in the subsequent frame.
- *Bond breakage* – this occurs when the bond order between 2 atoms decreases to below the `BondBreakingThreshold`. It is defined analogously to *bond formation*.

(2) Filtering and combining all bond change events into stable reactions using the TStable criterion

Many bond change events in a MD trajectory might represent the formation of short-lived intermediates that do not need to be explicitly included in the complete reaction. These intermediates, though perhaps important to the mechanism, do not affect the overall reactants and products of a reaction and may introduce unwanted complexity to ChemTraYzer2 output. For this reason, the adjustable parameter `TStable` is used to filter out reactive intermediates which exist for an amount of time less than `TStable`. An example of using `TStable` to filter reactions is provided below.

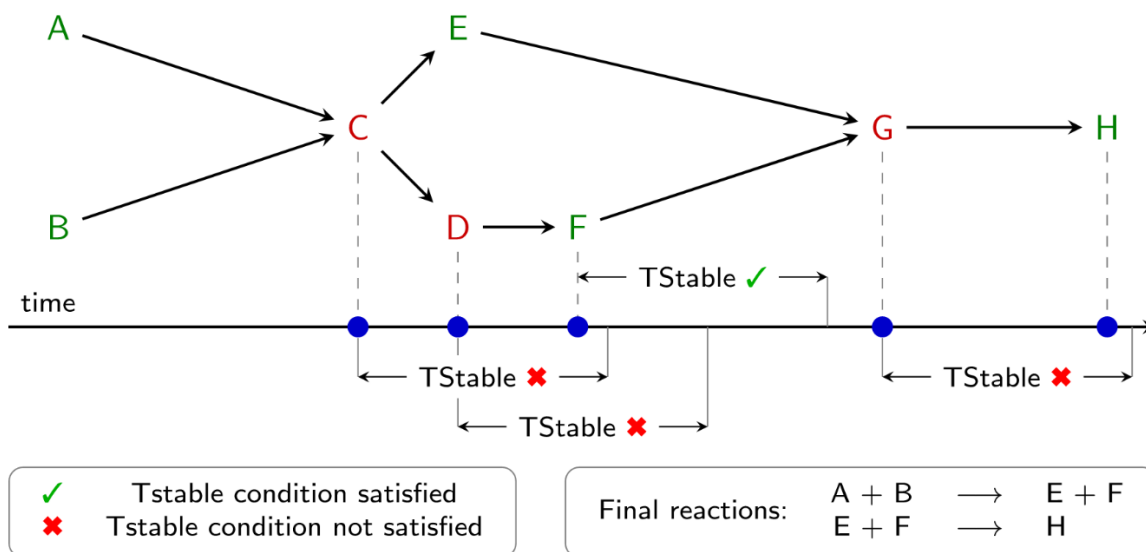
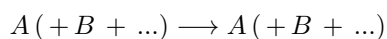


Fig. 1.1: An example of how ChemTraYzer2 filters reaction events based on the `TStable` criterion. In this reaction network, all species in red are determined to be short-lived intermediates and do not appear in the final reactions.

(3) Removing all reactions that have the same reactants and products

It is not uncommon for chemical equilibria to be observed in certain MD trajectories. Certain equilibria occur on a very short time-scale, meaning a series of bond change events may be filtered out using the `TStable` criterion. In these cases, the remaining reaction can have identical molecules on both sides of the reaction, as shown below.



These reactions are removed from the final reaction list as they have no effect on net species fluxes, rate constants, etc.

Note: Reactions that involve bond changes but result in the same molecules will also be filtered. For example, the following proton transfer will not be included in the final reaction list: $\text{H}_3\text{O}^+ + \text{H}_2\text{O} \rightarrow \text{H}_2\text{O} + \text{H}_3\text{O}^+$. Options for including these reactions will be present in the next version of CT2.

(4) Aggregating equivalent reactions

After the filtering steps are complete, all equivalent reaction events are combined into a set of unique reactions that have occurred in the MD trajectory. More specifically, the *reaction event* $A \rightarrow B$ may have happened multiple times in the trajectory, and each of these will count toward one occurrence of the $A \rightarrow B$ *reaction*. More detail about determining when two reactions (or molecules) are equivalent is provided in the following section.

1.4 Distinguishing reactions with ChemTraYzer2

In ChemTraYzer2, reactions are determined to be equivalent using a very straightforward condition: two reactions (R1 and R2) are equivalent if the sets of reactant/product molecules of R1 and the sets of reactant/product molecules of R2 are equivalent. Comparing reactions in this way requires defining the equivalence of two individual molecules, and this is more challenging to assess. In the original ChemTraYzer, molecule equivalence is determined via a comparison of canonical SMILES strings. Though SMILES can represent a large number of chemical structures, they fall short in representing the complete space of chemical reactions. For this reason, ChemTraYzer2 evaluates each molecule using a subgraph-based descriptor, which is generalizable to the complete reactive chemical space. ChemTraYzer2's subgraph descriptor builds local atomic environments using a breath-first search of each atom in a molecule, evaluates a unique hash value for each atom, and finally sums these hash values to produce a unique hash value for each unique molecule. This is summarized in the figure below.

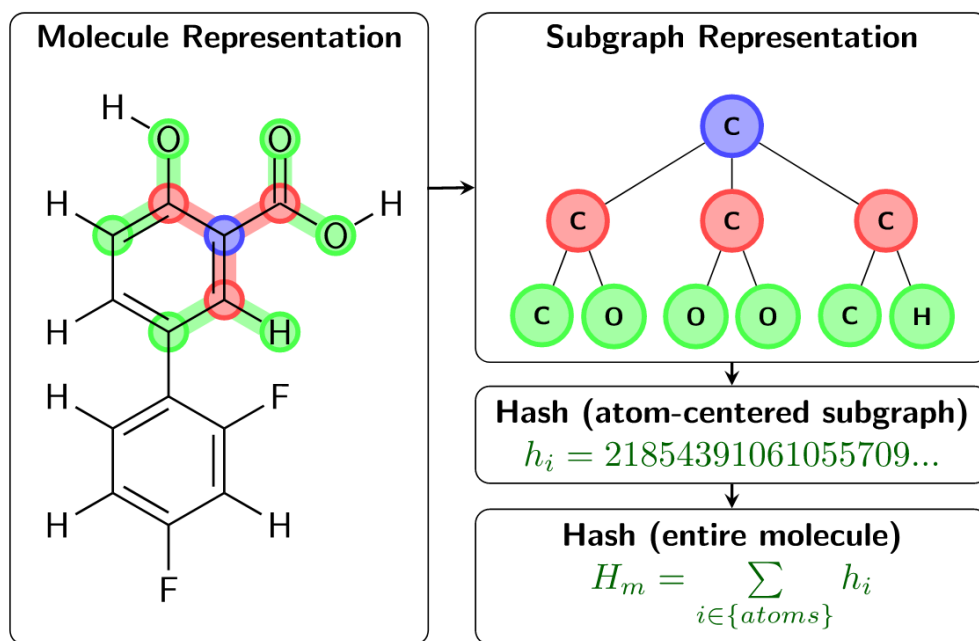


Fig. 1.2: The subgraph-based descriptors used to distinguish molecules in ChemTraYzer2

Note: The current version of the subgraph descriptors do not distinguish stereoisomers

1.5 Using ChemTraYzer2 from the GUI

ChemTraYzer2 is fully supported in the AMS GUI. A thorough description of using the GUI can be found in the [ChemTraYzer2 GUI tutorial](#).

1.6 Tips for getting the most out of ChemTraYzer2

1.6.1 The MD simulation

- It is important to ensure the simulation is on a **time scale that is long enough to observe multiple reaction events**. Multiple occurrences of reactions improve the accuracy of calculations of kinetic parameters such as reaction rate constants.
- **The sampling frequency of MD trajectories should be sufficiently small** to observe all important reactions. In AMS MD simulations, this is controlled by the `SamplingFreq` keyword in the `Trajectory` block (see the [Molecular dynamics](#) page for more details). If the sampling frequency is too large, important reaction events may not be detected by ChemTraYzer2, which will have an effect on the quality of the reported properties. A rough recommendation would be to set the sampling frequency to at most 10 for a time step of 0.25 fs, but the best value for this parameter depends on the temperature of the simulation.

1.6.2 ChemTraYzer2 Settings

- **Set the `TStable` parameter to an appropriate value.** Typically, the default value will work for many applications. However, the user can adjust this parameter to generate output on the spectrum between *many reactive intermediates* (low `TStable`) and *a summary of only the main reactions* (high `TStable`). Generally, it is best to adjust `TStable` to a level where all important intermediates are long-lived enough to appear in the final output. You may want to perform a few CT2 analysis using different values for `TStable` to see how this affects the results.
- **Set the `BondBreakingThreshold` and `BondFormationThreshold` parameters to appropriate values for the chemical system.** The default values are suitable for most types of systems, but these threshold values may need to be changed in certain cases (e.g., the MD engine calculates bond orders with a systematic error, bonds in the system have partial ionic character, etc.).
- **Set the rate confidence interval `RateConfidence` to adjust bounds for the reaction rate constants.** CT2 assumes the number of observed reactive events are distributed according to a Poisson distribution, where the expected value is used to calculate the reaction rate constant. The confidence interval specifies what ratio of the event counts will fall between the lower and upper bounds, with the condition that both bounds represent an equal number of events. Usually, a confidence interval of 95% is used, which corresponds roughly to 2σ in a normal distribution. For more details about this approach, see ^{Page 1, 1}.

1.7 Minimal input

This is the minimal input script for performing a chemtrayzer2 analysis of your MD trajectory:

```
#!/bin/sh

$AMSBIN/chemtrayzer2 << EOF
Trajectory
  Path path/to/the/ams/results/folder
```

(continues on next page)

(continued from previous page)

```
End
EOF
```

1.8 Input options

Several input options can be specified in the chemtrayzer2 input.

The trajectory the user wants to analyze can be specified in the `Trajectory` block:

```
Trajectory
  FinalFrame integer
  FirstFrame integer
  Path string
End
```

Trajectory

Type

Block

Description

Info regarding the trajectory to analyze.

FinalFrame

Type

Integer

Default value

-1

Description

Last frame of the trajectory to analyze.

FirstFrame

Type

Integer

Default value

1

Description

First frame of the trajectory to analyze.

Path

Type

String

Description

The path to ams results dir of an AMS calculation. This folder must contain a ams.rkf file.

Reaction detection options can be specified in the `ReactionDetection` block:

```
ReactionDetection
  BondBreakingThreshold float
  BondFormationThreshold float
```

(continues on next page)

(continued from previous page)

```
InitialBondThreshold float
TStable float
End
```

ReactionDetection**Type**

Block

Description

Parameters for the the reaction detection algorithm.

BondBreakingThreshold**Type**

Float

Default value

0.3

Description

The bond-order threshold for bond breaking. If the bond order of a bond goes below this value, the bond is considered broken.

BondFormationThreshold**Type**

Float

Default value

0.8

Description

The bond-order threshold for bond formation. If the bond order between two atoms goes above this value, then this will be considered to be a new bond.

InitialBondThreshold**Type**

Float

Description

The bond-order threshold for determining the connectivity for the first frame of the simulation. If not specified, the value in BondFormationThreshold will be used instead.

TStable**Type**

Float

Default value

10.0

Unit

fs

GUI name

T stable

Description

The minimum time for a molecule to be considered stable.

Options for the analysis of the reactions:

```

Analysis
  PerformAnalysis Yes/No
  RateConfidence float
End

```

Analysis**Type**

Block

Description

Statistical post-detection analysis, includes reaction coefficients calculation.

PerformAnalysis**Type**

Bool

Default value

Yes

Description

Determine the reaction rate coefficients and statistical errors for the detected reactions.

RateConfidence**Type**

Float

Default value

0.9

Description

Upper and lower bounds to the rate coefficients will be calculated for this confidence ($0 < \text{confidence} < 1$), assuming a Poisson distribution of the number of reactive events. A value of 0.9 means that the kinetics of 90% of events of one reaction can be described by a coefficient between the bounds.

Options for Output file writing:

```

Output
  CreateLegacyOutput Yes/No
  ShowReactionGraph Yes/No
  WriteEventsPerTime Yes/No
  WriteKF Yes/No
  WriteMolPopulation Yes/No
  WriteReactions Yes/No
  WriteXYZFiles Yes/No
End

```

Output**Type**

Block

Description

Settings for program output and output file generation.

CreateLegacyOutput**Type**

Bool

Default value

No

Description

Whether to save the reactions, species, and rates as 'reac.reac.tab', 'reac.spec.tab', and 'reac.rate.tab' in the same format as ChemTraYzer 1.

ShowReactionGraph

Type

Bool

Default value

No

Description

Whether or not to show the reaction graph at the end of the calculation. Requires the python library matplotlib to be installed.

WriteEventsPerTime

Type

Bool

Default value

No

Description

Write two .csv files that contain the number of reactions in every frame (reaction_events_per_time.csv) and the number of bond changes in every frame(bond_change_events_per_time.csv)

WriteKF

Type

Bool

Default value

No

Description

Whether to write output to KF

WriteMolPopulation

Type

Bool

Default value

No

Description

Write two .csv files: (1) mol_statistics.csv, which contains basic population statistics (counts, averages) for each unique species over the entire trajectory; and (2) mol_population.csv, which provides the count of each unique species in every frame.

WriteReactions

Type

Bool

Default value

Yes

Description

Write two .csv files that contain information about (1) all unique reactions (reactions.csv); and (2) all individual reaction events (reaction_events.csv).

WriteXYZFiles**Type**

Bool

Default value

No

Description

Write XYZ files (geometries) for detected species and XYZ movies for detected reactions into a subfolder named 'xyz'.

1.9 Output

1.9.1 Summarizing reactions

ChemTraYzer2 produces 2 main output files for summarizing reactions, `reaction_events.csv` and `reactions.csv`. These 2 files are produced with the option `WriteReactions` in the `Output` block.

`reaction_events.csv`

This file contains a list of all bond breaking or bond forming events. These events are complete reactions that occur for some specific set of molecules at some specific point in the trajectory. Various important properties are included in this file, a few of which are listed below.

- `Initial frame` – the MD frame at which the bond change event began
- `Final frame` – the MD frame at which the bond change event ended
- `Reactants/Products` – a SMILES-like representation of molecules involved in the reaction
- `Reactants atoms indices/Products atoms indices` – the atom indices of the molecules involved in the reaction

`reactions.csv`

This file contains aggregate information about all unique reactions that occurred in the trajectory. A few important properties contained in this file are listed below.

- `Rate constant` – the calculated value of the reaction rate constants. Note that the units for the reaction rate depend on the reaction order.
- `Number of events` – the number of times this reaction occurred in the trajectory
- `Reaction event indices` – the indices of all reactive events that are equivalent to this reaction. The indices correspond to indices in the `reaction_events.csv` file.

1.9.2 Reaction frequency

The option `WriteEventsPerTime` in the `Output` block will produce two files that detail the accumulated number of reactions and reaction events per frame over the entire trajectory.

`reaction_events_per_time.csv`

- `Frame` – the MD frame
- `Time` – the simulation time for the frame
- `Events` the number of reactions that *begin* in the specified frame

`bond_change_events_per_time.csv`

- `Frame` – the MD frame
- `Time` – the simulation time for the frame
- `Events` the number of bond change events that occur in the specified frame

1.9.3 Molecular population analysis

The option `WriteMolPopulation` in the `Output` block will produce two files that provide summary statistics for each unique molecule in the trajectory as well as population counts for all frames.

`mol_statistics.csv`

- `Molecule hash` – the hash value used to identify a molecule
- `SMILES` – the SMILES representation of a molecule, should one be available
- `Average count` – the average number of molecule over the entire trajectory
- `Average conc.` – the average concentration (in mol/L) of molecule over the entire trajectory
- `Mann-Kendall value"` – a value in the range [-1,1] that indicates whether a molecule behaves more like a reactant (with a maximum value of -1) or a product (with a maximum value of +1). Intermediates are expected to have values around 0.

`mol_population.csv`

- `Frame` – the MD frame
- `Time` – the simulation time for the frame
- `Count` the number of a particular molecule in a particular frame

1.9.4 Geometry output

The option `WriteXYZFiles` will produce xyz files for each unique molecule and a series of xyz frames for each unique reaction. These files are named according to the molecule and reaction indices and will be placed into a directory called `xyz`.

1.9.5 Additional output files

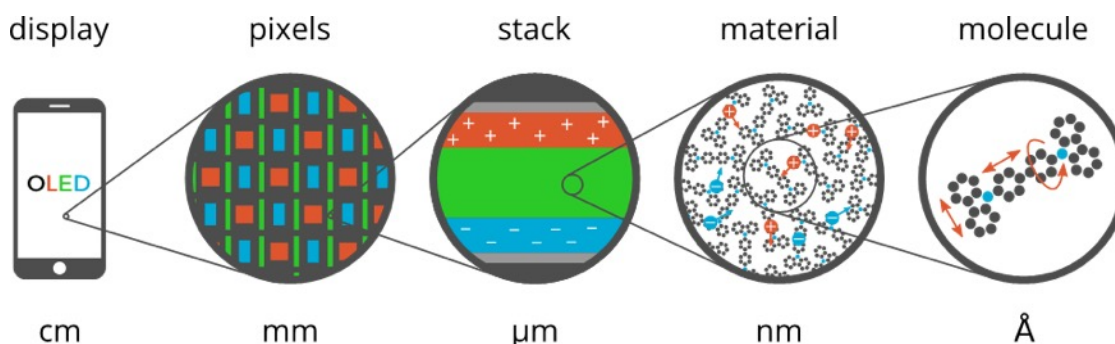
In addition to the main csv output files, ChemTraYzer2 generates a `gml` (https://en.wikipedia.org/wiki/Graph_Modelling_Language) file (`reaction_network.gml`) containing the full reaction network. At the moment, we don't offer any built-in tool for visualizing or manipulating this file. The savvy user might want to import and analyze the `.gml` file using the `networkx` (<https://networkx.org/>) python library or visualize it with third party graph visualization tools.

1.10 References

OLED DEPOSITION AND PROPERTIES

2.1 General

Starting with the 2022 release of the Amsterdam Modeling Suite we include a set of *workflow scripts* for multiscale OLED modeling. These workflows are developed and validated in close *collaboration* (<https://research.tue.nl/en/impacts/oled>) with the Eindhoven University of Technology to bridge the gap between ab-initio atomistic modeling of OLED molecules with AMS, and device level kinetic Monte Carlo simulations using our *Bumblebee code*. We aim to provide a fully integrated multiscale simulation platform for the digital screening and prediction of successful OLED materials and devices.



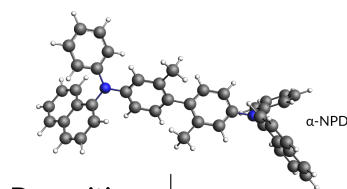
The Amsterdam Modeling Suite implements the atomistic simulation part of this multiscale toolchain in the form of two workflow scripts:

- **Deposition** (page 15)

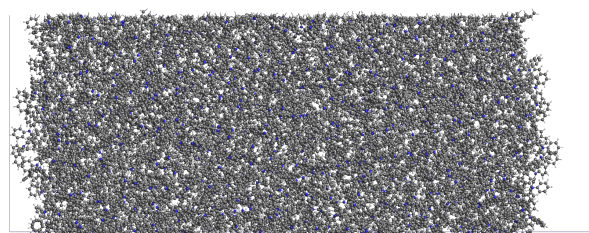
The first step simulates the growth of a thin film in a *molecular dynamics* and *force-bias Monte Carlo* calculation mimicking physical vapor deposition.

- **Properties** (page 26)

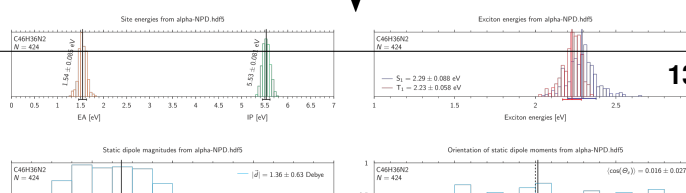
In the second step the morphology resulting from the deposition is used to obtain the distributions (and possibly spatial correlations) of molecular properties such as ionization potential, electron affinity and exciton energies at the DFT level. Each molecule's environment is taken into account in a polarizable QM/MM scheme using the *DRF model*.



Deposition



Properties



The output of the [Properties](#) (page 26) workflow is an [HDF5 file](#) (page 28) containing a summary of the calculated material parameters. This file can be opened in [AMSview](#) for a visualization of the results. The data can be imported into [Bumblebee](#) to perform simulations at the device level.

This manual page describes the technical details and options of the OLED workflow scripts. For a more hands-on introduction, you may want to start with the GUI tutorial, which will guide you through the entire workflow using the host-guest mixture of 95% CBP + 5% Ir(ppy)₃ as an example.

See also:

[Tutorial on multiscale modeling of OLED devices](#)

Note: The OLED workflow scripts use [ADF](#) and [DFTB](#). In addition to the Advanced Workflows and Tools license, you will therefore also need a license for ADF and DFTB in order to use the OLED workflows.

2.1.1 New in AMS2025.1

- **Improvements in the deposition workflow:**

- The option of *offloading the calculation of forces to LAMMPS* (page 25) has been further tested and optimized. It is no longer considered experimental and about ~4x faster than in AMS<2025. On systems with a GPU, this should allow for depositions with standard settings to finish in less than half a day.
- Error handling and error messages have been improved.

- **Improvements and fixes in the properties workflow:**

- The `NumSelectedMoleculesPerSpecies` keyword allows calculating properties only for a random subset of molecules per species. This can be used to quickly do “enough” molecules to get sufficiently good statistics as input for our [Bumblebee](#) kinetic Monte Carlo code.
- A summary with the results of the calculation is written to a YAML file that can be imported directly into [Bumblebee](#).
- Startup time for the properties workflow script has been reduced.
- The properties workflow is now more reliable in determining the SMILES strings of the different molecules.
- An issue in the properties workflow has been resolved that would lead to wrongly extracted, broken-up molecules for small boxes. This did not affect morphologies from depositions with the default settings, but could happen if the box was smaller than about twice the extent of a molecule.

2.1.2 New in AMS2023.1

- The exciton energies are now calculated using the PBE0 functional, and should be more accurate.
- The transfer integrals are now calculated with DFT and should be much more accurate than the GFN1-xTB calculated transfer integrals in AMS2022. (The GFN1-xTB transfer integrals were much too small.)
- The OLED material database has been updated and now contains the PBE0 exciton energies as well as the DFT transfer integrals. (The morphologies did not change with respect to the 2022 version.)
- Experimental: The deposition workflow can now *offload the calculation of forces to LAMMPS* (page 25), allowing much faster depositions if a GPU accelerated LAMMPS installation is available.
- **Various technical improvements to the properties workflow script:**
 - The HDF5 file is now created and populated with NaN values in the beginning of the workflow script. The NaN values are overwritten with the results as they come in, allowing the file to still be used in case the workflow is interrupted.
 - An interrupted workflow can now be restarted by specifying the output HDF5 file from the interrupted run with the `Restart` keyword. Any results on the HDF5 file will then not be recalculated.
 - The `SelectedMolecules` keyword allows you to calculate properties for just a subset of the molecules.
 - Result files of finished jobs will now be removed from disk after extracting the relevant properties. This massively reduces the disk space required to run the properties workflow script.
 - Should now consume less memory on the node executing the workflow script, and be more reliable.
 - Thanks to improvements in PLAMS, it is now much easier to run the workflow on multi-node allocations under SLURM.
- An issue with the automatic atom-typing in the deposition workflow has been fixed. (In AMS2022 nitrogen atoms would often get the `N_3` type, when they should have been `N_R`.)

2.2 Deposition

The deposition workflow implements a series of mixed *molecular dynamics* and *force-bias Monte Carlo* calculations to simulate the growth of a thin film with physical vapor deposition.

The *molecule gun* is used to shoot molecules at the substrate. Upon reaching the surface, the *force-bias Monte Carlo* method is used to accelerate the search for favorable adsorption sites. This process is repeated until a thin film of a user-defined thickness has grown on the substrate.

To make this process computationally more efficient, the deposition happens in so-called “cycles”. At the end of each deposition cycle, the bulk material at the bottom of the growing film is “trimmed off” and stored. Only the two top layers (each about 10 Å thick) are transferred to the next deposition cycle, where the lower of the two layers is frozen. This ensures that the trimmed off parts of the system fit perfectly together when the system is reassembled in the end of the simulation. By depositing in cycles we avoid simulating a lot of bulk material and are able to make the total computational cost linear in the thickness of the deposited film.

At the end of the simulation the layer deposited by the individual cycles are stacked and a short *molecular dynamics* calculation on the entire film is used to anneal it from the deposition temperature down to 300K at ambient pressure.

An entire deposition with 6 deposition cycles (and the final equilibration to room temperature) is shown in the video on the right.

2.2.1 Basic input

The minimal input to the deposition workflow just specifies what to deposit:

```
#!/bin/sh

DEPOSITION_JOBNAME=myDeposition $AMSBIN/oled-deposition << EOF

Molecule
  SystemName myMol
End

System myMol
  ...
End

EOF
```

The `Molecule` block is only really used when depositing mixed molecule materials, e.g. host-guest systems. This will be explained in a [separate section](#) (page 21) below. For a single molecule deposition there should just be one `Molecule` block that references the only `System` block by name via the `SystemName` keyword, as shown in the example above.

The `System` block used by the OLED deposition script closely follows the `System` block in the input for the AMS driver, but supports only a subset of the keywords:

System

Type

Block

Recurring

True

Description

Specification of the chemical system. For some applications more than one system may be present in the input. In this case, all systems except one must have a non-empty string ID specified after the `System` keyword. The system without an ID is considered the main one.

Atoms

Type

Non-standard block

Description

The atom types and coordinates. Unit can be specified in the header. Default unit is Angstrom.

GeometryFile

Type

String

Description

Read the geometry from a file (instead of from `Atoms` and `Lattice` blocks). Supported formats: `.xyz`

BondOrders

Type

Non-standard block

Description

Defined bond orders. Each line should contain two atom indices, followed by the bond order

(1, 1.5, 2, 3 for single, aromatic, double and triple bonds) and (optionally) the cell shifts for periodic systems. May be used by MM engines and for defining constraints. If the system is periodic and none of the bonds have the cell shift defined then AMS will attempt to determine them following the minimum image convention.

Just like in the AMS driver, as an alternative to the `System` block, you can also use the `LoadSystem` block to load a system directly from a `.rkf` file of a previous calculation.

The deposition workflow uses the `ForceField` engine for the molecular dynamics simulation of the physical vapor deposition. In order to also support the deposition of metal-containing compounds, we use the UFF force field with the `UFF4MOF-II` (<https://doi.org/10.1021/acs.jctc.6b00664>) parametrization for the deposition. As with any calculation with the `ForceField` engine you may manually provide (UFF4MOF-II) atom-types, atomic charges and bond orders in the input file:

```
System
  Atoms
    C [...] ForceField.Type=C_R ForceField.Charge=-0.1186
    N [...] ForceField.Type=N_R ForceField.Charge=-0.2563
    H [...] ForceField.Type=H_  ForceField.Charge=+0.1021
    [...]
  End
  BondOrders
    1 2 1.0
    1 5 1.5
    1 6 1.5
    [...]
  End
End
```

Whatever is not specified in the input will automatically be determined: the input system is optimized with ADF using the S12g exchange-correlation functional with a TZP basis set. At the optimized geometry, the `Charge Model 5` is used to calculate the atomic charges, while the rounded `Nalewajski-Mrozek bond orders` determine the topology. See the [ADF manual](#) for details on the calculation of charges and bond orders. Finally, using the topology determined by the calculated bond orders, the `automatic UFF atom-typing` that is built into the `ForceField` engine is used to determine the atom-types.

If you want to make sure the correct atom-types and bonds are used in your calculation, we recommend building the system in `AMSinput`, where you can visually check the bond orders and atom-types to make sure they are correct. The result can then be exported into a file as a `System` block via **File → Export coordinates → .in**. For the atomic charges we recommend relying on the automatic calculation with ADF. (Just make sure the `ForceField.Charge` suffixes are not included in the atom block. Their absence will trigger the automatic charge calculation with ADF.)

By default a box of 60 x 60 x 120 Å is deposited. The first two dimensions give the surface area of the deposited layer, while the third dimension is the thickness of the layer. The size of the deposited box can be changed using the `Size` keyword in the `Box` block:

```
Box
  Size 60 60 120
End
```

Box

Type

Block

Description

Specifications of the box into which the material is deposited.

Size

Type

Float List

Default value

[60.0, 60.0, 120.0]

Unit

Angstrom

GUI name

Box size

Description

Specify the desired size of the box. The final deposited box may have a different size. The x- and y-axis are perpendicular to the direction of deposition, so these may be regarded as the width of the growing layer. The z-axis is the direction along which the deposition happens, so this determines the thickness of the deposited layer. Note that the x- and y-axis will be ignored if a custom substrate is used: the area of the box is then determined by the lattice of the substrate. The z-axis can still be freely chosen, but should be large enough that there is enough space for the substrate itself and to deposit more molecules on top of it.

With sizes typical for molecules used in OLED devices, the default box size results in a deposition of ~500 molecules. Note that the computational time of a deposition scales linearly with the thickness of the layer, but quadratically with the surface area. This is because a larger area requires both the deposition of more molecules to fill the box, but also makes each MD step more expensive as more molecules have to be simulated at the same time. When increasing the thickness of the layer, molecules at the bottom are first frozen, and later removed from the simulation altogether, giving an overall linear scaling.

The temperature at which the deposition is performed can be configured in the `Deposition` section.

```
Deposition
  Temperature float
End
```

Deposition**Type**

Block

Description

Specifies the details of how molecules are deposited.

Temperature**Type**

Float

Default value

600.0

Description

The temperature at which the deposition happens.

Finally, there are a couple more technical options in the `Deposition` section, that we suggest to leave at their default values.

```
Deposition
  Frequency integer
  TimeStep float
  ConstrainHXBonds Yes/No
```

(continues on next page)

(continued from previous page)

```

NumMolecules integer
End

```

Deposition**Type**

Block

Description

Specifies the details of how molecules are deposited.

Frequency**Type**

Integer

Default value

10000

Description

The frequency in MD steps at which new molecules will be added to the system.

TimeStep**Type**

Float

Default value

1.0

Unit

Femtoseconds

Description

The time difference per step.

ConstrainHXBonds**Type**

Bool

Default value

Yes

GUI name

Constrain H-* bonds

Description

Constrain the bond length for all H-* bonds (i.e. any bond to a hydrogen atom). Doing this allows choosing a larger time step. If this option is disabled, the TimeStep needs to be reduced manually.

NumMolecules**Type**

Integer

Description

The number of molecules that we will try to deposit. If not specified the number will be determined automatically such that the box becomes approximately full.

2.2.2 Output

Running the `oled-deposition` workflow script creates a single directory in which you can find all results of a deposition. By default this directory is named `oled-deposition.results`, but in order to avoid name clashes, that location can be changed with the `AMS_JOBNAME` environment variable. The example below will collect all results in the directory `myLayer.results`:

```
#!/bin/sh

AMS_JOBNAME=myLayer $AMSBIN/oled-deposition << EOF
...
EOF
```

Let us go through all files and folders in the working directory in the order in which they are created.

Firstly, the working directory contains the `oled-deposition.log` logfile. The contents of the logfile are identical to what you see on standard output when running the `oled-deposition` workflow.

The deposition workflow starts with a couple of calculations on single molecules in vacuum. Each of them runs in a separate folder, in which you can find the usual [AMS output files](#) (such as `ams.rkf`):

```
myMol.dft_opt/
myMol.ff_opt/
myMol.equilibrate_ff_input_molecule/
```

Here `myMol` corresponds to the name of the molecule that was used in the input file. The `myMol.dft_opt` directory contains the results of the initial geometry optimization with ADF, which is used to determine the atomic charges and bond orders if these were not specified in the input. The `myMol.ff_opt` directory contains the results of a subsequent geometry optimization using the [ForceField](#) engine with the UFF4MOF-II forcefield. Finally in the `myMol.equilibrate_ff_input_molecule` directory a short MD simulation at the deposition temperature is performed to equilibrate the molecule to the desired temperature. We suggest visualizing the trajectory of this equilibration in [AMSmovie](#) to make sure the molecule does not undergo unexpected conformational changes that could be caused by wrong atom-types or bonds. If the molecule behaves strangely (or falls apart) at this point, one may need to go back and [assign atom-types and bonds manually](#) (page 17) in the input.

When [depositing mixtures](#) (page 21) you will see multiple instances of the three directories above: one for each deposited species.

Once all the preparatory work is done, the actual [deposition cycles](#) (page 15) each write a folder and (upon completion of the cycle) two files:

```
depo_cycle_1/
depo_box.1.in
depo_box.1.xyz
```

You can follow the progress of your deposition by opening the `ams.rkf` in the last `depo_cycle_*/` directory. The `depo_box.*.in` and `depo_box.*.xyz` files contain the entire morphology deposited so far: by visualizing them in order you can watch your material grow!

Important: The files with the `.in` extension contain the System geometry in the form of a [System block](#). This format contains bond orders, force field atom types as well as atomic charges. It can be opened in [AMSinput](#) and [PLAMS](#) and should be the preferred format when working with the OLED workflow scripts in AMS. The `.xyz` file is in [extended XYZ format](#) and does *not* contain that extra information. Always use a `.in` file when transferring a system from one script to the next, e.g. when going from the OLED deposition to the [OLED properties workflow](#) (page 26).

Once all molecules have been deposited the entire box is annealed from the deposition temperature down to room temperature. This creates one directory and (upon completion) `.in` and `.xyz` files containing the annealed morphologies:

```

equilibrate_box/
equil_box.in
equil_box.xyz

```

The last step is to take the room temperature morphology and perform a geometry optimization on it. This essentially removes all thermal vibrations and results in a geometry that is relaxed at the force field level. As you might expect, the last step also produces a folder and (upon completion) `.in` and `.xyz` files:

```

optimize_box/
morphology.in
morphology.xyz

```

It is up to the user to decide whether to continue to the OLED [properties workflow](#) (page 26) with the morphology from `equil_box.in` (equilibrated to 300K) or `morphology.in` file (fully relaxed). We recommend using the fully relaxed morphology though. We also used fully relaxed morphologies for the generation of the [OLED material database](#) (page 38).

2.2.3 Deposition of host-guest materials

A deposition of host-guest materials can easily be done by specifying multiple `Molecule` and `System` blocks in the input. The following runscript generates a 95 to 5 mixture (by number of molecules) of two compounds:

```

#!/bin/sh

AMS_JOBNAME=host_guest $AMSBIN/oled-deposition << EOF

Molecule
  SystemName myHost
  MoleFraction 0.95
End
Molecule
  SystemName myGuest
  MoleFraction 0.05
End

System myHost
  ...
End
System myGuest
  ...
End

EOF

```

```

Molecule
  MoleFraction float
  SystemName string
End

```

Molecule

Type
Block

Recurring
True

GUI name

Molecules

Description

Specification of the molecule to be deposited.

MoleFraction**Type**

Float

Default value

1.0

GUI name

Molar fraction

Description

The relative occurrence of the molecule with regard to other deposited species. Only relevant for mixed molecule depositions.

SystemName**Type**

String

GUI name

Molecule

Description

String ID of a named [System] to be inserted. The lattice specified with this System, if any, is ignored and the main system's lattice is used instead.

You can have an arbitrary number of `Molecule` blocks in your input to deposit multi-component mixtures. Obviously, the box you are depositing must be large enough that it still contains at least a few molecules of the rarest component.

Note that multiple `Molecule` and `System` blocks can also be used to deposit different conformers of the same compound. While conformational changes can in principle happen over the course of the MD simulation, it may be a good idea to deposit a mixture of conformers directly if their geometries are very different.

2.2.4 Deposition of interfaces

By default the deposition will use a single graphene layer as a substrate. The graphene layer is removed after the first *deposition cycle* (page 15) and will not be included in the output morphologies, i.e. the `.in` files in the *working directory* (page 20). Note that the graphene layer is **not** present in the annealing of the entire morphology from deposition temperature to 300K, which is performed at the end of the workflow. The result of this is that both the bottom and top of the deposited thin-film by default represents an interface between the material and a vacuum.

Instead of depositing on a clean graphene sheet, the deposition workflow also supports custom substrates. This is intended to be used for depositing a thin film of one material on top of another material and allows users to study the interface between the two. A custom substrate is set up using the `Substrate` and `SubstrateSystem` keys in the `Box` block.

```
Box
  Substrate [Graphene | Custom]
  SubstrateSystem string
End
```

Box**Type**

Block

Description

Specifications of the box into which the material is deposited.

Substrate**Type**

Multiple Choice

Default value

Graphene

Options

[Graphene, Custom]

Description

The substrate on which to grow the layer.

SubstrateSystem**Type**

String

GUI name

Custom substrate

Description

String ID of a named [System] to be used as a substrate. (This is only used when the Substrate key is set to Custom.)

Here the value of the SubstrateSystem refers to a named System block in the input, representing the geometry of the substrate. The following example shows how to deposit a molecule B on top of a substrate of molecule A:

```
#!/bin/sh

AMS_JOBNAME=molB_on_molA $AMSBIN/oled-deposition << EOF

Molecule
  SystemName molB
End
System molB
...
End

Box
  Size 0 0 240
  Substrate Custom
  SubstrateSystem molA_substrate
End

System molA_substrate
  Atoms
    ...
  End
  BondOrders
    ...
  End
  Lattice
    ...
  End
End
```

(continues on next page)

(continued from previous page)

EOF

The contents of the block `System molA_substrate` should be obtained by first running a deposition of molecule A: just use the `System` block found in e.g. the `equil_box.in` file of that deposition as the custom substrate for the next job. (Note that no attempt will be made to automatically determine atomic charges, bond orders, or force-field atom types for the molecules in the substrate. Taking the `System` block from the results of an earlier deposition is the easiest way to ensure that you are using exactly the same bonds, atom types and charges for the substrate molecules in the new calculation.)

Warning: The custom substrate option can currently not be used to deposit thin films on top of crystalline materials. The substrate must consist of individual molecules and be >20 Å thick, so that it can be split into a frozen (lower) and a thermostatted (upper) layer.

Note that the `Box%Size` in the x- and y-direction is ignored when using a custom substrate: the size of the custom substrate is used instead. The thickness of the layer can be set manually when using a custom substrate, but it needs to accommodate both the already existing substrate as well as the newly grown film on top. Assume that the thickness of the substrate film is 120 Å in the example above. By setting the the z-value of the `Box%Size` to 240 Å, we will have space to accommodate the substrate and then grow another layer of 120 Å thickness on top of it. Note that while the default graphene layer is *removed* from the morphology, a custom substrate will be *included* in the morphology.

2.2.5 Restarting

The OLED workflow scripts are based on the [PLAMS](#) scripting framework. As such it can rely on the [PLAMS rerun prevention](#) to implement restarting of interrupted depositions.

The easiest way to restart a deposition is to include the `--restart` (or short: `-r`) command line flag:

```
#!/bin/sh
AMS_JOBNAME=myDeposition $AMSBIN/oled-deposition --restart << EOF
...
EOF
```

This first (interrupted) run will have created the `myDeposition.results` directory. Running the above script again will move that directory to `myDeposition.results.bak` and reuse all successful jobs from the first run. (People already familiar with PLAMS will recognize that this works just like the `-r` flag on the [PLAMS launch script](#).) Note that this does not restart the previous deposition precisely at the point where it was interrupted. Instead it restarts from the beginning of the last *deposition cycle* (page 15).

When running a deposition workflow on a batch system such as SLURM, you may want to consider always including the `--restart` flag in your runscript. It is not a problem if there are no previous results to restart from, but in case your job gets interrupted and automatically rescheduled, the `--restart` flag will make sure that it continues (approximately) from where it stopped.

There is also the `--load` (or short: `-l`) command line flag:

```
#!/bin/sh
AMS_JOBNAME=newDepo $AMSBIN/oled-deposition -l oldDepo.results << EOF
...
EOF
```

While this can be used to accomplish the same thing the `--restart` flag would do, its best use is to specify a directory of a previous deposition of the same molecules. This can save you the initial step of doing the DFT calculations in order to determine the atomic charges and bonds. A perfect use for this is when you have already *deposited a mixture* (page 21), and later decide to change the ratio between the compounds: by specifying the results directory of the first deposition, the initial DFT calculations can be skipped entirely. (Again, people already familiar with PLAMS will recognize that this works just like the `-l` flag on the [PLAMS launch script](#).)

2.2.6 LAMMPS offload

The OLED deposition workflow supports offloading the calculation of the forces to a local LAMMPS installation. This can easily speed up the deposition by a factor of 10 and more. If a GPU is available then another factor of 2 can be achieved.

Note: Before enabling offloading, you will need to install LAMMPS and set up the environment as described in [Setting up LAMMPS](#).

If a local LAMMPS installation is available, it can easily be used through the following keywords in the OLED deposition input file.

```
LAMMPSOffload
  Enabled Yes/No
  UseGPU Yes/No
  UseGPUForKSpace Yes/No
  UseGPUForNeighbor Yes/No
  UseOpenMP Yes/No
End
```

LAMMPSOffload

Type

Block

Description

Offload the calculation to LAMMPS via AMSPipe.

Enabled

Type

Bool

Default value

No

Description

Enable offloading the force field evaluation to LAMMPS instead of handling it internally in AMS.

UseGPU

Type

Bool

Default value

No

GUI name

Use GPU

Description

Accelerate LAMMPS calculations using a GPU. Requires a LAMMPS library built with the GPU package.

UseGPUForKSpace**Type**

Bool

Default value

Yes

Description

When UseGPU is enabled, also use the GPU to accelerate reciprocal space electrostatic interactions. Disabling this can improve performance on less powerful GPUs.

UseGPUForNeighbor**Type**

Bool

Default value

Yes

Description

When UseGPU is enabled, also use the GPU to accelerate neighbor searches. Disabling this can improve performance on less powerful GPUs.

UseOpenMP**Type**

Bool

Default value

No

GUI name

Use OpenMP

Description

Parallelize LAMMPS calculations using OpenMP threading. Requires a LAMMPS library built with the OMP package.

2.3 Properties

The properties workflow is used to obtain distributions (and possibly spatial correlations) of molecular properties such as the ionization potential and electron affinity from the morphology. To accomplish this, it will perform DFT calculations on all the individual molecules from the morphology, taking their environment into account in a QM/MM calculation.

The exact workflow (with all default settings) is as follows:

1. For each molecule in the box, do a quick DFT calculation with **LDA** and a **DZP basis** and use the **MDC-D charge model** to determine atomic charges. These charges will be used for the electrostatic part of the embedding potential in the next step.
2. For each molecule in the box:
 - Determine which other molecules to consider as the environment. By default all molecules within 15 Å (atom-atom distance) are considered.
 - **Individually, for neutral molecule, cation, and anion:**

- Optimize geometry of central QM molecule in frozen MM environment using GFN1-xTB and UFF4MOF-II with electrostatic embedding in the Hybrid engine.
 - Do a DFT single point on the optimized geometry using PBE and an all-electron TZ2P basis. The environment is taken into account using a polarizable DRF embedding.
- Calculate the (approximately) adiabatic ionization potential and electron affinity from the differences in total energy with respect to the neutral system.
 - Calculate exciton energies and transition dipole moments of the molecule with TD-DFT using the PBE0 functional. This calculation is performed on the optimized geometry of the neutral molecule obtained earlier, and the environment is again taken into account using a polarizable DRF embedding.
3. For all pairs of neighboring molecules (within 4 Å atom-atom distance of each other) calculate the electron and hole charge transfer integrals with DFT.

What is described above is the workflow with all default settings. Various aspects of this (such as the ranges) can be tweaked from the input, as shown in the [Settings section](#) (page 32) below.

2.3.1 Basic input

The simplest possible input for the oled-properties workflow script is just a single System block.

```
#!/bin/sh

$AMSBIN/oled-properties << EOF

System
  Atoms
    ...
  End
  Lattice
    ...
  End
  [BondOrders
    ...
  End]
End

EOF
```

Obviously, the Atoms and Lattice blocks are required, while the BondOrders block is optional. If the bond orders are present, they will be used to determine which parts of the system are connected, which ultimately determines which sets of atoms are considered distinct molecules. If the BondOrders block is not present, the bonds will be guessed. Since we only care about which atoms are bonded at all, and not on details such as the bond order, this should work quite reliably.

Nevertheless, if the morphology was obtained with the AMS [deposition workflow](#) (page 15), we can use the fact that it writes out the morphology as a .in file containing exactly the System block we need. Basically, we use the morphology.in output file of the deposition as the input for the properties script.

```
#!/bin/sh

$AMSBIN/oled-deposition << EOF
... see oled-deposition manual page ...
EOF

$AMSBIN/oled-properties < oled-deposition.results/morphology.in
```

This has the advantage that the bonds are guaranteed to be transferred without change between the two workflows.

By default the properties are calculated for all molecules in the morphology, but this can be limited with the following keywords:

NumSelectedMoleculesPerSpecies

Type

Integer

Description

Number of molecules per species to calculate properties for. Around 50 molecules per species should be enough to estimate distribution means and standard deviations as input for the Bumblebee KMC code. Mutually exclusive with the `SelectedMolecules` keyword. If neither this key nor `SelectedMolecules` is present, all molecules will be selected.

SelectedMolecules

Type

Integer List

Description

Indices of the molecules to calculate properties for. Note that indexing starts at 0. Mutually exclusive with the `NumSelectedMoleculesPerSpecies` keyword. If neither this key nor `NumSelectedMoleculesPerSpecies` is present, all molecules will be selected.

2.3.2 Output

Results directory

Running the `oled-properties` workflow script creates a single directory in which you can find all results of the calculation. By default this directory is named `oled-properties.results`, but in order to avoid name clashes, that location can be changed with the `AMS_JOBNAME` environment variable. The example below will collect all results in the directory `myMaterial.results`:

```
#!/bin/sh
AMS_JOBNAME=myMaterial $AMSBIN/oled-properties << EOF
...
EOF
```

This will create the `myMaterial.results` directory with the following files:

```
myMaterial.results/
├─ oled-properties.log
├─ oled-properties.rkf
├─ properties.hdf5
└─ properties.yml
```

The primary output file is the `properties.hdf5` file, which we will discuss in the next section. The `properties.yml` file contains a summary of the results that can be directly used as input for the [Bumblebee](#) kinetic Monte Carlo code.

While the workflow is still running, you will also find subdirectories with the output of the individual DFT calculations in the results directory. By default these subdirectories are deleted as soon as the relevant properties have been extracted. Only the output of failed calculations is kept to aid in debugging any issues. What is kept or deleted can be configured with the following keyword:

StoreResultFiles

Type

Multiple Choice

Default value

Failed

Options

[None, Failed, All]

Description

Whether to keep the full result files from all the individual jobs. By default the result files from all jobs for a particular molecule will be deleted after all relevant results have been extracted and stored on the HDF5 file. Note that keeping the full results for all molecules can easily require hundreds of gigabytes of storage space.

Data on the HDF5 file

The main output file of an OLED properties calculation is a small [HDF5](https://en.wikipedia.org/wiki/Hierarchical_Data_Format) file (https://en.wikipedia.org/wiki/Hierarchical_Data_Format) called `properties.hdf5`. It contains the results that are of interest for the design of OLED materials, such as site energies, exciton energies, (transition) dipole moments, etc.

The following groups and datasets can be found on the HDF5 file. (Note that all arrays on the HDF5 file are indexed starting from zero.)

The `species` group contains information about the different molecular species making up the morphology. There are two arrays in the `species` group whose size is equal to the number of different species (`numSpecies`):

`species.name`

An array of human readable names identifying the molecular species making up the morphology. Currently this is just the molecular formula in [Hill notation](https://en.wikipedia.org/wiki/Chemical_formula#Hill_system) (https://en.wikipedia.org/wiki/Chemical_formula#Hill_system).

`species.smiles`

An array of [SMILES](https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system) (https://en.wikipedia.org/wiki/Simplified_molecular-input_line-entry_system) strings for the different molecular species. May contain a dummy value in case the determination of the SMILES string from the 3D structure fails for a species.

The `molecules` group contains the complete geometrical description of the morphology. It contains a number of arrays, (almost) all of which have the total number of molecules (`numMolecules`) as their size:

`molecules.species`

An array of integers denoting the species of each molecule. Used to index into the arrays in the `species` group.

`molecules.lattice`

(3 x 3) array containing the lattice vectors in Ångstrom.

`molecules.position`

(`numMolecules` x 3) array containing the center of mass positions of all molecules in Ångstrom. Note that all center of mass positions are within the parallelepiped spanned by the lattice vectors, i.e. all fractional coordinates are in the [0,1] range.

`molecules.atoms`

This is a `numMolecules` sized 1D array, where each element itself is an array of `string, float, float, float` tuples representing `symbol, x, y, z`. The x, y and z coordinates are given in Ångstrom.

`molecules.bonds`

This is a `numMolecules` sized 1D array, where each element itself is an array of `int, int, float` tuples representing `atom1, atom2, bondOrder`. Here `atom1` and `atom2` are indices into the corresponding element

of the `molecules.atoms` array. The `bondOrder` is a floating point number, where the value of 1.5 is used to represent an aromatic bond.

The site energies are contained in the `energies` group on the HDF5 file:

`energies.IP`

A `numMolecules` sized array containing the first ionization potential for each molecule in eV.

`energies.EA`

A `numMolecules` sized array containing the first electron affinity for each molecule in eV.

`energies.HOMO`

A `numMolecules` sized array containing the Kohn-Sham orbital energy of the highest occupied orbital in eV. If requested via the `NumAdditionalOrbitalEnergies` keyword in the *input* (page 32) of the properties workflow, more arrays of this type (HOMO-1, HOMO-2, ...) may exist and contain the orbital energies of lower lying occupied orbitals.

`energies.LUMO`

A `numMolecules` sized array containing the Kohn-Sham orbital energy of the lowest unoccupied orbital in eV. If requested via the `NumAdditionalOrbitalEnergies` keyword in the *input* (page 32) of the properties workflow, more arrays of this type (LUMO+1, LUMO+2, ...) may exist and contain the orbital energies of higher lying virtual orbitals.

Similarly the exciton energies (in eV) can be found in the `exciton_energies` group. If the calculation of exciton energies was disabled by setting `NumExcitations` to 0 in the *input* (page 32), this information is not present.

`exciton_energies.S1`

Energies of the first excited singlet state (S1) with respect to the ground state. Higher singlet excitation energies may be found in more arrays of this type (S2, S3, ...) if their calculation was requested by setting `NumExcitations` to a value larger 1.

`exciton_energies.T1`

Energies of the first excited triplet state (T1) with respect to the ground state. Higher triplet excitation energies may be found in more arrays of this type (T2, T3, ...) if their calculation was requested by setting `NumExcitations` to a value larger 1.

Static dipole moments and transition dipole moments (in Debye) can be found in their respective groups:

`static_multipole_moments.dipole_moment`

(`numMolecules` x 3) array containing the dipole moment vectors for each molecule.

`transition_dipole_moments.S1_S0`

(`numMolecules` x 3) array containing the transition dipole moment vectors for the S0 → S1 transition for each molecule. Transition dipole moments for higher singlet excitations may be found in more arrays of this type (S2_S0, S3_S0, ...) if their calculation was requested by setting `NumExcitations` to a value larger 1.

If the calculation of transfer integrals is requested with the `TransferIntegrals%Type` key in the *input* (page 32), the `pairs` and `transfer_integrals` groups will also be available on the HDF5 file, containing the following datasets:

`pairs.indices`

A (`numPairs` x 2) array of integers containing the molecule indices for all pairs of molecules that were considered close enough to trigger the calculation of transfer integrals between them.

`transfer_integrals.electron`

A `numPairs` sized array containing the transfer integral (in eV) for electrons between each pair.

`transfer_integrals.hole`

A `numPairs` sized array containing the transfer integral (in eV) for holes between each pair.

Accessing the HDF5 file

The easiest way to view the data from the HDF5 file is to open it in the GUI using the [AMSview](#) module. There you can easily plot histograms of all the calculated properties, but also visualize the spatial distribution of the properties.

For more custom-built analysis, the HDF5 file can easily be opened from Python using the [h5py](https://www.h5py.org/) (<https://www.h5py.org/>) library, which is included in the [AMS Python Stack](#). The following code snippet shows how to calculate the mean and standard deviation of the ionization potential:

```
import h5py

with h5py.File("properties.hdf5", "r") as f:
    IPs = f['energies']['IP'][:]
    print("IP = ", IPs.mean(), "±" , IPs.std())
```

The above snippet is only suitable for calculations of pure compounds, as we are calculating the mean and standard deviation over *all* molecules, not taking their species into account. For *mixtures* (page 21), calculating these properties *per species* would be much more useful. This can easily be accomplished by using an appropriate mask on the `IPs` array for the calculation of the mean and standard deviation:

```
import h5py
import numpy as np

with h5py.File("properties.hdf5", "r") as f:
    IPs = f['energies']['IP'][:]
    speciesIDs = f['molecules']['species'][:]

    for specID, specName in enumerate(f['species']['name']):
        mask = (speciesIDs==specID) & (~np.isnan(IPs))
        print(specName)
        print("IP = ", IPs[mask].mean(), "±" , IPs[mask].std())
```

Note how we also use the mask to exclude all NaN elements in the array from the calculation of the mean and standard deviation. Occasional NaN values in the arrays on the HDF5 file indicate that a property could not be calculated for a molecule because the job for it crashed or failed in some other way. This is not a problem as long as it happens only rarely, but the NaN values need to be excluded from the analysis.

Summarizing the results

Often one is not interested in all the values for the individual molecules, but just the centers and widths of the distributions. Such a summary is already printed at the end of the `oled-properties` workflow script. Using the Python [OLEDPropertiesSummary](#) (page 31) class, such a summary can easily be produced from an HDF5 file.

```
class OLEDPropertiesSummary(f, outlier_Zmax=16)
```

A summary of the results of the OLEDProperties workflow.

```
__init__(f, outlier_Zmax=16)
```

Calculates a summary of the results on an HDF5 file.

Parameters

- `f` (`h5py.File`) – The HDF5 file to summarize.
- `outlier_Zmax` (`float`, *optional*) – The modified Z score above which a datapoint is excluded as an outlier. Defaults to 16.

`__str__()`

Produces a human readable summary of the results.

The same summary is also printed at the end of an `oled-properties` workflow run.

`as_yaml()`

Returns a a summary of the calculation in a YAML format that can be imported into Bumblebee Web.

This is for example useful to produce the Bumblebee compatible YAML files from already existing HDF5 files.

```
import h5py
from scm.oledtools import OLEDPropertiesSummary

summary = OLEDPropertiesSummary(h5py.File("oled-properties/properties.hdf5"))
print(summary.as_yaml())
```

Note that starting with AMS2025, the `properties.yml` file is automatically produced at the end of the `oled-properties` workflow script.

2.3.3 Additional settings

The OLED properties workflow script has a few options that determine what properties will be calculated and/or written to the *HDF5 file* (page 29):

```
NumAdditionalOrbitalEnergies integer
NumExcitations integer
TransferIntegrals
  Include
    Cutoff float
    Metric [CoM | Atoms | Atoms_noH]
  End
  Exclude
    Cutoff float
    Metric [CoM | Atoms | Atoms_noH]
  End
  Type [None | Fast | Full]
End
```

NumAdditionalOrbitalEnergies

Type

Integer

Default value

1

Description

The number of additional orbital energies to write to the HDF5 file. A value of N means to write everything up to HOMO-N and LUMO+N.

NumExcitations

Type

Integer

Default value

1

Description

The number of excited states to calculate. By default the S_1 and T_1 states will be calculated. The calculation of excited states is currently only supported for systems with a closed-shell ground state.

TransferIntegrals**Type**

Block

Description

Configures the details of the calculation of electron and hole transfer integrals.

Exclude**Type**

Block

Description

Configures which dimers NOT to calculate transfer integrals for.

Cutoff**Type**

Float

Default value

4.0

Unit

Angstrom

GUI name

Exclude beyond

Description

Exclude dimers for which the distance is larger than this threshold. Acts as a quick pre-screening to reduce the number of dimers to calculate transfer integrals for.

Metric**Type**

Multiple Choice

Default value

Atoms

Options

[CoM, Atoms, Atoms_noH]

Description

The metric used to calculate the distance between two molecules.

- CoM: use the distance between the centers of mass of the two molecules.
- Atoms: Use the distance between the two closest atoms of two molecules.
- Atoms_noH: Use the distance between the closest non-hydrogen atoms of the two molecules.

Include**Type**

Block

Description

Configures which dimers transfer integrals are calculated for.

Cutoff**Type**

Float

Default value

4.0

Unit

Angstrom

GUI name

Include within

Description

Transfer integrals will be calculated for all molecule pairs within a cutoff distance from each other. This distance can be measured using different metrics, see the corresponding `Metric` keyword.

Metric**Type**

Multiple Choice

Default value

Atoms

Options

[CoM, Atoms, Atoms_noH]

Description

The metric used to calculate the distance between two molecules.

- CoM: use the distance between the centers of mass of the two molecules.
- Atoms: Use the distance between the two closest atoms of two molecules.
- Atoms_noH: Use the distance between the closest non-hydrogen atoms of the two molecules.

Type**Type**

Multiple Choice

Default value

Fast

Options

[None, Fast, Full]

Description

The method used for the calculation of the transfer integrals.

There are also a few options to tweak some aspects of the workflow. For typical OLED molecules, it is recommended to keep the default settings. When changing these options, verify your results against benchmark calculations for the single molecules.

```
Embedding
  Charges [DFTB | DFT]
  Cutoff float
  Metric [CoM | Atoms | Atoms_noH]
  Type [None | DRF]
End
```

(continues on next page)

(continued from previous page)

Relax [**None** | Neutral | All]
 OccupationSmearing [**None** | Ions | All]

Embedding**Type**

Block

Description

Configures details of how the environment is taken into account.

Charges**Type**

Multiple Choice

Default value

DFT

Options

[DFTB, DFT]

Description

Which atomic charges to use for the DRF embedding.

- DFTB: Use the self-consistent Mulliken charges from a quick DFTB calculation with the GFN1-xTB model.
- DFT: Use the MDC-D charges from a relatively quick DFT calculation using LDA and a DZP basis set.

Cutoff**Type**

Float

Default value

15.0

Unit

Angstrom

Description

The cutoff distance determining which molecules will be considered the environment of the central molecule. The maximum possible cutoff distance is half the length of the smallest lattice vector. The distance can be measured using different metrics, see the `Metric` keyword.

Metric**Type**

Multiple Choice

Default value

Atoms

Options

[CoM, Atoms, Atoms_noH]

Description

The metric used to calculate the distance between two molecules.

- CoM: use the distance between the centers of mass of the two molecules.

- Atoms: Use the distance between the two closest atoms of two molecules.
- Atoms_noH: Use the distance between the closest non-hydrogen atoms of the two molecules.

Type**Type**

Multiple Choice

Default value

DRF

Options

[None, DRF]

Description

The type of embedding used to simulate the molecular environment.

Relax**Type**

Multiple Choice

Default value

All

Options

[None, Neutral, All]

Description

Which geometries to relax prior to taking the energy differences for the calculation of ionization potential and electron affinity. The relaxation is done at the DFTB level using the GFN1-xTB model Hamiltonian with electrostatic embedding in a UFF environment.

- None: Use the geometries directly from the input.
- Neutral: Relax the uncharged molecule and use its optimized geometry for the neutral as well as the ionic systems. This gives (approximately) the vertical ionization potential and electron affinity.
- All: Individually relax the neutral systems and the ions before calculating the total energies. This gives (approximately) the adiabatic ionization potential and electron affinity.

OccupationSmearing**Type**

Multiple Choice

Default value

Ions

Options

[None, Ions, All]

Description

Determines for which systems the electron smearing feature in ADF will be used. If enabled, the molecular orbital occupations will be smeared out with a 300K Fermi-Dirac distribution. This makes SCF convergence easier, as the occupation of energetically close orbitals does not jump when their energetic order flips. See the ADF manual for details. It is recommended to keep this option enabled for the ionic systems, which are more likely to suffer from difficult SCF convergence.

2.3.4 Parallelization

The OLED properties workflow consists of independent chains of calculations for the individual molecules, and therefore scales very well when running on parallel machines. The OLED properties workflow is computationally very expensive. While you could theoretically run it on your local machine, you will need HPC facilities to do these calculations within any reasonable time frame.

Luckily, running the OLED workflow via a batch system has become much easier in the 2023 version, thanks to improvements in the underlying PLAMS library. The `.run` script for the `oled-properties` workflow script can now be submitted to the batch system like any other AMS job.

Note: Multi-node OLED properties jobs are currently only supported for SLURM-based clusters. For other batch systems, you will be limited to running the jobs on a single node.

The workflow script will internally take care of scheduling the individual jobs within the allocation that was made for it. All available CPUs are divided into groups, and each group of codes works together on one molecule before moving on to the next. By default 8 CPUs work together, so the `oled-properties` script submitted to a 128 core cluster allocation would internally do calculations for 16 molecules at the same time, each one using 8 cores. The size of the groups can be configured with the `CoresPerJob` keyword:

```
CoresPerJob integer
```

CoresPerJob

Type

Integer

Default value

8

Description

The number of CPU cores used for each job in the workflow. Combined with the total number of cores used (set by the `NSCM` environment variable or the `-n` command line argument), this indirectly determines the number of simultaneously running jobs. The default value should usually be a good choice. When changing this value, make sure you are using all allocated cores by setting a value that divides the total number of cores, as well as the number of cores on each node.

2.3.5 Restarting

Results are continuously written to the HDF5 file as they are calculated. If a job is interrupted, it will therefore leave an incomplete HDF5 file on disk, which can be used to restart the workflow by passing it to the `Restart` keyword:

```
Restart string
```

Restart

Type

String

Description

The HDF5 file from a previous calculation on the same morphology. Data already calculated on the restart file will just be copied over and not be recalculated.

Calculations are then only done for molecules (and dimers) for which not all results have been found on the restart HDF5 file.

This can be combined with the `--load` (or short: `-l`) flag, which uses the [PLAMS rerun prevention](#) and may prevent rerunning jobs for which the full result files are still available in the results directory of the failed job.

```
$AMSBIN/oled-properties -l "failed.results" << EOF

Restart failed.results/properties.hdf5

System
...
End

EOF
```

2.4 Material database

The OLED workflows come with a set of precalculated results for standard materials. These are just the results you would get if you ran both the deposition and properties workflow with all default settings on these materials. This data can be used as a reference to judge the performance of the workflows before running them on your own compounds.

Due to its size, the OLED material database is not included with AMS, but can easily be installed via [AMSPackages](#). Once the material database is installed, you can click either the folder icon next to it in [AMSPackages](#), or the **Open** button on the input panel for the deposition workflow in [AMSinput](#) to open it in your file browser. Data for each material is stored in a separate directory, e.g.:

```
beta-NPB/
├─ beta-NPB.in
├─ morphology.in
├─ properties.hdf5
└─ properties.pdf
```

Here `beta-NPB.in` contains the 3D structure of the deposited molecule. (It is basically the input to the deposition workflow, as all other settings were left at their default values.) Note that all molecules from the OLED material database can also be found in [AMSinput](#) through the search box at the top right. The result of the deposition is stored as the `morphology.in` file, which can be opened in [AMSinput](#) and directly be used as input for the properties workflow. The resulting properties are stored in the `properties.hdf5` file, see [HDF5 file](#) (page 29) above for details. A quick overview of the properties (e.g. the histograms) can be viewed by opening the `properties.pdf` file in a document viewer.

As of the 2023.1 release, the OLED material database contains data for the following pure materials, as well as a number of host-guest systems:

2.4.1 Pure materials

Compound	PubChem CID	calc. IP $\pm \sigma$ [eV] ^{Page 39, 1}	calc. EA $\pm \sigma$ [eV]
BCP	65149 (https://pubchem.ncbi.nlm.nih.gov/compound/65149)	6.63 \pm 0.20	1.37 \pm 0.18
CBP	11248716 (https://pubchem.ncbi.nlm.nih.gov/compound/11248716)	6.06 \pm 0.11	1.43 \pm 0.12
mCBP	23386664 (https://pubchem.ncbi.nlm.nih.gov/compound/23386664)	6.16 \pm 0.099	1.34 \pm 0.13
mCP	22020377 (https://pubchem.ncbi.nlm.nih.gov/compound/22020377)	6.24 \pm 0.089	1.14 \pm 0.090
mer-Alq3	16683111 (https://pubchem.ncbi.nlm.nih.gov/compound/16683111)	5.82 \pm 0.29	1.86 \pm 0.29
fac-Alq3	16683111 (https://pubchem.ncbi.nlm.nih.gov/compound/16683111)	5.88 \pm 0.33	1.85 \pm 0.32
fac-Ir(ppy)3	59117881 (https://pubchem.ncbi.nlm.nih.gov/compound/59117881)	5.78 \pm 0.25	1.54 \pm 0.25
mer-Ir(ppy)3	59117881 (https://pubchem.ncbi.nlm.nih.gov/compound/59117881)	5.48 \pm 0.17	1.56 \pm 0.18
α -MADN	53403806 (https://pubchem.ncbi.nlm.nih.gov/compound/53403806)	6.06 \pm 0.081	1.66 \pm 0.082
β -MADN	53403806 (https://pubchem.ncbi.nlm.nih.gov/compound/53403806) (isomer)	5.99 \pm 0.082	1.70 \pm 0.083
α -NPB	5069127 (https://pubchem.ncbi.nlm.nih.gov/compound/5069127)	5.43 \pm 0.090	1.55 \pm 0.083
α -NPB-2Me	5069127 (https://pubchem.ncbi.nlm.nih.gov/compound/5069127) (+ 2 methyl groups)	5.53 \pm 0.081	1.54 \pm 0.084
β -NPB	21881245 (https://pubchem.ncbi.nlm.nih.gov/compound/21881245)	5.42 \pm 0.073	1.51 \pm 0.072
β -NPB-2Me	21881245 (https://pubchem.ncbi.nlm.nih.gov/compound/21881245) (+ 2 methyl groups)	5.50 \pm 0.078	1.47 \pm 0.074
2-TNATA	16184079 (https://pubchem.ncbi.nlm.nih.gov/compound/16184079)	5.02 \pm 0.088	1.63 \pm 0.068
MT-DATA	11061735 (https://pubchem.ncbi.nlm.nih.gov/compound/11061735)	4.98 \pm 0.089	1.11 \pm 0.078
NBPhen	53403424 (https://pubchem.ncbi.nlm.nih.gov/compound/53403424)	6.07 \pm 0.17	1.84 \pm 0.16
Spiro-TAD	16134428 (https://pubchem.ncbi.nlm.nih.gov/compound/16134428)	5.23 \pm 0.060	1.38 \pm 0.081
T2T / TMBT ²	59336459 (https://pubchem.ncbi.nlm.nih.gov/compound/59336459)	6.63 \pm 0.079	1.80 \pm 0.076
T2T / TMBT	59336459 (https://pubchem.ncbi.nlm.nih.gov/compound/59336459)	6.63 \pm 0.082	1.80 \pm 0.074
TAPC	94071 (https://pubchem.ncbi.nlm.nih.gov/compound/94071)	5.25 \pm 0.058	0.784 \pm 0.069
TBRb	23576810 (https://pubchem.ncbi.nlm.nih.gov/compound/23576810)	5.33 \pm 0.060	1.98 \pm 0.062
TCTA	9962045 (https://pubchem.ncbi.nlm.nih.gov/compound/9962045)	5.66 \pm 0.098	1.49 \pm 0.090
TPBi	21932919 (https://pubchem.ncbi.nlm.nih.gov/compound/21932919)	6.41 \pm 0.19	1.58 \pm 0.20

¹ For the calculation of the mean and standard deviation of IP and EA, data points with a modified Z score > 16 were discarded as outliers. This only affects a few systems that had isolated outliers, e.g. due to SCF convergence problems.

² T2T and TMBT are two different names for the same compound. Both are included in the database.

2.4.2 Host-guest systems

Component	PubChem CID	calc. IP $\pm \sigma$ [eV]	calc. EA $\pm \sigma$ [eV]
95% CBP	11248716 (https://pubchem.ncbi.nlm.nih.gov/compound/11248716)	6.06 ± 0.12	1.44 ± 0.13
5% fac-Ir(ppy) ₃	59117881 (https://pubchem.ncbi.nlm.nih.gov/compound/59117881)	5.91 ± 0.12	1.59 ± 0.12

Component	PubChem CID	calc. IP $\pm \sigma$ [eV]	calc. EA $\pm \sigma$ [eV]
95% CBP	11248716 (https://pubchem.ncbi.nlm.nih.gov/compound/11248716)	6.05 ± 0.095	1.43 ± 0.11
5% PtOEP	636283 (https://pubchem.ncbi.nlm.nih.gov/compound/636283)	6.06 ± 0.095	1.86 ± 0.12

Component	PubChem CID	calc. IP $\pm \sigma$ [eV]	calc. EA $\pm \sigma$ [eV]
93.05% mCBP	23386664 (https://pubchem.ncbi.nlm.nih.gov/compound/23386664)	6.14 ± 0.11	1.34 ± 0.13
6.3% 4CzIPN-Me	102198498 (https://pubchem.ncbi.nlm.nih.gov/compound/102198498) (+ 8 methyl groups)	5.95 ± 0.13	2.46 ± 0.13
0.65% TBRb	23576810 (https://pubchem.ncbi.nlm.nih.gov/compound/23576810)	5.39 ± 0.098	2.04 ± 0.090

REACTIONS DISCOVERY

3.1 General

Reactions Discovery is a three-step workflow to find possible (side) reactions from constituent molecules:

1. *Reactive molecular dynamics* (page 48) based on the *NanoReactor* (page 50) or *Lattice Deformation* (page 52)
2. *Network Extraction* (page 58) using *ChemTraYzer2* (page 1) and geometry optimizations
3. *Product Ranking* (page 61)

To run Reactions Discovery in AMS2024, you need a license for *Advanced Workflows and Tools*.

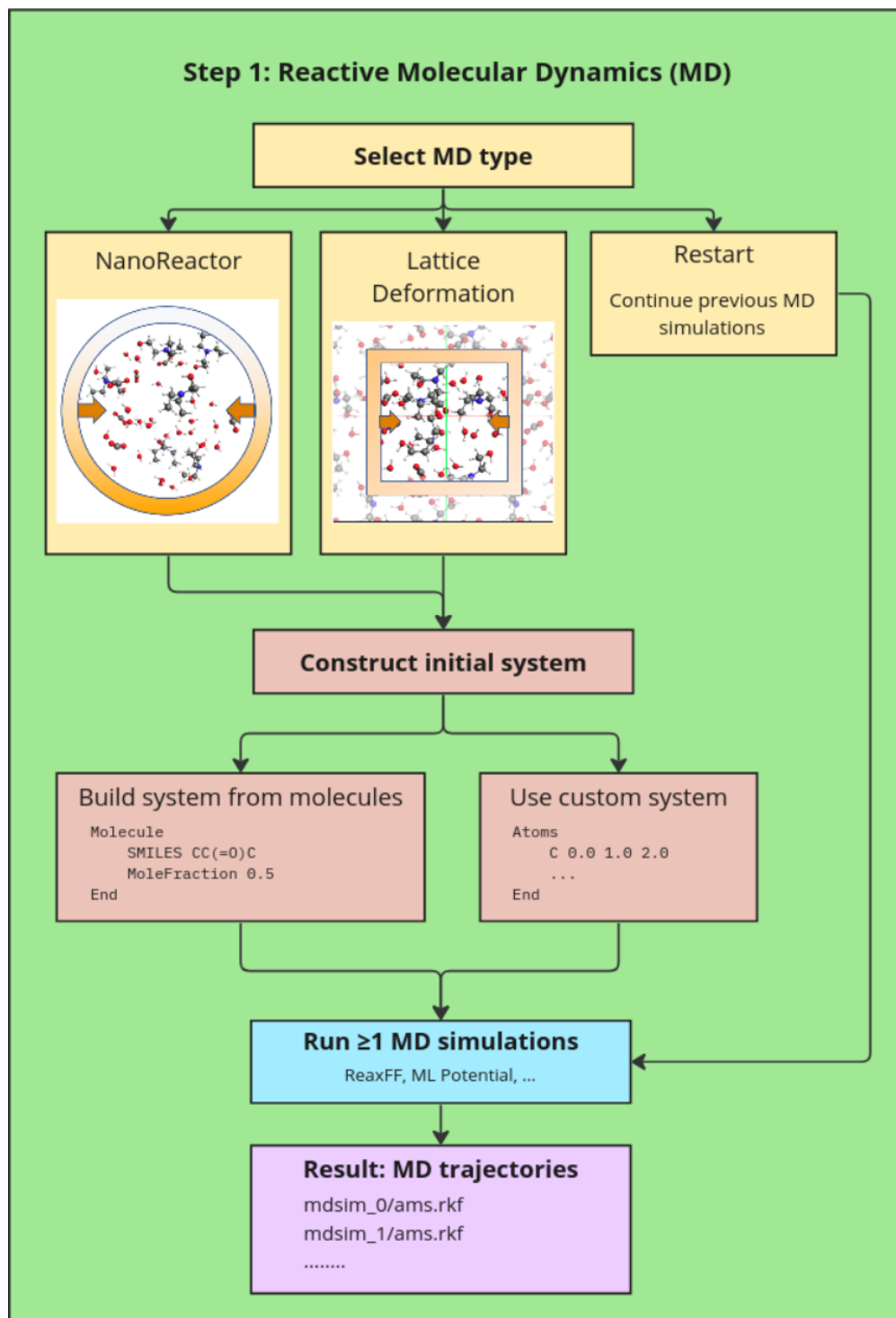
See also:

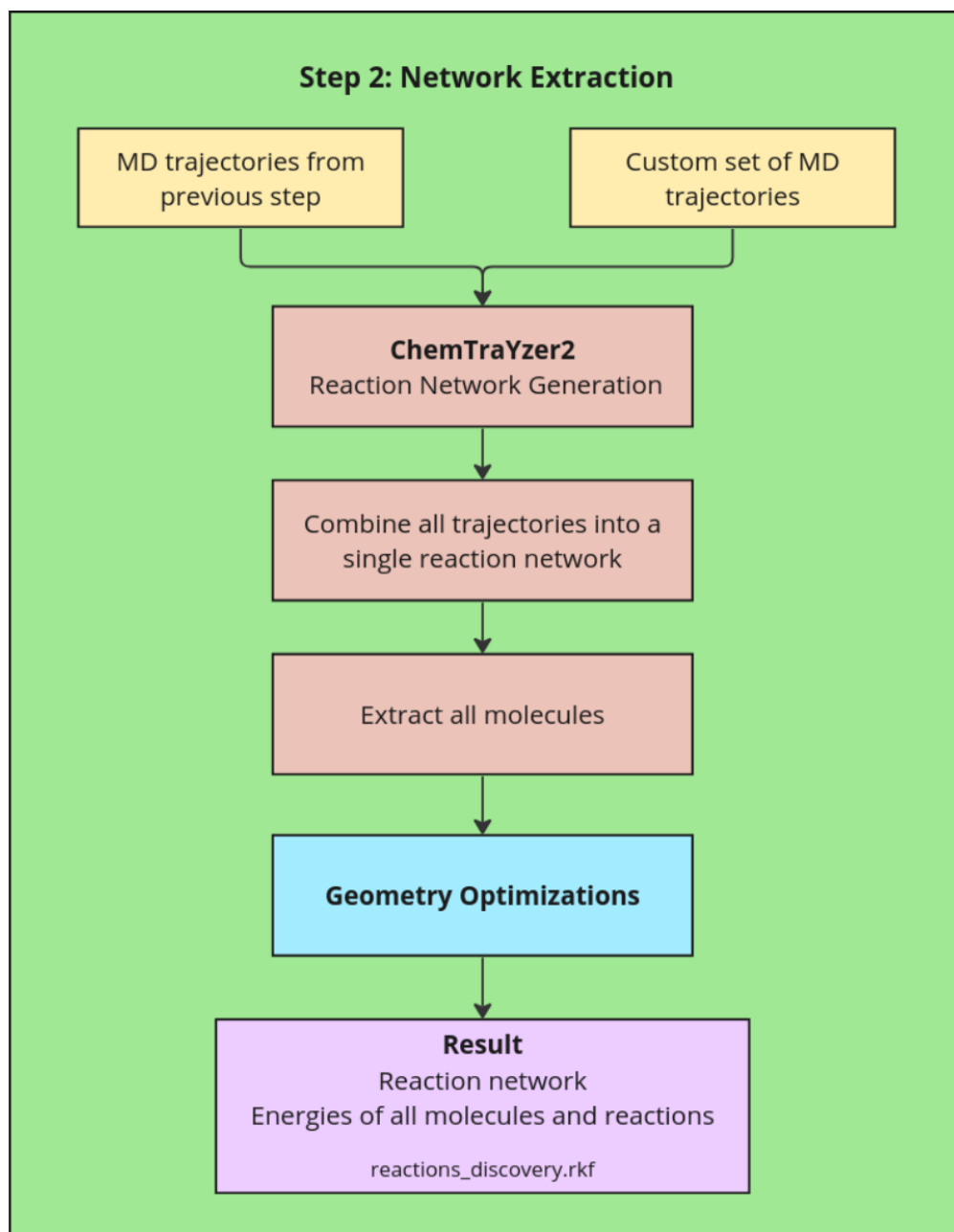
- *Quickstart guide with example input file* (page 45)
- Reactions Discovery graphical user interface tutorial
- *Reactions Discovery in Python (PLAMS)* (page 75)
- *Frequently Asked Questions* (page 79)

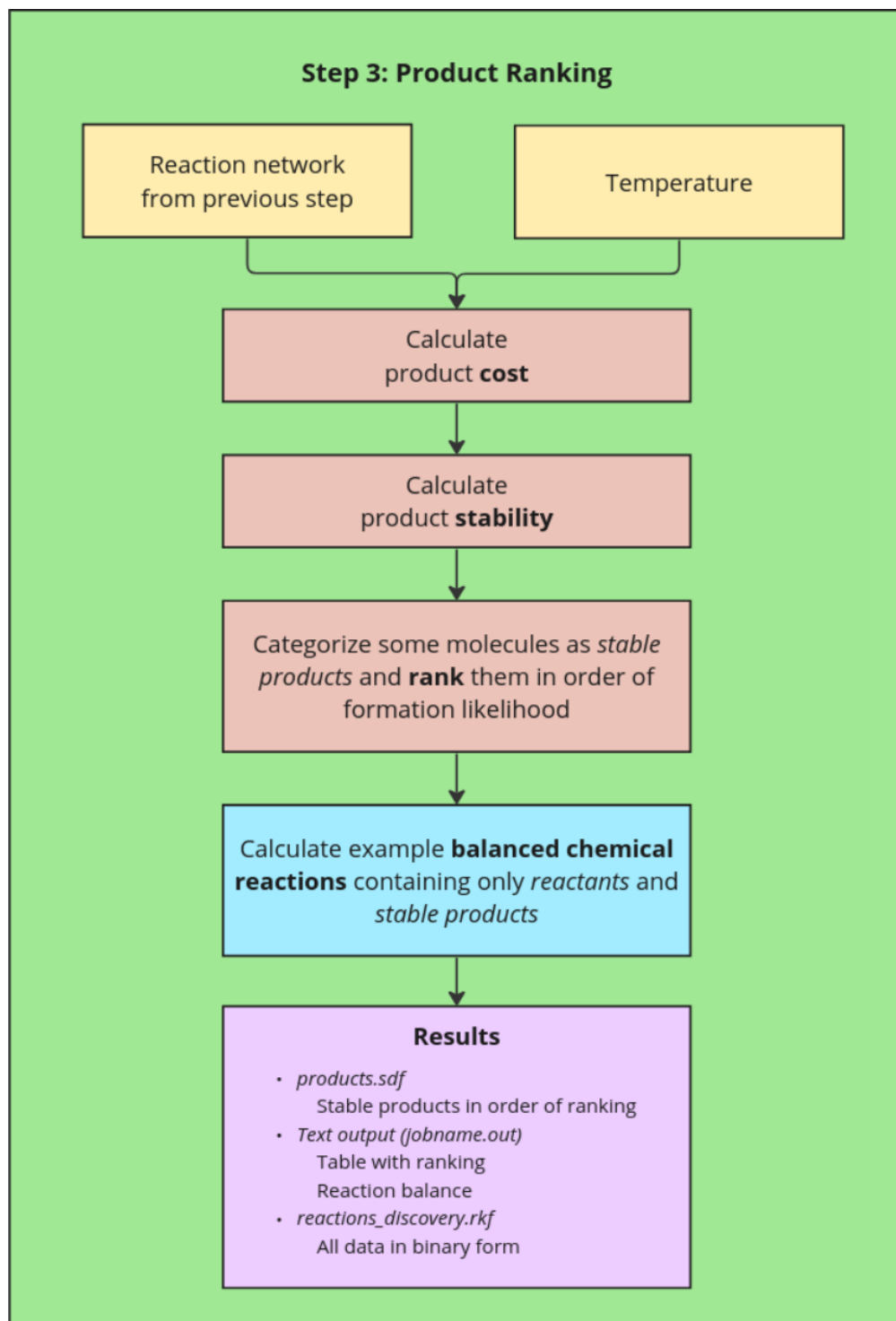
3.1.1 What's new in AMS2024?

The Reactions Discovery workflow is new in AMS2024.

3.2 Overview of workflow







3.3 Quickstart guide with example input file

A short example illustrating how to run the complete workflow, and how to restart from previous calculations.

```
#!/bin/sh

# Reactions discovery example for a mixutre of CH3CH2 and OH radicals
# The mixture should form ethanol CH3CH2OH and hydrogen peroxide HOOH
# and possibly more products.

rm -rf complete_nanoreactor.results

AMS_JOBNAME=complete_nanoreactor $AMSBIN/reactions_discovery << eor
MolecularDynamics
  Enabled Yes
  NumSimulations 4
  BuildSystem
    NumAtoms 50
    Density 0.9
    Molecule
      SMILES C[CH2] # a CH3CH2 radical
      MoleFraction 1
    End
    Molecule
      SMILES [OH] # an OH radical
      MoleFraction 3
    End
  End
  Type NanoReactor
  NanoReactor
    NumCycles 5
    Temperature 500
    MinVolumeFraction 0.6
  End
  BondOrders Method=Guess
End

NetworkExtraction Enabled=Yes UseCharges=Yes
ProductRanking Enabled=Yes

Engine ReaxFF
  ForceField CHON-2019.ff
  TaperBO Yes
EndEngine
eor

#Test if ethanol is found:
echo ETHANOL: `grep -c " CCO " complete_nanoreactor.results/reactions_discovery.log`
#Test if hydrogen peroxide is found:
echo HYDROGEN PEROXIDE: `grep -c " OO " complete_nanoreactor.results/reactions_
↪discovery.log`

# Use the MD trajectories from the previous job
rm -rf restart_extraction.results
AMS_JOBNAME=restart_extraction $AMSBIN/reactions_discovery << eor
MolecularDynamics Enabled=No
NetworkExtraction Enabled=Yes MDTrajectories=complete_nanoreactor.results
```

(continues on next page)

(continued from previous page)

```

    ProductRanking Enabled=Yes
    Engine ReaxFF
        ForceField CHON-2019.ff
        TaperBO Yes
    EndEngine
eor

#Test if ethanol is found:
echo ETHANOL: `grep -c " CCO " restart_extraction.results/reactions_discovery.log`
#Test if hydrogen peroxide is found:
echo HYDROGEN PEROXIDE: `grep -c " OO " restart_extraction.results/reactions_
↪discovery.log`

# Use the previous NetworkExtraction results to restart the ranking
rm -rf restart_ranking.results
AMS_JOBNAME=restart_ranking $AMSBIN/reactions_discovery << eor
    MolecularDynamics Enabled=No
    NetworkExtraction Enabled=No
    ProductRanking Enabled=Yes ReactionNetwork=restart_extraction.results
eor

#Test if ethanol is found:
echo ETHANOL: `grep -c " CCO " restart_ranking.results/reactions_discovery.log`
#Test if hydrogen peroxide is found:
echo HYDROGEN PEROXIDE: `grep -c " OO " restart_ranking.results/reactions_discovery.
↪log`

# Demonstrate LatticeDeformation
rm -rf complete_lattice_deformation.results

AMS_JOBNAME=complete_lattice_deformation $AMSBIN/reactions_discovery << eor
    MolecularDynamics
        Enabled Yes
        NumSimulations 2
        BuildSystem
            NumAtoms 50
            Density 0.4
            Molecule
                SMILES C[CH2] # a CH3CH2 radical
                MoleFraction 1
            End
            Molecule
                SMILES [OH] # an OH radical
                MoleFraction 3
            End
        End
        Type LatticeDeformation
        LatticeDeformation
            NumCycles 3
            Temperature 500
            MinVolumeFraction 0.2
            Period 200
        End
        BondOrders Method=Guess
    End

    NetworkExtraction Enabled=Yes UseCharges=No

```

(continues on next page)

(continued from previous page)

```

ProductRanking Enabled=Yes

Engine ReaxFF
  ForceField CHON-2019.ff
  TaperBO Yes
EndEngine
eor

#Test if ethanol is found:
echo ETHANOL: `grep -c " CCO " complete_lattice_deformation.results/reactions_
↪discovery.log`
#Test if hydrogen peroxide is found:
echo HYDROGEN PEROXIDE: `grep -c " OO " complete_lattice_deformation.results/
↪reactions_discovery.log`

```

3.4 Command to execute, parallelization

Normally you would run:

```
"$AMSBIN/reactions_discovery" < input_file.in > output_file.out
```

This will **run in parallel** and use as many cores as are available on the system or allocation.

You can also explicitly set the number of cores with the `-n` flag. For example, to run the `reactions_discovery` program **in serial**, you would run:

```
"$AMSBIN/reactions_discovery" -n 1 < input_file.in > output_file.out
```

Or you could also run *Reactions Discovery in Python (PLAMS)* (page 75).

3.5 Engine settings

For *Molecular dynamics* (page 48) and *Network Extraction* (page 58) specify the AMS **engine**:

Tip: Reactions Discovery requires a computationally efficient engine. We recommend to use

- ReaxFF,
- DFTB, or
- ML Potential

Engine

Type

Block

Description

The input for the computational engine for the MolecularDynamics and NetworkExtraction tasks. The header of the block determines the type of the engine.

3.6 Molecular dynamics

To enable the molecular dynamics, set `MolecularDynamics%Enabled` to Yes. If it is set to No, you can still load previously run MD simulations and continue with the [Network Extraction](#) (page 58) and [Product Ranking](#) (page 61) parts.

The reactions discovery tool implements special non-equilibrium MD methods to promote chemical reactions, namely the [Nanoreactor](#) (page 50) and [Lattice deformation](#) (page 52). The type is controlled by `MolecularDynamics%Type`.

The number of simulations to run is controlled by `MolecularDynamics%NumSimulations`. To get enough statistics, it is recommended to run several MD simulations. The results will be gathered into a single summary in the [Network Extraction](#) (page 58) and [Product Ranking](#) (page 61) parts.

To replace all hydrogens with deuterium set `MolecularDynamics%UseDeuterium` Yes. If you prefer, you can use a bond guessing algorithm instead of the engine-calculated bonds by setting `MolecularDynamics%BondOrders%Method` to Guess.

See also:

[Bond orders & Molecule detection](#)

```
MolecularDynamics
  Enabled Yes/No
  Type [NanoReactor | LatticeDeformation | Restart]
  NumSimulations integer
  UseDeuterium Yes/No
  TimeStep float
  BondOrders
    Method [Guess | EngineWithGuessFallback]
  End
End
```

MolecularDynamics

Type

Block

Description

Settings for reactive molecular dynamics.

Enabled

Type

Bool

Default value

Yes

GUI name

Reactive Molecular Dynamics

Description

Whether to run molecular dynamics.

Type

Type

Multiple Choice

Default value

NanoReactor

Options

[NanoReactor, LatticeDeformation, Restart]

Description

The type of molecular dynamics.

NumSimulations**Type**

Integer

Default value

4

GUI name

Number of simulations

Description

Total number of MD simulations to run.

UseDeuterium**Type**

Bool

Default value

No

Description

If true, all hydrogen atoms will be replaced by deuterium during the MD. This helps to slow down the motion of the hydrogen atoms. This options does **not** affect the density you should insert in BuildSystem%Density. However, it **does** affect the density on the resulting MD trajectory file.

TimeStep**Type**

Float

Default value

0.5

Unit

fs

Description

Molecular dynamics time step.

BondOrders**Type**

Block

Description

Details regarding the calculation/guessing of bond orders during Molecular Dynamics. The bond changes during the MD are later analyzed in the Network Extraction step.

Method**Type**

Multiple Choice

Default value

EngineWithGuessFallback

Options

[Guess, EngineWithGuessFallback]

Description

How to compute the bond orders.

'Guess': Use a bond guessing algorithm based on the system's geometry. This is the same algorithm that is used by the Graphical User Interface to guess bonds.

'EngineWithGuessFallback': let the engine compute the bond orders but if the engine did not produce any bond orders, use the bond guessing algorithm as a fallback option.

3.6.1 Nanoreactor

To enable the nanoreactor mode, set `MolecularDynamics%Type` to `NanoReactor`.

The nanoreactor is run **without** periodic boundary conditions. Details can be found in the [Nanoreactor AMS Driver documentation](#).

The following 4 phases are looped throughout the simulation:

phase	volume fraction	time (fs)	thermostat (K)	T	force (Ha/bohr^2)	constant
pre_compression	1.05	25	250		0.0005	
compression	MinVolumeFraction	25	250		0.01	
post_compression	1.05	100	250		0.0005	
diffusion	1.05	Diffusion-Time	Temperature		0.0004	

The radius of the nanoreactor for each individual phase is calculated as:

- $r_{\text{nanoreactor}} \text{ (per phase)} = \text{volume fraction}^{1/3} * \text{InitialRadius}$

`InitialRadius` is automatically determined, but can also be explicitly specified in the input.

If you want to customize any numbers other than `MinVolumeFraction`, `DiffusionTime`, and `Temperature`, or if you want to change the number of phases, you can

- [set up your own NanoReactor simulations](#) outside the Reactions Discovery tool, and then
- specify the path to the results using [NetworkExtraction](#) (page 59).

All nanoreactor-specific input options:

```
MolecularDynamics
  NanoReactor
    DiffusionTime float
    InitialRadius float
    MinVolumeFraction float
    NumCycles integer
    Temperature float
  End
End
```

MolecularDynamics**Type**

Block

Description

Settings for reactive molecular dynamics.

NanoReactor**Type**

Block

Description

Option for the reactive molecular dynamics.

DiffusionTime**Type**

Float

Default value

250.0

Unit

fs

Description

The length of the diffusion phase in femtoseconds.

InitialRadius**Type**

Float

Unit

angstrom

Description

The radius of the initial (spherical) system. If `BuildSystem` is used, the value is ignored (then the value is automatically determined). If `BuildSystem` is not used, then a guess for the `InitialRadius` will be made if it is not specified.

MinVolumeFraction**Type**

Float

Default value

0.6

GUI name

Minimum volume fraction

Description

The minimum (compressed) volume of the system, as a fraction of the initial (maximum) system volume.

NumCycles**Type**

Integer

Default value

10

GUI name

Number of cycles

Description

How many compression-expansion cycles to perform.

Temperature**Type**

Float

Default value

500.0

Unit

K

Description

Temperature during the diffusion phase. The temperature during the compression phase will be much higher as a result of the inward acceleration.

3.6.2 Lattice deformation

To enable the lattice deformation mode, set `MolecularDynamics%Type` to `LatticeDeformation`.

Lattice deformation requires that the system is 3D-periodic. For more details, see the [Lattice deformations \(volume regimes\)](#) AMS Driver documentation.

The volume oscillates for `NumCycles` cycles with a period of `Period fs` between

- V_{initial} , and
- $V_{\text{compressed}} = V_{\text{initial}} * \text{MinVolumeFraction}$.

All lattice deformation input options:

```
MolecularDynamics
  LatticeDeformation
    MinVolumeFraction float
    NumCycles integer
    Period float
    Temperature float
  End
End
```

MolecularDynamics**Type**

Block

Description

Settings for reactive molecular dynamics.

LatticeDeformation**Type**

Block

Description

Option for the reactive molecular dynamics.

MinVolumeFraction**Type**

Float

Default value

0.3

GUI name

Minimum volume fraction

Description

The minimum (compressed) volume of the system, as a fraction of the initial (maximum) system volume.

NumCycles**Type**

Integer

Default value

10

GUI name

Number of cycles

Description

How many compression-expansion cycles to perform.

Period**Type**

Float

Default value

100.0

Unit

fs

Description

The period with which the lattice will oscillate in femtoseconds.

Temperature**Type**

Float

Default value

500.0

Unit

K

Description

Thermostat temperature during the MD simulation.

3.6.3 Build the initial system

Note: This section describes a convenient way to build the system directly from the input.

You can also build the initial system in any way you like. Then set `MolecularDynamics%BuildSystem%Enabled` to `False` and give the coordinates/lattice or coordinate file in the `System` block.

`BuildSystem` will build a

- **sphere centered at the origin** if `MolecularDynamics%Type` is `NanoReactor` - the radius of the sphere will automatically be passed on to the `NanoReactor` settings.
- **cubic box** if `MolecularDynamics%Type` is `LatticeDeformation`.

The **initial system** is specified through a series of `Molecule` blocks. Each `Molecule` block contains either a SMILES string or a reference to a `System` block, as well as the mole fraction of that particular molecule.

The **initial density** and the **system size** are specified with the `Density` and `NumAtoms` keywords:

You can choose to run a short **equilibration simulation** by setting `Equilibration`. This can be useful if the packmol-generated structure is unrealistic.

```
MolecularDynamics
  BuildSystem
    Density float
    Enabled Yes/No
    Equilibration Yes/No
    Molecule
      MoleFraction float
      SMILES string
      SystemID string
    End
    NumAtoms integer
  End
End
```

MolecularDynamics

Type

Block

Description

Settings for reactive molecular dynamics.

BuildSystem

Type

Block

Description

Build the initial system for molecular dynamics with packmol. If `MolecularDynamics%Type` is set to `Restart`, then `BuildSystem` is ignored.

Density

Type

Float

Default value

1.0

Unitg/cm³**Description**

The initial density of the system. This should be the lowest density (corresponding to the largest volume) that you want the system to have. The standard atomic masses are used when building the system.

Enabled**Type**

Bool

Default value

Yes

GUI name

Build System

Description

If True, build the initial system using packmol. If False, a System block must be provided with the initial system.

Equilibration**Type**

Bool

Default value

No

Description

Whether to run a short 250 fs equilibration simulation on the packmol-built structure.

Molecule**Type**

Block

Recurring

True

Description

A molecule to put in the MD simulation.

MoleFraction**Type**

Float

Default value

1.0

GUI name

Relative mole fraction

Description

Mole fraction of the molecule (the mole fractions of the various molecules will be normalized, so only the relative MoleFraction values matter)

SMILES**Type**

String

Description

The SMILES string of the molecules.

SystemID**Type**

String

Description

The ID of the corresponding System (i.e. the header of the corresponding System block).

NumAtoms**Type**

Integer

Default value

200

GUI name

Number of atoms

Description

Approximate total number of atoms in each MD simulation.

3.6.4 Fixed MD settings

The reactions discovery tool automatically sets the following for all MD simulations:

- [Short-range repulsive potential](#),
- Frames are saved every 10 fs. To save disk space, velocities are not saved.
- MD checkpoints are saved every 1000 MD steps (allowing to `Restart`)
- A Berendsen thermostat with a very short time constant and `BerendsenApply = Local`.

For a complete view of all the MD input, open the `mdsim_0/mdsim_0.in` file created by the Reactions Discovery tool.

See also:

[Molecular dynamics](#) in the AMS Driver documentation

3.6.5 Molecular dynamics restart

Set `MolecularDynamics Type=Restart` in order to

- continue interrupted MD simulations (for example if they exceeded the walltime limit), or
- add extra steps to already finished MD simulations

```
MolecularDynamics
  Restart
    Directory string
    NSteps integer
  End
End
```

MolecularDynamics

Type

Block

Description

Settings for reactive molecular dynamics.

Restart**Type**

Block

Description

Settings for restarting MD simulations.

Directory**Type**

String

GUI name

Restart directory

Description

Directory containing a previous Reactions Discovery calculation with MD simulations that were not finished. Note: This directory will be scanned recursively for ams.rkf files!

NSteps**Type**

Integer

GUI name

Number of steps

Description

Number of MD steps. If left empty, the number of MD steps from the original MD trajectory will be used. Note that you need to manually increase this number if you want to continue finished simulations.

Example: restart MD simulations after they exceeded walltime limit

Input:

```
MolecularDynamics
  Enabled Yes
  Type Restart
  Restart
    Directory /some/path
  End
End
Engine ...
...
EndEngine
```

If /some/path contains the following files:

```
/some/path/dir1/ams.rkf
/some/path/dir2/subdir/ams.rkf
```

where those ams.rkf files come from MD simulations, then if

- `ams.rkf` contains an **unfinished simulation** (e.g. because the job exceeded the walltime limit), it will be restarted from where it finished, but if
- `ams.rkf` contains a **finished simulation**, then the trajectory will simply be copied

Note:

- The rest of the `MolecularDynamics` reactions discovery input block is **ignored**, meaning that it doesn't matter if you specify `LatticeDeformation` or `NanoReactor` or their respective settings. The MD settings from the `ams.rkf` file will be used.
 - The `Engine` block from the reactions discovery input block is **used**.
 - If you restart from a previous reactions discovery results directory (which you likely do), then it may contain several directories `mdsim_0`, `mdsim_1`, etc., or you may have called them something else if you set up the MD simulations manually. The **numbering may change** in the restart run, so that the new `mdsim_1` actually continues the trajectory from the original `mdsim_0`.
-

Example: Continue MD simulations for more steps

```
MolecularDynamics
  Enabled Yes
  Type Restart
  Restart
    Directory /some/path
    NSteps 20000
  End
End
Engine ...
...
EndEngine
```

The above will continue all MD simulations until 20000 steps. If the original simulation ran for 5000 steps, then the new simulation will continue for another 15000 steps.

3.7 Network Extraction

In this second step of the workflow a reaction network is extracted, and the individual molecules are optimized.

3.7.1 Initial Network from ChemTrayzer2

To extract the reaction network, *ChemTraYzer2* (page 1) is used. By default, only the trajectories from simulations that terminated normally are used for network extraction. The reaction network is fully stored in the `ams.rkf` file, as well as in the file `reaction_network.gml`, which can be directly read into the `networkx` (<https://www.networkx.org>) python module. Reaction network information is also printed to output, and the details of the latter can be set in the input block `NetworkExtraction%Print`. In the output, the molecules are listed in order of their net flux; The difference between the number of instances at the end of the simulation and the number of instances at the start of the simulations. At this first approximation, products with the highest positive net flux are considered to be the most stable.

3.7.2 Geometry Optimization

For all unique molecules, an attempt is made to find the optimized geometry for that molecule. For each molecule the coordinates are extracted either from the molecular dynamics runs (reported with source file, frame and atom ids) or constructed from the SMILES string using `from_smiles`. Finally, a geometry optimization is performed and each molecule is assigned the total energy corresponding to the optimized geometry. If the geometry optimization fails, or if our bond guessing algorithm predicts a different connectivity after optimization, the energy of the unoptimized coordinates is used instead. If desired, the geometry optimizations can be performed with a different engine, e.g. using an implicit solvent, if this better approximates the target experimental reaction conditions. To do this, first perform a run with molecular dynamics only and then perform a restart with molecular dynamics disabled and the engine of choice defined in the input.

3.7.3 Molecular Charge Assignment

In order to correctly perform geometry optimizations, the charge of each molecule needs to be determined beforehand. The charges are obtained by collecting the atomic charges from the output of the molecular dynamics simulations, averaging the resulting molecular charges over the relevant frames, and rounding to integer values. Every molecule will be connected to at most 2 reactions; The reaction that creates it, and the reaction that destroys it. A check is performed, to see if all the reaction charges are balanced. If this is not the case, then the corresponding molecular charges are adjusted to ensure charge balance through-out. The charge adjustments are restricted in that the change Δq per molecule may not exceed 1.0e. If unbalanced reactions still remain after the charge adjustments, then an ionization reaction is added, connecting two versions of the molecule, each with a different charge.

3.7.4 Manual specification of MD trajectories

You can run the network extraction on any MD trajectories, even if they were not calculated by the Reactions Discovery tool.

If

- `MolecularDynamics%Enabled` is `True`, then those MD trajectories will be used for the network extraction,
- `MolecularDynamics%Enabled` is `False`, then you can specify a directory with `ams.rkf` files to analyze with `NetworkExtraction%MDTrajectories`

Example:

```
MolecularDynamics Enabled=No
NetworkExtraction Enabled=Yes MDTrajectories=/some/path
```

Unfinished simulations are only used if `NetworkExtraction%ExtractFromFailedMDJobs` is enabled.

```
NetworkExtraction
  Enabled Yes/No
  ExtractFromFailedMDJobs Yes/No
  MDTrajectories string
  Print
    FilterFluctuations Yes/No
    MaxReactionOrder integer
    MinReactionsThreshold integer
    SkipRareReactions Yes/No
  End
  UseCharges Yes/No
End
```

NetworkExtraction

Type

Block

Description

Options for extracting the reactive network from MD trajectories

Enabled**Type**

Bool

Default value

Yes

GUI name

NetworkExtraction

Description

Whether to perform network extraction.

ExtractFromFailedMDJobs**Type**

Bool

Default value

No

GUI name

Extract from failed MD jobs

Description

Whether to extract from failed/crashed MD jobs (by default, only successful jobs are used)

MDTrajectories**Type**

String

GUI name

MD Trajectories

Description

If MolecularDynamics%Enabled is False, this directory will be recursively scanned for ams.rkf files containing MD trajectories. All found trajectories will be used for the analysis. It should typically be a Reactions Discovery results directory containing finished MD simulations.

Print**Type**

Block

Description

Printing details

FilterFluctuations**Type**

Bool

Default value

Yes

Description

If true, do not print molecules that are only part of recrossing reactions.

MaxReactionOrder**Type**

Integer

Default value

4

Description

If the reaction order is larger than this value, the reaction will not be printed.

MinReactionsThreshold**Type**

Integer

Default value

3

Description

If a molecule is involved in fewer reaction than this value, skip printing the results corresponding to it. To print all molecules, set this value to 0.

SkipRareReactions**Type**

Bool

Default value

Yes

Description

If true, reduce the output by filtering out rare reactions. If false, always print all reactions.

UseCharges**Type**

Bool

Default value

Yes

Description

Use engine-calculated charges if they exist on the MD trajectory files for the NetworkExtraction

3.8 Product Ranking

Ranking the molecules based on the net flux over the molecular dynamics simulations is only as reliable as the simulations themselves. All reactive simulations use some kind of scheme to induce reactivity, and this means that reaction conditions like temperature and pressure will most likely be higher than the conditions in the target system. Letting a reaction under our extreme simulation conditions continue too long will most likely result in the degradation of viable products, combined with the formation of final products that would never be formed at the milder experimental conditions. To correct for the extreme conditions of the molecular dynamics simulations we assign stabilities to the molecules based on the reaction network.

Both a value for kinetic stability (cost) and thermodynamic stability are assigned, and ultimately the molecules are ordered based in the first place on cost, and in the second place on (thermodynamic) stability. The ranked stable products are saved in the files `products.sdf` and `products.rkf`.

3.8.1 Reaction Energies

As the first step in this procedure, we assign reaction energies to all reaction nodes in the network, based on the energies of the optimized molecules. These reaction energy values have the advantage that they are independent of the simulation conditions.

3.8.2 Product Cost

We perform a network search to assign a relative cost value to each molecule in the network. The cost c of the reactants is set to one. The cost c_p of any product/intermediate molecule p is then assigned in a breadth first search through the network.

$$c_p = \sum_r c_r + (1 + e^{E_R/k_b T})$$

Here R is the reaction that has molecule p as product, and results in the lowest possible cost c_p . The molecules r are the reactants involved in reaction R . The value E_R is the reaction energy of reaction R . T is the ranking temperature, which is set to 300K by default. Selecting a higher temperature brings the contributions of different reactions closer together, resulting in relatively lower costs for molecules that are formed via reactions with high reaction energies.

3.8.3 Product Stability

In a similar manner, a thermodynamic relative product stability is assigned, with the stability of the initial reactants set to zero.

$$s_p = \sum_r s_r + E_R$$

Again, R is the reaction that has p as product and results in the lowest value of the cost c_p .

3.8.4 Reaction Balance

Separate from the reaction network procedures, a set of stable products is extracted based on two different metrics. First, molecules with connectivities deviating vastly from the valences of the corresponding elements are discarded. Secondly, the molecular charges determined in the NetworkExtraction run are compared to the 'formal' molecular charges based on bond orders, and if the two values differ, the molecule is considered a radical, and discarded as a stable product.

For all stable products, an estimated balanced overall formation reaction is then determined by balancing the reaction of the initial reactants to this product. If it is not possible to balance the reaction, one of the other stable products is included as a co-product.

3.8.5 Example: ProductRanking from finished NetworkExtraction

Input:

```
MolecularDynamics Enabled=No
NetworkExtraction Enabled=No
ProductRanking Enabled=Yes ReactionNetwork=/some/path
```

ProductRanking%ReactionNetwork **must** be a previous Reactions Discovery result directory that has at least finished the network extraction.

```

ProductRanking
  BalanceFromNetwork Yes/No
  DiscardIons Yes/No
  Enabled Yes/No
  MaxBalancedReactions integer
  ReactionNetwork string
  Temperature float
  WritePaths Yes/No
End

```

ProductRanking**Type**

Block

Description

Options for ranking of the intermediates by stability

BalanceFromNetwork**Type**

Bool

Default value

No

Description

Use the network to determine the balanced reaction from the initial reactants to each stable product. This is not the default. By default, a balanced equation is determined directly by using the other stable products as possible side products.

DiscardIons**Type**

Bool

Default value

Yes

Description

Remove all ions from the final product list

Enabled**Type**

Bool

Default value

Yes

GUI name

ProductRanking

Description

Whether to perform ranking of the reaction network.

MaxBalancedReactions**Type**

Integer

Default value

1000

Description

The maximum number of stable products used to find a balanced reaction equation for each one.

ReactionNetwork**Type**

String

Description

Directory containing a previous Reactions Discovery calculation with 'NetworkExtraction%Enabled Yes'

Temperature**Type**

Float

Default value

298.0

Description

Temperature used to compute reaction rates from reaction energies of reactions in the reaction network.

WritePaths**Type**

Bool

Default value

Yes

Description

Write full paths to the reaction network for each molecule.

3.9 Output

3.9.1 Reactants, products, unstable

The results directory contains a file `reactions_discovery.rkf`. This file contains all the reaction network information. It can be read by the `reactions_discovery` workflow, which can use it to perform/redo the ranking process. Additionally, `AMSmovie` can be used to view the molecules in the reaction network.

All molecules in the reaction network are categorized either as “product”, “unstable”, or “reactant”.

- **Reactants** are the molecules that exist at the beginning of the MD runs.
- **Products** are the suggested stable side products in the reaction network.
- **Unstable** are the molecules that are not considered stable.

There are three reasons a molecule is considered unstable.

1. The number of bonds to the atoms deviates from the atom valence by more than a maximum amount. This maximum is hardcoded per element.
2. The assigned molecular charge deviates from the formal molecular charge of the molecule. This is a strong indication that the molecule is a radical.
3. By default all ions are considered unstable, but this can be changed in the input.

The suggested side products can be found in the file `products.sdf` in the results directory. This file contains all network intermediates that are considered stable.

The text output contains a list of ranked products, with an estimate of cost of formation (labeled 'Barrier') and reaction energy. This list of products is followed by a list containing for each stable product the shortest path from the initial reactants. The cost of formation is a soft maximum of all reaction energies in the shortest path to the product.

3.9.2 KF output files

Note: This section documents the sections and variables in the `reactions_discovery.rkf` file.

General

Section content: General information about the amsbatch calculation.

General%account

Type

string

Description

Name of the account from the license

General%ElapsedTime

Type

float

Description

Elapsed time of the AMS workflow.

Unit

second

General%engine messages

Type

string

Description

Message from the engine. In case the engine fails to solves, this may contains extra information on why.

General%file-ident

Type

string

Description

The file type identifier, e.g. RKF, RUNKF, TAPE21...

General%jobid

Type

int

Description

Unique identifier for the job.

General%ProcessTime

Type

float

Description

Time the AMS workflow spent in Python.

Unit

second

General%program**Type**

string

Description

The name of the program/engine that generated this kf file.

General%release**Type**

string

Description

The version of the program that generated this kf file (including svn revision number and date).

General%termination status**Type**

string

Description

The termination status. Possible values: 'NORMAL TERMINATION', 'NORMAL TERMINATION with warnings', 'NORMAL TERMINATION with errors', 'ERROR', 'IN PROGRESS'.

General%title**Type**

string

Description

Title of the calculation.

General%uid**Type**

string

Description

SCM User ID

General%user input**Type**

string

Description

The text input of the AMS workflow.

General%version**Type**

int

Description

Version number?

MolecularDynamicsResults**Section content:** Generic results.**MolecularDynamicsResults%NumSimulations****Type**

int

Description

Number of molecular dynamics simulations that were performed.

ChemicalSystem(#)**Section content:** Molecules**ChemicalSystem (#) %nAtoms****Type**

int

Description

The number of atoms in the system

ChemicalSystem (#) %nAtomsTypes**Type**

int

Description

The number different of atoms types

ChemicalSystem (#) %AtomicNumbers**Type**

int_array

Description

Atomic number 'Z' of the atoms in the system

Shape

[nAtoms]

ChemicalSystem (#) %AtomMasses**Type**

float_array

Description

Masses of the atoms

Unit

a.u.

Values range

[0, 'infinity']

Shape

[nAtoms]

ChemicalSystem (#) %AtomSymbols**Type**

string

Description

The atom's symbols (e.g. 'C' for carbon)

Shape

[nAtoms]

ChemicalSystem (#) %EngineAtomicInfo**Type**

string_fixed_length

Description

Atom-wise info possibly used by the engine.

ChemicalSystem (#) %Coords**Type**

float_array

Description

Coordinates of the nuclei (x,y,z)

Unit

bohr

Shape

[3, nAtoms]

ChemicalSystem (#) %bondOrders**Type**

float_array

Description

The bond orders for the bonds in the system. The indices of the two atoms participating in the bond are defined in the arrays 'fromAtoms' and 'toAtoms'. e.g. bondOrders[1]=2, fromAtoms[1]=4 and toAtoms[1]=7 means that there is a double bond between atom number 4 and atom number 7

ChemicalSystem (#) %toAtoms**Type**

int_array

Description

Index of the second atom in a bond. See the bondOrders array

ChemicalSystem (#) %fromAtoms**Type**

int_array

Description

Index of the first atom in a bond. See the bondOrders array

Categories

Section content: Different categories of molecules.

Categories%NumProducts**Type**

int

Description

Number of products.

Categories%NumReactants**Type**

int

Description

Number of reactants.

Categories%NumUnstable**Type**

int

Description

Number of unstable systems.

Categories%Products**Type**

int_array

Description

Indices of (RD)History entries that are products.

Shape

[NumProducts]

Categories%Reactants**Type**

int_array

Description

Indices of (RD)History entries that are reactants.

Shape

[NumReactants]

Categories%Unstable**Type**

int_array

Description

Indices of (RD)History entries that are unstable.

Shape

[NumUnstable]

Reaction(#)**Section content:** A reaction.**Reaction (#) %Composition****Type**

string

Description

The description of the reaction (for example, A => B + C) where the molecules are described by their empirical formula.

Reaction (#) %Count**Type**

int

Description

Total number of this this reaction was observed.

Reaction (#) %Hash**Type**

string

Description

Unique identifier for this reaction.

Reaction (#) %ProductHashes**Type**

lchar_string_array

Description

Hashes (i.e. the unique identifiers) of the product molecules.

Reaction (#) %ProductIndices**Type**

int_array

Description

Indices of the product molecules in the RDHistory section.

Reaction (#) %ReactantHashes**Type**

lchar_string_array

Description

Hashes (i.e. the unique identifiers) of the reactant molecules.

Reaction (#) %ReactantIndices**Type**

int_array

Description

Indices of the reactant molecules in the RDHistory section.

Reaction (#) %ReactionEnergy**Type**

float

Description

The reaction energy for this reaction.

Unit

hartree

Reaction (#) %SMILESDescription**Type**

string

Description

The description of the reaction (for example, A => B + C) where the molecules are described by their SMILES strings.

History

Section content: History of the system during the AMS calculation. What is stored here depends on the task of

the AMS calculation. For example, for a GeometryOptimization this will contain the intermediate steps of the GO, while for a MoleculeDynamics calculation it will contain the MD frames.

History%Bonds

Type

subsection

Description

?

History%Coords (#)

Type

float_array

Description

Coordinates of the systems of a given entry.

Shape

[3, :]

History%Energy (#)

Type

float

Description

Energy of the system of a given entry.

Unit

hartree

History%LatticeVectors (#)

Type

float_array

Description

The lattice vectors of a given entry.

Unit

bohr

Shape

[3, :]

History%nEntries

Type

int

Description

Number of history entries.

History%nLatticeVectors (#)

Type

int

Description

The number of lattice vectors (i.e. the number of periodic boundary conditions) of a given entry.

History%Step (#)

Type
int

Description
The step number in a Molecular Dynamics calculation.

History%SystemVersion (#)

Type
int

Description
Index of the versioned-chemical system of a given frame.

RDHistory

Section content: History of a Molecular dynamics simulation.

RDHistory%Balance (#)

Type
string

Description
The overall balanced reaction for this product

RDHistory%BalancedReaction (#)

Type
int_array

Description
Index of the estimated most efficient balanced reaction resulting in this molecule (estimated from the pool of all reactants and all stable products).

RDHistory%blockSize

Type
int

Description
Explain the block-system... ?

RDHistory%Cost (#)

Type
float_array

Description
The sum of the exponentials ($\exp(E/RT)+1$) of the reaction energy of each reaction involved in formation of this product. Taking the logarithm of this effectively results in the highest 'barrier' along the path.

RDHistory%CreatedBy (#)

Type
string

Description
This command was used to obtain the coordinates for stability analysis for this molecule

RDHistory%currentEntryOpen

Type
bool

Description

?

RDHistory%Depth (#)**Type**

int_array

Description

The depth of the molecule in the network (number of elementary reactions separating it from the reactants)

RDHistory%FinalCount (#)**Type**

int_array

Description

Number of molecules of this kind at the end of the simulation.

RDHistory%GuessedCharge (#)**Type**

float_array

Description

The formal charge of the molecule as estimated by PLAMS

RDHistory%Hash (#)**Type**

string

Description

Unique identifier of the molecule.

RDHistory%Id (#)**Type**

string

Description

The indexed formula of this molecule.

RDHistory%InitialCount (#)**Type**

int_array

Description

Number of molecules of this kind at the beginning of the simulation.

RDHistory%ItemName (#)**Type**

string

Description

?

RDHistory%MolecularFormula (#)**Type**

string

Description

Molecular formula.

RDHistory%Name (#)**Type**

string

Description

SMILES string

RDHistory%nBlocks**Type**

int

Description

Explain the block-system... ?

RDHistory%nEntries**Type**

int

Description

Number of MD history entries.

RDHistory%ParentReaction (#)**Type**

int_array

Description

The index of the parent reaction in the shortest path from reactants to this compound

RDHistory%PathEnergy (#)**Type**

float_array

Description

The overall energy balance of the full path to this product

RDHistory%PathTree (#)**Type**

string

Description

The full path through the network to this product, as a string.

RDHistory%PotentialEnergy (#)**Type**

float_array

Description

The potential energy, i.e. the energy as computed by the engine. This is a 'blocked' property. See the 'blockSize' and 'nBlocks' variables for more details.

Unit

hartree

RDHistory%Radical (#)

Type

bool_array

Description

Is this molecule a radical or not.

RDHistory%ReactionsAsProduct (#)**Type**

int_array

Description

Indices of the reactions in which this molecule was part of the products.

RDHistory%ReactionsAsReactant (#)**Type**

int_array

Description

Indices of the reactions in which this molecule was part of the reactants.

RDHistory%Representative (#)**Type**

bool_array

Description

Signifies if the coordinates are representative of the molecule (as defined by the connectivity)

RDHistory%SMILES (#)**Type**

string

Description

SMILES string

3.10 Reactions Discovery in Python (PLAMS)

An [example](#) of how to use Reactions Discovery with Python (PLAMS) can be found in the examples section of the PLAMS documentation.

The `scm.reactions_discovery.plams_job` Python module contains the below classes.

class ReactionsDiscoveryResults (*job*)

Results class for ReactionsDiscoveryJob

get_errormsg ()

Returns the error message of this calculation if any were raised.

Returns

String containing the error message.

Return type

str

get_md_jobs ()

Returns the AMSJobs used during the calculation.

Returns

List of AMSJobs used during the calculation.

Return type

List[AMSJob]

rkfpath()

Returns path to reactions_discovery.rkf

Returns

Path to reactions_discovery.rkf

Return type

str

get_network_rd()

Returns the reaction network represented by Reactions Discovery CombinedMol and CombinedReaction classes.

Raises**KFError** – If the KF file associated with this result does not contain the required information.**Returns**

Graph representing the reaction network, a dictionary of categories and lists of CombinedMol, a dictionary of categories and CombinedReaction and a list of categories.

Return type

Tuple[DiGraph, Dict[str, List[CombinedMol]], Dict[int, CombinedReaction], List[str]]

get_network()**Returns the reaction network represented by a DiGraph and a dictionary of lists of PLAMS molecules.**

Each key in the dictionary is a category.

Returns

graph of the reaction network, dictionary of categories and lists of Molecules, and a list of categories.

Return type

Tuple[DiGraph, Dict[str, List[Molecule]], List[str]]

get_num_md_simulations()

Returns the number of MD simulations used during the Molecular Dynamics stage.

Raises**KFError** – If the KF file associated with this result does not contain the right information.**Returns**

The number of MD simulations used during the Molecular Dynamics stage.

Return type

int

```
class ReactionsDiscoveryJob (name='reactions_discovery_job', driver=None, settings=None,  
                             molecule=None, **kwargs)
```

PLAMS Job class for running Reactions Discovery.

This class inherits from the PLAMS SingleJob class. For usage, see the SingleJob documentation.

If you supply a Settings object to the constructor, it will be converted to a PISA (Python Input System for AMS) object.

Attributes:

- `input`: an alias for `self.settings.input`
- `builder`: an alias for `self.settings.input.MolecularDynamics.BuildSystem`

__init__ (*name*='reactions_discovery_job', *driver*=None, *settings*=None, *molecule*=None, ***kwargs*)

Initialize the ReactionsDiscoveryJob.

name

[str] The name of the job

driver

[scm.input_classes.drivers.ReactionsDiscovery] PISA object describing the input to the ReactionsDiscovery program

settings: scm.plams.Settings

All settings for the job. Input settings in the PLAMS settings format under `settings.input` are automatically converted to the PISA format. You can specify `settings.runscript.nproc` to set the total number of cores to run on.

molecule: scm.plams.Molecule or Dict[str, scm.plams.Molecule]

Two possibilities:

- `molecule` is of type `Molecule` - it should then be the *complete* system as a PLAMS `Molecule`. Cannot be combined with the `driver.input.MolecularDynamics.BuildSystem` or `settings.input.ams.MolecularDynamics.BuildSystem`. It will be written to the main System block in the input.
- `molecule` is a dictionary with string keys and `Molecule` values - the keys should then be given in the `driver.input.MolecularDynamics.BuildSystem.Molecule[i].SystemID` input option. The molecules will then be used to build the system before the MD.

classmethod from_rkf (*path*)

Initialize a job from a reactions_discovery.rkf file.

Parameters

path (*str*) – Path to a reactions_discovery.rkf file

Returns

A new ReactionsDiscoveryJob instance based on the information found in path.

Return type

[*ReactionsDiscoveryJob*](#) (page 76)

classmethod from_input (*text_input*)

Initialize a job from text input.

Parameters

text_input (*str*) – A multiline text input

Returns

A ReactionsDiscoveryJob

Return type

[*ReactionsDiscoveryJob*](#) (page 76)

static _extract_mol_from_pisa (*pisa*)

Remove a molecule from a System block in the ReactionsDiscovery PISA object and return it as molecule(s)

get_errormsg ()

Returns the contents of the jobname.err file if it exists. If the file does not exist an empty string is returned.

Returns

The error message

Return type

str

get_runscript()

Generates the runscript. Use `self.settings.runscript.preamble_lines = ['line1', 'line2']` or similarly for `self.settings.runscript.postamble_lines` to set custom settings.

`self.settings.runscript.nproc` controls the total number of cores to run on.

check()

Returns True if “NORMAL TERMINATION” is given in the General section of `reactions_discovery.rkf`, AND all molecular dynamics jobs also have finished successfully.

ok()

Synonym for `check()`

get_md_jobs()

Returns: List of `AMSJob`

property input

PISA format input

classmethod load_external(path, finalize=False)

Load a previous ReactionsDiscovery job from disk.

Parameters

- **path** (*Union[str, Path]*) – A reactions discovery results folder.
- **finalize** (*bool, optional*) – See `SingleJob`, defaults to False

Raises

FileError – When the path does not exist.

Returns

An initialized `ReactionsDiscoveryJob`

Return type

ReactionsDiscoveryJob (page 76)

get_input()

Obtain the input string used to run the Reactions Discovery workflow script.

Returns

An input string.

Return type

str

3.11 Frequently Asked Questions

3.11.1 There are no reactions

- Increase the temperature
- **NanoReactor:** Increase `DiffusionTime`, set `MinVolumeFraction` to a smaller value, set `InitialRadius` to a smaller value
- **LatticeDeformation:** Set `MinVolumeFraction` to a smaller value, decrease `Period`
- Set `MolecularDynamics UseDeuterium=No`
- Ensure that you use a reactive potential (e.g. ReaxFF, DFTB, MLPotential)

Note: The best value for `MinVolumeFraction` depends on the initial density (`MolecularDynamicsBuildSystemDensity`). If you decrease the initial density, you may need to also decrease the `MinVolumeFraction`.

Tip: Run the simulation with the fast UFF force field to get a feeling for if the initial settings are reasonable. With UFF no reactions will happen but you can still see how the density fluctuates in the MD simulation.

3.11.2 There are too many reactions

- Decrease the temperature
- **NanoReactor:** Decrease `DiffusionTime`, set `MinVolumeFraction` to a larger value, set `InitialRadius` to a larger value
- **LatticeDeformation:** Set `MinVolumeFraction` to a larger value, increase `Period`
- Set `MolecularDynamics UseDeuterium=Yes`

Note: The best value for `MinVolumeFraction` depends on the initial density (`MolecularDynamicsBuildSystemDensity`). If you decrease the initial density, you may need to also decrease the `MinVolumeFraction`.

3.11.3 The MD simulations are too slow

- Decrease the number of atoms
- Decrease the number of NanoReactor or LatticeDeformation cycles
- Increase the MD time step

3.11.4 How should I set the density and compression factor?

- **Nanoreactor:** The density should be approximately the normal liquid density of your system, with a compression factor of about 0.5-0.7
- **Lattice deformation:** The density should be about half the normal liquid density of your system, with a compression factor of about 0.15-0.30

3.11.5 The simulation explodes

- Follow the steps for *There are too many reactions*.
- Decrease the MD time step

3.11.6 How do I use computing resources efficiently?

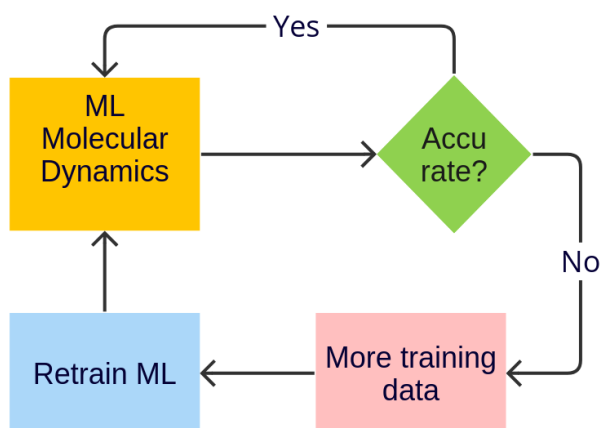
The steps are parallelized as follows:

- MolecularDynamics runs as many jobs in parallel as possible, respecting the allocation (NSCM)
- NetworkExtraction runs NumSimulations ChemTraYzer2 jobs sequentially in serial; then NSCM geometry optimizations and single points are run in parallel.
- ProductRanking runs in serial

Thus, most of the computational steps except ProductRanking are run efficiently in parallel. You may thus choose to set `ProductRanking Enabled=False` if you have a large node allocated, and then restart from the previous results using `ProductRanking Enabled=True` on a smaller allocation.

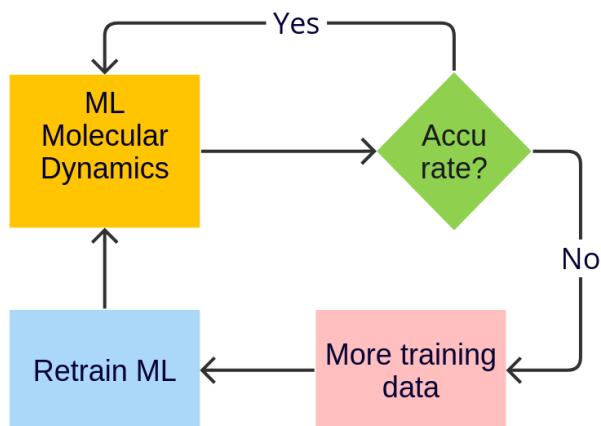
SIMPLE ACTIVE LEARNING

Simple Active Learning (SAL) is a workflow for on-the-fly training (active learning) of machine learning (ML) potentials during molecular dynamics (MD).



4.1 General

Simple Active Learning (SAL) is a workflow for on-the-fly training (active learning) of machine learning (ML) potentials during molecular dynamics (MD). It is “simple” because it only applies to a single MD simulation. In the AMSinput GUI it can be found under *MD Active Learning*.



The workflow

- **Trains** an initial ML potential
- **Runs** the MD simulation
- **Pauses** the MD simulation and launches new reference (typically DFT) calculations at set intervals or if the ML potential is not accurate enough
- **Retrains** the ML potential to the new reference data
- **Rewinds** the MD simulation to the last point where it was known to be accurate
- **Continues** the MD simulation, pauses, retrains, rewinds, continues, ...

Optionally, the workflow can be restarted from a previous workflow (skipping the initial training).

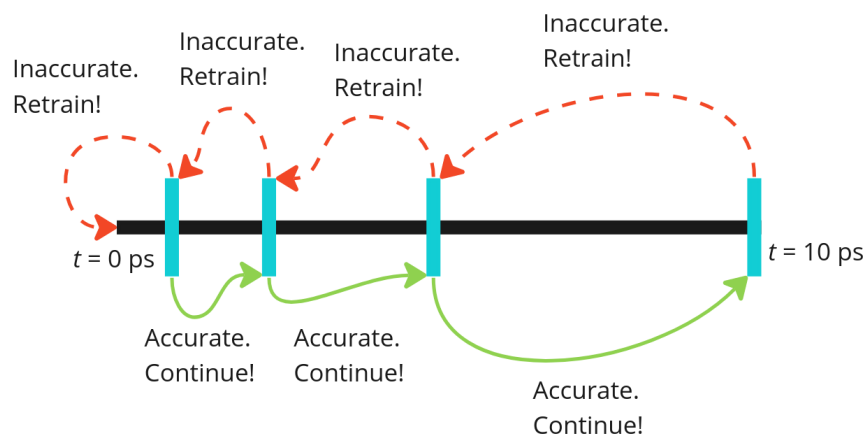


Fig. 4.1: Example: You run 10 ps MD, dividing the simulation into 4 segments (“active learning steps”) indicated by the blue bars. At the blue bars, the accuracy of the model is checked and a decision is made whether to continue the MD simulation or to retrain the model and rewind to a previous point.

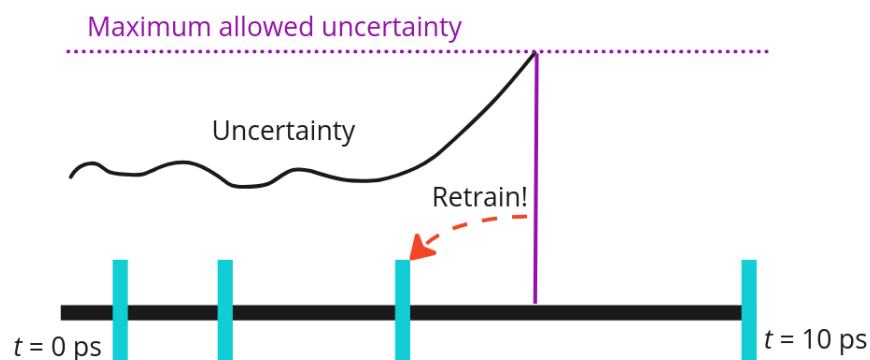


Fig. 4.2: Example: You train a committee model that estimates the model’s uncertainty. As soon as the uncertainty increases above a given threshold, the MD simulation stops. The model is retrained and the simulation rewinds to the previous active learning step.

There are five main pieces of input:

- **Input system.** This is the initial system for the MD simulation. The input is exactly the same as for any other AMS simulation.

- **Molecular dynamics settings.** It can be equilibrium or non-equilibrium MD. The settings/input are exactly the same as for any other AMS simulation.
- **Reference engine settings.** This can be any engine, but would typically be one of the DFT engines ADF, BAND, or Quantum ESPRESSO. The settings/input are exactly the same as for any other AMS simulation. This engine determines the level of theory to which the ML potential is trained.
- **ParAMS ML training settings.** You can train any ML potential that is supported by ParAMS, for example, M3GNet. The settings/input are exactly the same as for running standalone ParAMS with Task MachineLearning.
- **Active learning settings.** These settings determine, for example, how frequently to launch new reference calculation, and how to judge if the ML potential is accurate enough.

The three main pieces of output are:

- The requested **MD trajectory**, that can be analyzed for results
- The **trained ML model parameters**, that can potentially also be used for other (production) simulations
- All **training and validation data**, containing the results from the reference calculations

4.1.1 Licensing

To run Simple Active Learning, you need licenses for

- Advanced workflows and tools (includes the workflow and ParAMS),
- Classical force fields and machine learning potentials (to run the ML potential simulations)
- The reference engine (e.g., ADF, BAND, or Quantum ESPRESSO)

4.1.2 What's new in Simple Active Learning?

AMS2025

- Added a Python SimpleActiveLearningJob.restart_from() function for easier scripting/workflows
- Added the ActiveLearning%JobPrefix input option, to easier see the origin of training data in large training sets from multiple combined jobs
- Added the RNGSeed input option to control the random number seed for molecular dynamics (does not affect ML training).
- Added the FromScratchTraining input block

AMS2024

The Simple Active Learning workflow is new in AMS2024.

4.2 Quickstart guide

A short example illustrating how to run the workflow directly from the command-line.

See also:

- *Python Examples* (page 109)
- *Input* (page 85)
- *Tutorial* using the graphical user interface

Important: You must first [install M3GNet](#) before running this example!

```
# Example to train an M3GNet potential to reproduce the potential energy surface
# of the UFF force field for a small organic molecule

# In real applications, use a different reference engine!

# Before running this example you must install m3gnet:
# "$AMSBIN/amspackages" install m3gnet

# For details or off-line installation, see the package manager documentation.

"$AMSBIN/simple_active_learning" <<EOF
  ActiveLearning
    Steps
      Geometric
        NumSteps 5
        Start 10
      End
      Type Geometric
    End
  End

  MachineLearning
    Backend M3GNet
    CommitteeSize 1
    M3GNet
      Model UniversalPotential
    End
    MaxEpochs 200
  End

  MolecularDynamics
    InitialVelocities
      Temperature 300.0
      Type Random
    End
    NSteps 10000
    Thermostat
      Tau 200.0
      Temperature 300.0
      Type NHC
    End
    TimeStep 0.5
```

(continues on next page)

(continued from previous page)

```

Trajectory
  SamplingFreq 100
End
End

Task MolecularDynamics

Engine ForceField
  Type UFF
EndEngine

System
  Atoms
      O      1.5185424677      1.2528427606      -0.3382346351
      C      1.0167107700      0.2231953999      0.1069866215
      C      -0.3341736669     -0.1931701775     -0.3199821682
      O      -1.3067693409      0.4447398660      0.4319048572
      H      1.6797677292     -0.4284512471      0.6676972986
      H      -0.4715138766      0.0855120883     -1.3968084763
      H      -0.4208687404     -1.2969328351     -0.3134514232
      H      -1.6816953421     -0.0877358551      1.1618879254
  End
End
EOF

```

4.3 Input

Simple Active Learning reads all options from an input file, described here. You can also set up this input file in *Python* (page 109).

Table 4.1: AMS-like Molecular Dynamics input options

Block	Required?	Comment
System/LoadSystem	Yes	identical to AMS Driver System/LoadSystem
Task	No	must be set to MolecularDynamics
MolecularDynamics	Yes	identical to AMS Driver MolecularDynamics
Constraints	No	identical to AMS Driver Constraints
RNGSeed	No	Random number seed(s) for MD simulations
Engine	Yes	reference engine settings, identical to normal AMS calculations
MachineLearning	Yes	identical to ParAMS MachineLearning settings
ParallelLevels	No	identical to ParAMS ParallelLevels settings
ActiveLearning	Yes	described on this page

The engine settings for the MD simulations are determined from the MachineLearning input. For example, if you train an M3GNet model, this means that you will automatically run M3GNet also during the MD simulation.

This section only describes the `ActiveLearning` input block, which controls

- How to generate/load initial reference data
- When to perform reference calculations
- Criteria for deciding whether to retrain the model or continue the MD simulation

- How much output to save
- Whether to retrain the model and/or rerun the simulation after the active learning loop has finished

See also:

- [Quickstart guide](#) (page 84)
- [Python Examples](#) (page 109)

4.3.1 Overview

```

ActiveLearning
  AtEnd
    RerunSimulation Yes/No
    RetrainModel Yes/No
  End
  FromScratchTraining
    Enabled Yes/No
    EpochMultiplier float
    Probability float
  End
  InitialReferenceData
    Generate
      M3GNetShortMD
      Enabled Yes/No
    End
    ReferenceMD
      Enabled Yes/No
    End
  End
  Load
    Directory string
    FromPreviousModel Yes/No
  End
End
JobPrefix string
MaxAttemptsPerStep integer
MaxReferenceCalculationsPerAttempt integer
ReasonableSimulationCriteria
  Distance
    Enabled Yes/No
    MinValue float
  End
  EnergyUncertainty
    Enabled Yes/No
    MaxValue float
    Normalization float
  End
  GradientsUncertainty
    Enabled Yes/No
    MaxValue float
  End
  Temperature
    Enabled Yes/No
    MaxValue float
  End
End

```

(continues on next page)

(continued from previous page)

```

Save
  ReferenceCalculations [None | All]
  ReferenceData [Latest | All]
  TrainingDirectories [Latest | All]
  Trajectories [Latest | All]
End
Steps
  Geometric
    NumSteps integer
    Start integer
  End
  Linear
    Start integer
    StepSize integer
  End
  List integer_list
  Type [Geometric | List | Linear]
End
SuccessCriteria
  Energy
    Enabled Yes/No
    Normalization float
    Relative float
    Total float
  End
  Forces
    Enabled Yes/No
    MaxDeviationForZeroForce float
    MaxMAE float
    MinR2 float
  End
End
End

```

4.3.2 Initial reference data

Before the main active learning loop starts, there must be some training data.

The initial training data can be loaded from disk and/or automatically generated. If no data is loaded and no generation option is explicitly enabled, then the `ReferenceMD` option described below will be automatically enabled to ensure that there is some data for the initial model training.

```

ActiveLearning
  InitialReferenceData
    Generate
      M3GNetShortMD
        Enabled Yes/No
      End
      ReferenceMD
        Enabled Yes/No
      End
    End
  End
  Load
    Directory string
    FromPreviousModel Yes/No
  End
End

```

(continues on next page)

(continued from previous page)

```

End
End
End

```

Generate initial reference data

The **M3GNetShortMD** option (**recommended**) follows a short pre-programmed MD simulation using the universal M3GNet-UP-2022 potential. This gives some structural variation in the initial training data. It generates structures as follows:

- 300 MD steps with timestep 0.5 fs, temperature = 500 K
- If the system is 3d-periodic then linearly scale the density from 92% to 108% of the original density
- 5 frames are recalculated with the reference engine and added to the training/validation sets

The **ReferenceMD** option (**default** if nothing else is specified)

- Runs 3 MD steps (saving every frame) using the exact MolecularDynamics settings specified in the input
- Adds those frames to the training/validation sets

Load initial reference data

If you already have some reference data, for example if you have

- previously run Simple Active Learning, or
- manually created the data by importing into ParAMS and saving,

then you can load it in Simple Active Learning, so that the old data is combined with the new data generated during the workflow.

If you specify the `ActiveLearning%InitialReferenceData%Load%Directory` option, then the initial reference data will be taken from that directory.

Otherwise, if you're loading a previously trained model using `MachineLearning%LoadModel`, and if you enable `ActiveLearning%InitialReferenceData%Load%FromPreviousModel`, then both the parameters and the training and validation data will be loaded.

Initial reference data input

ActiveLearning

Type

Block

Description

Settings for Active Learning

InitialReferenceData

Type

Block

Description

Options for loading reference data.

Generate

Type

Block

Description

How to generate initial reference data from the initial structure. Can also be combined with the `Load` block.

The purpose of these options is to get some initial reference structures/data around the current structure that can be used for Step 1 of the active learning loop.

The `ReferenceMD` option will be automatically enabled if no data is otherwise loaded or generated.

M3GNetShortMD**Type**

Block

Description

Structure sampler using M3GNet-UP-2022

Enabled**Type**

Bool

Default value

No

GUI name

M3GNet-UP short MD:

Description

Run 300 steps with M3GNet-UP-2022 at T=600 K. If the system is 3D-periodic the density will be scanned around the initial value. Extract 5 frames and run reference calculations on those.

ReferenceMD**Type**

Block

Description

Run `NSteps` of the MD simulation using the reference engine.

Enabled**Type**

Bool

Default value

No

GUI name

Reference MD:

Description

Run 3 steps with the reference engine and add those 3 frames to the training and validation sets. If no other reference data is loaded or generated, this option will automatically be enabled.

Load

Type

Block

Description

How to load initial reference data from other sources. Can also be combined with the Generate block

Directory**Type**

String

Default value**Description**

Directory containing initial reference data. It can be

- * a ParAMS input directory or a `stepX_attemptY_reference_data` directory containing the files `job_collection.yaml`, `training_set.yaml`, and `validation_set.yaml`.

- * a ParAMS results directory.

If a directory is specified here it will be used instead of the data from a previously loaded model.

FromPreviousModel**Type**

Bool

Default value

Yes

Description

If `MachineLearning%LoadModel` is set, reuse reference data from that ParAMS run.

If `MachineLearning%LoadModel` is not set, or if `Directory` is specified, then this input option is ignored.

4.3.3 When to run reference calculations (step sequence type)

In the Simple Active Learning workflow, the MD simulation is divided into a sequence of **active learning (AL) steps**.

```
ActiveLearning
  Steps
    Geometric
      NumSteps integer
      Start integer
    End
    Linear
      Start integer
      StepSize integer
    End
    List integer_list
    Type [Geometric | List | Linear]
  End
  MaxAttemptsPerStep integer
  MaxReferenceCalculationsPerAttempt integer
  JobPrefix string
End
```


Step Type Geometric (default)

Example:

- You set up the MD simulation with $N_{\text{MD}} = 10000$ steps with a time step of 0.5 fs, giving a total simulation length of $10000 \times 0.5 = 5000$ fs = 5 ps.
- You set up the ActiveLearning with `Steps%Type = Geometric` with `Start` set to 10 (MD frames) and `NumSteps` set to 5, and `MaxAttemptsPerStep` set to 8

For example using the following input:

```
MolecularDynamics
  NSteps 10000
  TimeStep 0.5
  # ... other MD options
End

ActiveLearning
  Steps
    Type Geometric # default
    Geometric
      Start 10 # default
      NumSteps 5
    End
  End
  MaxAttemptsPerStep 8
  MaxReferenceCalculationsPerAttempt 4
  # ... other ActiveLearning options
End
```

This will divide the 10000 MD steps into 5 AL steps, where the first AL step contains 10 MD steps, and each subsequent AL step contains progressively more MD steps (following a Geometric progression):

```
The ACTIVE LEARNING loop will contain 5 steps, using the following scheme:
Active Learning Step 1:      10 MD Steps (cumulative:      10)
Active Learning Step 2:      46 MD Steps (cumulative:     56)
Active Learning Step 3:      260 MD Steps (cumulative:    316)
Active Learning Step 4:     1462 MD Steps (cumulative:   1778)
Active Learning Step 5:     8222 MD Steps (cumulative:  10000)
Total number of MD Steps: 10000
Max attempts per active learning step: 8
```

The progression is geometric because $56/10 \approx 316/56 \approx 1778/316 \approx 10000/1778 \approx 5.6$.

The above scheme means that the **active learning loop will be executed as follows**:

1. `step1_attempt1_simulation`: Run 10 MD steps using the initially trained model
2. `step1_attempt1_ref_calc1`: Run reference calculation on final frame
3. Evaluate the *Success criteria* (page 96):
 - If no success: **run up to 3 more** reference calculations, **retrain the model**, and **loop back to the beginning of the step 1**: rerun AL step 1 (the first 10 MD steps) as `step1_attempt2_simulation` using the new parameters, run reference calculation on final frame, evaluate the success criteria, ...
 - If success *or* if the number of attempts > 8: continue to AL step 2
1. `step2_attempt1_simulation`: Run 46 MD steps starting from the final frame of AL step 1, for a total (cumulative) length of 56 MD steps

2. `step2_attempt1_ref_calc1`: Run reference calculation on final frame
3. Evaluate the *Success criteria* (page 96):
 - If no success: **run up to 3 more** reference calculations, **retrain the model**, and **loop back to the beginning of the step** 4: rerun AL step 2 (the 46 MD steps) as `step2_attempt2_simulation` using the new parameters, run reference calculation on final frame, evaluate the success criteria, ...
 - If success *or* if the number of attempts > 8: continue to AL step 3
1. `step3_attempt1_simulation`: Run 260 MD steps starting from the final frame of AL step 2, for a total (cumulative) length of 315 MD steps
2. Etcetera....

Step Type Linear

The steps can also follow a linear progression.

This is especially useful if you run non-equilibrium MD where you linearly apply some restraint, for example if you use a [ReactionBoost RMSDRestraint](#) following the TargetCoordinate, or apply a linear [lattice deformation](#).

Instead of providing the number of steps, you provide the start step and the step size:

```
MolecularDynamics
  NSteps 10000
  # other MD options...
End

ActiveLearning
  Steps
    Type Linear
    Linear
      Start 100
      StepSize 2000
    End
  End
End
```

Active Learning Step	1:	100 MD Steps (cumulative:	100)
Active Learning Step	2:	2000 MD Steps (cumulative:	2100)
Active Learning Step	3:	2000 MD Steps (cumulative:	4100)
Active Learning Step	4:	2000 MD Steps (cumulative:	6100)
Active Learning Step	5:	2000 MD Steps (cumulative:	8100)
Active Learning Step	6:	1900 MD Steps (cumulative:	10000)

Step Type List

You can also list the (cumulative) number of MD steps per active learning step explicitly. The final MD step is always considered to be the end of an active learning step and does not need to be specified.

```
MolecularDynamics
  NSteps 10000
  # other MD options...
End

ActiveLearning
```

(continues on next page)

(continued from previous page)

```

Steps
  Type List
  List 100 3333 4567 7777
End
End

```

Active Learning Step	1:	100 MD Steps (cumulative:	100)
Active Learning Step	2:	3233 MD Steps (cumulative:	3333)
Active Learning Step	3:	1234 MD Steps (cumulative:	4567)
Active Learning Step	4:	3210 MD Steps (cumulative:	7777)
Active Learning Step	5:	2223 MD Steps (cumulative:	10000)

Steps input

```

ActiveLearning
  Steps
    Geometric
      NumSteps integer
      Start integer
    End
    Linear
      Start integer
      StepSize integer
    End
    List integer_list
  Type [Geometric | List | Linear]
End
MaxAttemptsPerStep integer
MaxReferenceCalculationsPerAttempt integer
JobPrefix string
End

```

ActiveLearning

Type
Block

Description
Settings for Active Learning

Steps

Type
Block

Description
Settings to determine the number of MD steps per active learning step.

Geometric

Type
Block

Description
Options for geometric.

NumSteps

Type

Integer

Default value

10

Description

The number of active learning steps to perform. The MD simulation will be split into this number of active learning steps. The active learning steps will progressively contain more and more MD steps.

Start**Type**

Integer

Default value

10

Description

The length of the first step (in MD time steps).

Linear**Type**

Block

Description

Options for linear.

Start**Type**

Integer

Default value

10

Description

The length of the first step (in MD time steps).

StepSize**Type**

Integer

Default value

1000

Description

The length of every subsequent active learning step (in MD time steps).

List**Type**

Integer List

Description

List of MD frame indices, for example 10 50 200 1000 10000 100000. Only indices smaller than `MolecularDynamics%NSteps` are considered. Note: the final frame of the MD simulation is always considered to be the end of a step and does not need to be specified here.

Type

Type

Multiple Choice

Default value

Geometric

Options

[Geometric, List, Linear]

GUI name

Step sequence type:

Description

How to determine the number of MD steps per active learning step.

MaxAttemptsPerStep**Type**

Integer

Default value

15

Description

Maximum number of attempts per active learning step. If this number is exceeded, the active learning will continue to the next step even if the potential is not accurate enough according to the criteria. If the default value is exceeded, it probably means that the criteria are too strict.

MaxReferenceCalculationsPerAttempt**Type**

Integer

Default value

4

GUI name

Max ref calcs per attempt:

Description

Maximum number of reference calculations per attempt. For successful attempts, only a single reference calculation is performed. For very short active learning steps, fewer calculations are done than the number specified.

JobPrefix**Type**

String

Default value**Description**

Jobs added to the job collection will receive this prefix. Example: set to `water_` to get jobs like `water_step1_attempt1_frame001`. If the prefix does not end with an underscore `_`, one will be automatically added.

4.3.4 Success criteria

At the end of an active learning step, a reference calculation (`stepX_attemptY_ref_calc1`) is performed on the last frame of the MD simulation.

The results (energy and forces) from this reference calculation are compared to the results of the trained ML potential.

Only if the agreement is accurate enough, such that **all** success criteria are fulfilled, will the Active Learning workflow continue to the next Active Learning Step.

Energy: total and relative

Enable the energy success checker with `ActiveLearning%SuccessCriteria%Energy%Enabled`.

Energies can optionally be normalized by some number before making the comparison, by specifying the `ActiveLearning%SuccessCriteria%Energy%Normalization` input option.

By default energies are normalized by the number of atoms. This is suitable for reasonably homogeneous systems and means that the same criteria can be used for any number of atoms.

You may consider changing the `Normalization` if your system is very inhomogeneous, for example if you're looking at single atom diffusing in a large bulk crystal.

Total energy

The `ActiveLearning%SuccessCriteria%Energy%Total` compares the ML-predicted energy E_{pred} directly to the reference energy E_{ref} :

- $\Delta E = E_{\text{pred}} - E_{\text{ref}}$
- Success if $|\Delta E|/\text{Normalization} < \text{ActiveLearning\%SuccessCriteria\%Energy\%Total}$

Relative energy

Compare the difference between calculated relative reference energies and relative predicted energies.

This success criterion is not invoked for `step1_attempt1` but for all subsequent steps and attempts.

- $\Delta E_{\text{ref}} = E_{\text{ref}}^{\text{current}} - E_{\text{ref}}^{\text{previous}}$
- $\Delta E_{\text{pred}} = E_{\text{pred}}^{\text{current}} - E_{\text{pred}}^{\text{previous}}$
- $\Delta \Delta E = \Delta E_{\text{pred}} - \Delta E_{\text{ref}}$
- Success if $|\Delta \Delta E|/\text{Normalization} < \text{ActiveLearning\%SuccessCriteria\%Energy\%Relative}$

Forces (gradients)

Enable the forces success criterion with `ActiveLearning%SuccessCriteria%Forces%Enabled`.

The predicted forces are compared to the reference forces in three ways:

- Mean absolute error (MAE) in eV/angstrom, `MaxMAE`
- R^2 in the correlation plot between reference and predicted values, `MinR2`
- Maximum deviation, `MaxDeviationForZeroForce`

For structures with large components, it is usually not so important the the forces are predicted very accurately, as they represent unstable structures that are unlikely to appear in an MD simulation. For large force components, one can accept a larger error (deviation) between the reference and predicted values.

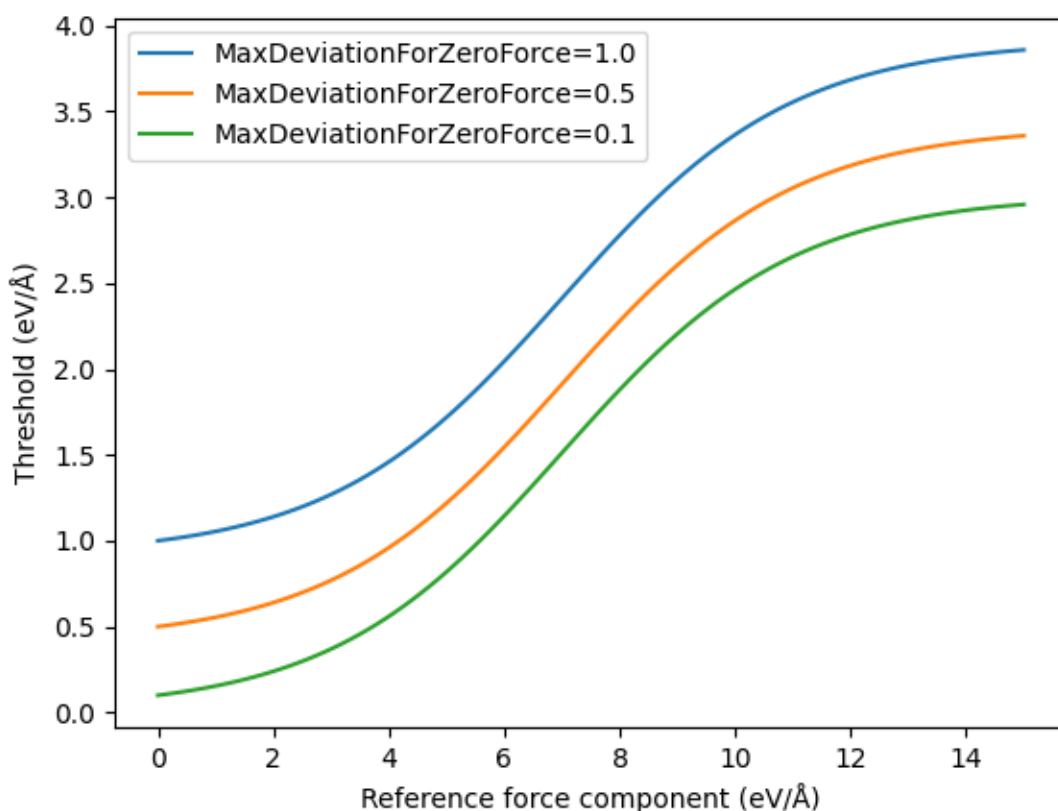
For this reason, the maximum deviation criterion depends on the magnitude of the reference force. The maximum allowed deviation between predicted and reference force components is determined by the following equation:

$$y(x) = y_0 + \frac{L}{1 + \exp(-k(|x| - x_0))} - \frac{L}{1 + \exp(-k(-x_0))}$$

where y is the threshold, x is the reference force, y_0 is `MaxDeviationForZeroForce`, $L = 3$, $x_0 = 7$, and $k = 0.5$.

There is no theoretical basis for this equation other than that it in practice seems to give reasonable thresholds.

This gives the following calculated threshold vs. reference force for a few different values of `MaxDeviationForZeroForce`:



Success criteria input

```
ActiveLearning
  SuccessCriteria
    Energy
      Enabled Yes/No
      Normalization float
      Relative float
      Total float
    End
    Forces
      Enabled Yes/No
      MaxDeviationForZeroForce float
      MaxMAE float
      MinR2 float
    End
  End
End
```

ActiveLearning

Type

Block

Description

Settings for Active Learning

SuccessCriteria

Type

Block

Description

Criteria for determining whether an active learning step was successful. These criteria compare one or more reference calculations to the predictions. If any of the criteria are exceeded, the active learning loop will reparametrize the model and repeat the step.

Energy

Type

Block

Description

Conditions to decide whether the calculated energy is accurate enough with respect to reference energies.

Enabled

Type

Bool

Default value

Yes

Description

Enable energy checking during the active learning.

Normalization

Type

Float

Description

Normalize (divide) energies by this number before comparing to the specified thresholds. If not specified, it will become the number of atoms.

Relative**Type**

Float

Default value

0.005

Unit

eV

GUI name

Relative energy:

Description

$|\Delta\Delta E|$ /Normalization: Maximum allowed difference between the calculated relative reference energies and relative predicted energies. The relative energies are calculated for the current structure with respect to the structure in the previous reference calculation.

$$\Delta E_{\text{ref}} = E_{\text{ref}}(\text{current}) - E_{\text{ref}}(\text{previous}).$$

$$\Delta E_{\text{pred}} = E_{\text{pred}}(\text{current}) - E_{\text{pred}}(\text{previous}).$$

$$|\Delta\Delta E| = |\Delta E_{\text{pred}} - \Delta E_{\text{ref}}|$$

Total**Type**

Float

Default value

0.2

Unit

eV

GUI name

Total energy:

Description

$|\Delta E|$ /Normalization: Maximum allowed total energy difference between the reference and predicted energy. This criterion is mostly useful when restarting a workflow from a previously trained model but on a new stoichiometry / system, for which the total energy prediction may be very far from the target. The default value is quite large so it is normally not exceeded.

$$|\Delta E| = |E_{\text{pred}} - E_{\text{ref}}|$$

Forces**Type**

Block

Description

Conditions to decide whether calculated forces are accurate enough with respect to reference forces.

Enabled

Type

Bool

Default value

Yes

Description

Enable checking the forces during the active learning.

MaxDeviationForZeroForce**Type**

Float

Default value

0.5

Unit

eV/angstrom

Description

The maximum allowed deviation between a calculated force component and the corresponding reference force component. For larger reference forces, the allowed deviation will also be larger (see the documentation). If any deviation is larger than the (magnitude-dependent) threshold, the active learning step will be repeated after a reparametrization.

MaxMAE**Type**

Float

Default value

0.3

Unit

eV/angstrom

GUI name

Max MAE:

Description

Maximum allowed mean absolute error when comparing reference and predicted forces for a single frame at the end of an active learning step. If the obtained MAE is larger than this threshold, the active learning step will be repeated after a reparametrization.

MinR2**Type**

Float

Default value

0.2

GUI nameMin R²:**Description**

Minimum allowed value for R² when comparing reference and predicted forces for a single frame at the end of an active learning step. If the obtained R² is smaller than this threshold, the active learning step will be repeated after a reparametrization. Note that if you have very small forces (for example by running the active learning at a very low temperature or starting from a geometry-optimized structure), then you should decrease the MinR2 since it is difficult for the ML model predict very small forces accurately.

4.3.5 Reasonable simulation criteria (uncertainties, temperature, ...)

When running MD simulations with ML potentials, it may happen that the simulation explores configurational space where the ML potential was not trained.

This can lead to strange behavior like

- atoms crashing into each other
- extremely high temperatures

The active learning workflow will try to detect these events and discard any subsequent structures.

If you train a ParAMS ML Committee (`MachineLearning%CommitteeSize > 1`), the ML model will also return an estimated uncertainty of predicted energies and forces.

You can also set a **threshold for these uncertainties**, such that if they are exceeded the MD simulation immediately stops, even before the end of the active learning step. You can thus choose to use the predicted uncertainties to decide when to stop the simulation, and use structures with high uncertainty for the training set. This method can be used in addition to *active learning step division* (page 90).

Criterion	Implementation
Temperature	inside active learning workflow
Distance	AMS Exit Condition
Energy uncertainty	AMS Exit Condition
Forces uncertainty	AMS Exit Condition

Note: If a “reasonable simulation criterion” is exceeded, this will never count as a successful step/attempt.

It will always lead to a retraining of the model and an increase of the attempt number, **even if MaxAttemptsPerStep is exceeded**.

```
ActiveLearning
  ReasonableSimulationCriteria
    Distance
      Enabled Yes/No
      MinValue float
    End
    EnergyUncertainty
      Enabled Yes/No
      MaxValue float
      Normalization float
    End
    GradientsUncertainty
      Enabled Yes/No
      MaxValue float
    End
    Temperature
      Enabled Yes/No
      MaxValue float
    End
  End
End
```

ActiveLearning

Type

Block

Description

Settings for Active Learning

ReasonableSimulationCriteria**Type**

Block

Description

Criteria for determining whether a simulation is reasonable. If any of the criteria are exceeded, this will be reported as 'ENERGY_UNCERTAINTY', 'TEMPERATURE', etc., with capital letters in the output. If a simulation is unreasonable, it will never lead to an increase of the Step, even if the number of attempts exceeds `MaxAttemptsPerStep`.

Distance**Type**

Block

Description

Stop the simulation if any interatomic distance is smaller than the specified value.

Enabled**Type**

Bool

Default value

Yes

Description

Stop the simulation if any interatomic distance is smaller than the specified value.

MinValue**Type**

Float

Default value

0.6

Unit

angstrom

GUI name

Minimum

Description

Minimum allowed interatomic distance.

EnergyUncertainty**Type**

Block

Description

Stop the simulation if the uncertainty in the energy is too high. Currently only applicable when training committees.

Enabled

Type

Bool

Default value

No

Description

Stop the simulation if the uncertainty in the energy is too high. Currently only applicable when training committees. If CommitteeSize = 1 then this keyword has no effect.

MaxValue**Type**

Float

Default value

0.015

Unit

eV

GUI name

Maximum

Description

Threshold for allowed [energy uncertainty divided by Normalization].

Normalization**Type**

Float

Description

Normalize (divide) the energy uncertainty by this number before comparing to the specified threshold. If not specified, it will become the number of atoms.

GradientsUncertainty**Type**

Block

Description

Stop the simulation if the uncertainty in the gradients (forces) is too high. Currently only applicable when training committees.

Enabled**Type**

Bool

Default value

No

Description

Stop the simulation if the uncertainty in the gradients (forces) is too high. Currently only applicable when training committees. If CommitteeSize = 1 then this keyword has no effect.

MaxValue**Type**

Float

Default value

0.5

Unit

eV/angstrom

GUI name

Maximum

Description

Maximum allowed gradients (forces) uncertainty.

Temperature**Type**

Block

Description

Discard all frames after the temperature has reached the specified value.

Enabled**Type**

Bool

Default value

Yes

Description

Discard all frames after the temperature has reached the specified value.

MaxValue**Type**

Float

Default value

5000.0

Unit

K

GUI name

Maximum

Description

Maximum allowed temperature

4.3.6 From scratch training

By default, ParAMS will reuse the parameters from the previous step/attempt as a starting point for the parametrization. Sometimes, this means that the optimizer gets stuck in a local minimum that is good for the structures encountered early in the simulation, but not for later ones.

By enabling FromScratchTraining, the optimization can be made to start from the original parameters with a given probability. Here the “original parameters” refers to what you would get if there was no `LoadModel` provided in the `MachineLearning` input block.

```
ActiveLearning
  FromScratchTraining
    Enabled Yes/No
    EpochMultiplier float
    Probability float
```

(continues on next page)

(continued from previous page)

End
End

ActiveLearning

Type

Block

Description

Settings for Active Learning

FromScratchTraining

Type

Block

Description

Custom options when training ‘from scratch’ (not restarting).

Enabled

Type

Bool

Default value

No

Description

With the given probability, start parameter training from the original starting point (from ‘scratch’) instead of restarting from the previous step/attempt.

EpochMultiplier

Type

Float

Default value

5.0

Description

The maximum number of epochs is multiplier by this number when training from scratch. When not restarting from the previous parameters, it is usually a good idea to train for more epochs.

Probability

Type

Float

Default value

0.1

Description

With the given probability, start parameter training from the original starting point (from ‘scratch’) instead of restarting from the previous step/attempt.

4.3.7 Output to save

The active learning workflow produces many directories containing reference calculations, MD simulations, and ParAMS training. You can choose how much output to save.

By default, the workflow only keeps the directories it needs to keep going. This **always** includes

- the entire training and validation sets, and
- the MD trajectory from the beginning of the workflow.

By default, the reference calculation directories are **not** saved unless the reference calculation fails.

```
ActiveLearning
  Save
    ReferenceCalculations [None | All]
    ReferenceData [Latest | All]
    TrainingDirectories [Latest | All]
    Trajectories [Latest | All]
  End
End
```

ActiveLearning

Type

Block

Description

Settings for Active Learning

Save

Type

Block

Description

The files/directories on disk to keep. If you set these options to `All`, a lot of output will be created. This output is usually not necessary but can be used for debugging purposes, or to better understand what the workflow is doing.

ReferenceCalculations

Type

Multiple Choice

Default value

None

Options

[None, All]

Description

The reference calculation directories (`initial_reference_calculations` or `stepX_attemptY_ref_calcZ`) including the original input and output.

These directories may take up a lot of disk space and are not kept by default. Enable this option if you need to investigate why reference calculations fail (incorrect input, SCF convergence problems, ...), or if you want to keep them for some other reason.

Note: The output used for parametrization (energy, forces) is always stored in the Reference-Data (training and validation sets).

ReferenceData

Type

Multiple Choice

Default value

Latest

Options

[Latest, All]

Description

The reference data directories (`stepX_attemptY_reference_data`) containing the training and validation sets in ParAMS .yaml format (and ASE .xyz format). These can be opened in the ParAMS GUI or used as input for ParAMS.

TrainingDirectories**Type**

Multiple Choice

Default value

Latest

Options

[Latest, All]

Description

The ParAMS training directories (`stepX_attemptY_training`).

Trajectories**Type**

Multiple Choice

Default value

Latest

Options

[Latest, All]

Description

The MD trajectory calculation directories (`stepX_attemptY_simulation`) using the trained ML potential. Note: the trajectories in these directories are the entire trajectories from the beginning of the simulation.

4.3.8 At workflow end: retrain model, rerun simulation

Retrain model

After the final active learning step, you have the option to retrain the model using all reference data.

This may be useful to not “waste” reference calculations that have been performed but not used for training.

Example: if the the last 3 active learning *steps* (page 90) are successful at the first attempt, then the workflow will have run 3 reference calculations (for the evaluation of the success criteria) that have not been used for training or validation.

The downside of retraining the model is that you may end up with a model that would have failed the success criteria!

By default, the model is not automatically retrained.

Rerun simulation (final production simulation)

After the final active learning step is successful, you can rerun the entire MD simulation from scratch using the final model parameters.

This will give you an MD trajectory with consistent sampling frequency and calculated using a single potential energy surface.

It is run in a directory called `final_production_simulation`, and replaces the `ams.rkf` file in the main results directory.

The *reasonable simulation criteria* (page 101) are **not** applied to the final production simulation.

AtEnd input

```
ActiveLearning
  AtEnd
    RerunSimulation Yes/No
    RetrainModel Yes/No
  End
End
```

ActiveLearning

Type

Block

Description

Settings for Active Learning

AtEnd

Type

Block

Description

What to do at the end of the active learning loop.

RerunSimulation

Type

Bool

Default value

Yes

Description

Rerun the MD simulation (folder: `final_production_simulation`) using the last set of parameters. This guarantees that the entire trajectory is calculated using the same model / potential energy surface, and that the trajectory has a consistent sampling frequency. This means that it can be used with all MD postanalysis tools.

RetrainModel

Type

Bool

Default value

No

Description

Train a final model (folder: `final_training`) using all reference (training and validation) data, including any reference calculations that have not yet been trained to.

4.4 Output

See also:

- [SALPythonAPI](#) (page 176)
- How to specify which *output to save* (page 106)

Simple Active Learning produces the following output directories:

- `simple_active_learning.results`: The main results folder
- `loaded_training`: copy of the previous ParAMS training directory if *MachineLearning%LoadModel* (page 85) is specified.
- `initial_training`: ParAMS training directory to the *initial reference data* (page 87)
- `stepX_attemptY_simulation`: MD simulation with (re-)trained ML model parameters
- `stepX_attemptY_ref_calc1`: Reference calculation for the *success criterion* (page 96)
- `stepX_attemptY_ref_calcN`: (N>1) Additional reference calculations if the step was unsuccessful
- `stepX_attemptY_reference_data`: Directory containing reference data in the ParAMS .yaml format, including the reference calculations for the same X and Y
- `stepX_attemptY_training`: ParAMS training directory for unsuccessful steps
- `final_training`: ParAMS training directory if *retraining the model after the last step* (page 107)
- `final_production_simulation`: *MD simulation run from scratch* (page 108) using the final model parameters

4.5 Python Examples

These examples show how to run **Simple Active Learning** with Python.

Getting Started

4.5.1 Single molecule: setup and run

Note: This example requires AMS2024 or later.

To follow along, either

- Download `sal_single_molecule_setup_run.py` (run as `$AMSBIN/amspython sal_single_molecule_setup_run.py`).
- Download `sal_single_molecule_setup_run.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports

from scm.simple_active_learning import SimpleActiveLearningJob
import scm.plams as plams
from scm.external_engines.core import interface_is_installed

assert interface_is_installed("m3gnet"), "You must first install m3gnet with the AMS_
↳package manager"

# ## Initialize PLAMS

plams.init()

# ## Input system

mol = plams.from_smiles("OCC=O")
for at in mol:
    at.properties = {}
plams.plot_molecule(mol)

# ## Reference engine settings
# For time reasons we use the UFF force field as the reference method. Typically you_
↳would instead train to DFT using ADF, BAND, or Quantum ESPRESSO.

ref_s = plams.Settings()
ref_s.input.ForceField.Type = "UFF"
ref_s.runscript.nproc = 1

print(plams.AMSJob(settings=ref_s).get_input())

# ## Molecular dynamics settings
# Here, we use the convenient ``AMSNVTJob`` recipe to easily initialize some MD_
↳settings.

md_s = plams.AMSNVTJob(temperature=300, timestep=0.5, nsteps=10000).settings

print(plams.AMSJob(settings=md_s).get_input())

# ## ParAMS ML Training settings
#
# (Technical note: When using ``SimpleActiveLearningJob`` the ParAMS settings go_
↳under ``input.ams``. When using ``ParAMSJob`` the settings instead simply go under_
↳``input``. See the ParAMS Python tutorials.)

ml_s = plams.Settings()
```

(continues on next page)

(continued from previous page)

```

ml_s.input.ams.MachineLearning.Backend = "M3GNet"
ml_s.input.ams.MachineLearning.CommitteeSize = 1
ml_s.input.ams.MachineLearning.M3GNet.Model = "UniversalPotential"
ml_s.input.ams.MachineLearning.MaxEpochs = 200
print(SimpleActiveLearningJob(settings=ml_s).get_input())

# ## Active learning settings

al_s = plams.Settings()
al_s.input.ams.ActiveLearning.Steps.Type = "Geometric"
al_s.input.ams.ActiveLearning.Steps.Geometric.Start = 10 # 10 MD frames
al_s.input.ams.ActiveLearning.Steps.Geometric.NumSteps = 5 # 10 AL steps
print(SimpleActiveLearningJob(settings=al_s).get_input())

# ## Simple Active Learning Job

settings = ref_s + md_s + ml_s + al_s
job = SimpleActiveLearningJob(settings=settings, molecule=mol, name="sal")
print(job.get_input())

# ## Run the job

job.run(watch=True)

```

4.5.2 Single molecule: access results

Note: This example requires AMS2024 or later.

This example shows how to get results from *Single molecule: setup and run* (page 109), so run through that example first!

To follow along, either

- Download `sal_single_molecule_results.py` (run as `$AMSBIN/amsipython sal_single_molecule_results.py`).
- Download `sal_single_molecule_results.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```

#!/usr/bin/env amspython
# coding: utf-8

# ## Load a SimpleActiveLearningJob from disk
#
# Use ``load_external`` to load the job from the previous Jupyter Notebook tutorial.
# ↳ Make sure to provide the correct path!

from scm.simple_active_learning import SimpleActiveLearningJob
import scm.plams as plams
import matplotlib.pyplot as plt
import os

```

(continues on next page)

(continued from previous page)

```

# replace the path with your own path !
previous_sal_job_path = os.path.expandvars("$AMSHOME/examples/SAL/Output/
↳SingleMolecule/plams_workdir/sal")
job = SimpleActiveLearningJob.load_external(previous_sal_job_path)

# ## Access the log file
#
# The results of the active learning are printed in a human-friendly format in the_
↳log file. For example, let's print the last few lines:

n_lines = 40
end_of_logfile = "\n".join(job.results.read_file("simple_active_learning.log").split(
↳"\n")[-n_lines:])
print(end_of_logfile)

# Above we can easily see that there were 5 active learning steps, and the engine_
↳settings for the final trained ML potential.
#
# Tip: You can copy-paste the lines from ``Engine MLPotential`` to ``EndEngine`` into_
↳AMSinput to use those engine settings for other production simulations in the GUI.

# ## Access the MD trajectories
#
# By default, the ``ActiveLearning%AtEnd%RerunSimulation`` option is enabled. This_
↳means that after the active learning loop has finished, the entire simulation is_
↳rerun from scratch with the final set of parameters.
#
# There are thus two trajectories:
#
# * A trajectory run only with the final parameters, and that is just a normal AMS_
↳job. This trajectory is located in the directory ``final_production_simulation`` if_
↳it exists.
#
# * A trajectory where the parameters have been updated on-the-fly, and which may_
↳also have an inconsistent MD sampling frequency. This trajectory is located in one_
↳of the ``stepX_attemptY_simulation`` directories.
#
# Use the ``get_simulation_directory`` method to get the corresponding directories.
#
# Let's start with the final production simulation:

final_production_simulation_dir = job.results.get_simulation_directory(allow_
↳final=True)
print(final_production_simulation_dir)

# View the trajectory in AMSmovie:

final_job = plams.AMSJob.load_external(final_production_simulation_dir)
final_rkf = final_job.results.rkfpath()

# Let's then get the other (with on-the-fly-updated-engine) trajectory:

```

(continues on next page)

(continued from previous page)

```

onthe-fly_simulation_dir = job.results.get_simulation_directory(allow_final=False)
print(onthe-fly_simulation_dir)

onthe-fly_job = plams.AMSJob.load_external(onthe-fly_simulation_dir)
onthe-fly_rkf = onthe-fly_job.results.rkfpath()

# Let's compare energy-vs-frame for the two trajectories:

plt.plot(
    final_job.results.get_history_property("Time", "MDHistory"),
    final_job.results.get_history_property("EngineEnergy"),
)
plt.plot(
    onthe-fly_job.results.get_history_property("Time", "MDHistory"),
    onthe-fly_job.results.get_history_property("EngineEnergy"),
)
plt.legend(["Final", "On-the-fly"])
plt.xlabel("Time (fs)")
plt.ylabel("Engine energy (hartree)")

# The energy profiles look quite similar. For the on-the-fly trajectory, there are
→ more datapoints for short times since the Simple Active Learning tool samples more
→ frequently in the beginning of the simulation when there are only a few MD steps
→ per active learning step.

# ## Access the ParAMS training results
#
# Similarly to the final production trajectory, there is an input option
→ ``ActiveLearning%AtEnd%RetrainModel`` which will retrain the model at the end,
→ guaranteeing that all the generated reference data is used during the training or
→ validation. However, this option is off by default.
#
# The method ``get_params_results_directory()`` returns the ParAMS results directory,
→ which can be
# * used as the value for ``MachineLearning%LoadModel`` to continue with another
→ active learning run, or
# * opened in the ParAMS GUI to view all results, including loss function
→ minimization and predicted-vs-reference scatter plots
#
# The method ``get_params_job()`` returns a ``ParAMSJob`` whose results can directly
→ be accessed using the normal ``ParAMSJob`` and ``ParAMSResults`` python APIs.

params_results_dir = job.results.get_params_results_directory(allow_final=True)
print(params_results_dir)

# **Open it in the ParAMS GUI**:

# The ParAMS GUI is the best way to quickly get an overview of the data sets and
→ results.
#
# However, it can also be useful to **access results from Python**.
#

```

(continues on next page)

(continued from previous page)

```
# The details of the ParAMSJob and ParAMSResults classes are shown in the ParAMS_
↳Python examples, here we just provide a quick example:

params_job = job.results.get_params_job()
params_job.results.plot_simple_correlation("forces", source="best")

# If you want to access individual data entries or the MAE in Python, you can use the_
↳`params_job.results.get_data_set_evaluator()` method. See the ParAMS_
↳DataSetEvaluator documentation for details.

# ## Access the production engine settings
#
# The engine settings used for production simulation can be accessed from the ParAMS_
↳job:

engine_settings = params_job.results.get_production_engine_settings()
print(plams.AMSJob(settings=engine_settings).get_input())

# ## Access the reference data

# The ``stepX_attemptY_reference_data`` directories can be accessed using ``get_
↳reference_data_directory()``:

ref_dir = job.results.get_reference_data_directory()
print(ref_dir)
```

4.5.3 Single molecule: Compare to M3GNet-UP-2022

Note: This example requires AMS2024 or later.

This example uses results from *Single molecule: setup and run* (page 109), so run through that example first!

To follow along, either

- Download `sal_single_molecule_compare_to_m3gnet_up_2022.py` (run as `$AMSBIN/amsipython sal_single_molecule_compare_to_m3gnet_up_2022.py`).
- Download `sal_single_molecule_compare_to_m3gnet_up_2022.ipynb` (see also: how to install Jupyterlab in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# ## Load the Simple Active Learning job from disk.
#
# ``retrained_params_job`` is the best ParAMS training job that was done during the_
↳Active Learning.

from scm.simple_active_learning import SimpleActiveLearningJob
from scm.params.plams.paramsjob import ParAMSJob
from scm.params import ResultsImporter, NoParameters
```

(continues on next page)

(continued from previous page)

```

import scm.plams as plams
import os

# replace the path with your own path !
previous_sal_job_path = os.path.expandvars("$AMSHOME/examples/SAL/Output/
↳SingleMolecule/plams_workdir/sal")
sal_job = SimpleActiveLearningJob.load_external(previous_sal_job_path)
retrained_params_job = sal_job.results.get_params_job()

# ## Get results for M3GNet-UP-2022 universal potential
#
# Are the retrained results any better than those from the M3GNet-UP-2022 universal
↳potential?
#
# To find out, we need to evaluate the training and validation sets also with M3GNet-
↳UP-2022. This can be done with the ParAMS "SinglePoint" task, which does not
↳perform any parameter optimization but instead evaluates the training and
↳validation sets with a given engine.
#
# To set the engine settings, we need to call the ``set_extra_engine()`` method on
↳the job collection and then store the results in a new folder that can be read by
↳the new ParAMSJob. The easiest way to achieve this is to use the
↳``ResultsImporter`` class, even though we do not import any new results. When
↳running the SinglePoint, we also have to explicitly specify that there are no
↳parameters and store the ``NoParameters`` parameter interface:

m3gnet_up_s = plams.Settings()
m3gnet_up_s.input.MLPotential.Model = "M3GNet-UP-2022"
ri = ResultsImporter.from_params_results(retrained_params_job.results)
ri.job_collection.set_extra_engine(m3gnet_up_s)
print(ri.job_collection.engines)

pi = NoParameters()
folder = "m3gnet-up-data"
ri.store(folder, backup=False) # will create the directory
pi.yaml_store(f"{folder}/parameter_interface.yaml")
os.listdir(folder)

# We can now **initialize the new ParAMS SinglePoint job and run it**:

new_params_job = ParAMSJob.from_yaml(folder)
new_params_job.settings.input.Task = "SinglePoint"
new_params_job.name = "m3gnet-up"

plams.init(folder="plams_workdir_singlepoint_validation")
new_params_job.run()

# ### M3GNet-UP-2022 (predicted) forces vs. the reference (here UFF) forces

new_params_job.results.plot_simple_correlation("forces", source="best", title="M3GNet-
↳UP-2022")

```

(continues on next page)

(continued from previous page)

```
# Here we can see that M3GNet-UP-2022 gives quite different force prediction compared
↳to our chosen reference method (UFF force field).
#
# Note that M3GNet-UP-2022 was trained to PBE DFT data, and the plot above shows the
↳agreement to the UFF force field. The plot does not show the agreement to the PBE
↳level of theory to which M3GNet-UP-2022 was originally trained!

# ### Retrained M3GNet (predicted) forces vs. the reference (here UFF) forces

retrained_params_job.results.plot_simple_correlation("forces", source="best", title=
↳"Retrained M3GNet")

# This is the same plot as shown in the previous tutorial. We can see that the active
↳learning retraining has led to significant improvements in reproducing the
↳reference data!
```

4.5.4 Single molecule: Production simulation with retrained ML potential

Note: This example requires AMS2024 or later.

This example uses results from *Single molecule: setup and run* (page 109), so run through that example first!

To follow along, either

- Download `sal_single_molecule_production_simulation.py` (run as `$AMSBIN/amsipython sal_single_molecule_production_simulation.py`).
- Download `sal_single_molecule_production_simulation.ipynb` (see also: [how to install Jupyterlab in AMS](#))

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports and load active learning job from disk

from scm.simple_active_learning import SimpleActiveLearningJob
import scm.plams as plams
import os
import matplotlib.pyplot as plt

plams.init()

# replace the path with your own path !
previous_sal_job_path = os.path.expandvars("$AMSHOME/examples/SAL/Output/
↳SingleMolecule/plams_workdir/sal")
sal_job = SimpleActiveLearningJob.load_external(previous_sal_job_path)
params_job = sal_job.results.get_params_job()

# ## Structure for production job
```

(continues on next page)

(continued from previous page)

```

#
# We could initialize a PLAMS molecule in many different ways. Here, we get the final_
↳ frame from the final production simulation in the SAL job, and preoptimize it with_
↳ UFF.

molecule = sal_job.results.get_main_molecule()
molecule = plams.preoptimize(molecule, model="UFF")
plams.plot_molecule(molecule)

# ## Settings for production job
#
# Let's run a geometry optimization + frequencies. But you could run any AMS job!
#
# The ``MaxRestarts`` option is useful when calculating normal modes. If the geometry_
↳ optimizer converges to a transition state, it will continue until it finds a local_
↳ minimum!

s = plams.Settings()
s.input.ams.Task = "GeometryOptimization"
s.input.ams.Properties.NormalModes = "Yes"
s.input.ams.GeometryOptimization.MaxRestarts = 5
s.runscript.nproc = 1

retrained_engine_settings = params_job.results.get_production_engine_settings()

new_job = plams.AMSJob(settings=s + retrained_engine_settings, name="retrained_m3gnet_
↳", molecule=molecule)
print(new_job.get_input())

new_job.run()

# ### Optimized structure

plams.plot_molecule(new_job.results.get_main_molecule())

# ### Frequencies

width = 50 # cm-1
x, y = new_job.results.get_ir_spectrum(broadening_width=width, post_process="all_
↳ intensities_to_1")
plt.plot(x, y, label="Retrained M3GNet")
plt.xlabel("Frequency (cm-1)")
plt.ylabel("Normal mode count")
plt.legend()

# ## Compare to reference UFF
# In this case, we can also calculate the normal modes with the reference method_
↳ (UFF) and compare:

ref_engine_settings = plams.Settings()
ref_engine_settings.input.ForceField.Type = "UFF"
ref_job = plams.AMSJob(settings=s + ref_engine_settings, name="uff_ref",

```

(continues on next page)

(continued from previous page)

```

↪molecule=molecule)
ref_job.run()

retrained_structure_rmsd = plams.Molecule.rmsd(
    ref_job.results.get_main_molecule(),
    new_job.results.get_main_molecule(),
    ignore_hydrogen=True,
)
print(f"Structural RMSD: {retrained_structure_rmsd:.2f} angstrom")

x_ref, y_ref = ref_job.results.get_ir_spectrum(broadening_width=width, post_process=
↪"all_intensities_to_1")
plt.plot(x, y, label="Retrained M3GNet")
plt.plot(x_ref, y_ref, label="UFF (reference method)")
plt.xlabel("Frequency (cm-1)")
plt.ylabel("Normal mode count")
plt.legend()

# The agreement looks very good! The only significant difference is the highest-
↪frequency vibration (the O-H stretching vibration). This frequency is very sensitive
↪to the calculated forces near the equilibrium (minimum) structure. The agreement
↪could have been improved by
#
# * having more training data, for example by setting a tighter success criterion for
↪the forces and energy in the active learning
# * running the active learning MD at a lower temperature (closer to the equilibrium
↪structure, but this would mean less conformational sampling)
# * training for more epochs
#
# Tip: check if the vibrational frequencies with retrained M3GNet or M3GNet-UP-2022
↪agree better or worse with the reference UFF calculation.

```

4.5.5 Continue active learning with a new system or new simulation settings

Note: This example requires AMS2024 or later.

This example uses results from *Single molecule: setup and run* (page 109), so run through that example first!

To follow along, either

- Download `sal_continue_with_new_system.py` (run as `$AMSBIN/amsipython sal_continue_with_new_system.py`).
- Download `sal_continue_with_new_system.ipynb` (see also: how to install Jupyterlab in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# ## Initialization

from scm.simple_active_learning import SimpleActiveLearningJob
import scm.plams as plams
import matplotlib.pyplot as plt
import os

plams.init(folder="plams_workdir_continuation")

# Set the correct path to the previous Simple Active Learning job. The path should be
→ a directory containing the file "simple_active_learning.rkf"

# replace the path with your own path !
previous_sal_job_path = os.path.expandvars("$AMSHOME/examples/SAL/Output/
→ SingleMolecule/plams_workdir/sal")
previous_sal_job = SimpleActiveLearningJob.load_external(previous_sal_job_path)
previous_params_path = previous_sal_job.results.get_params_results_directory()

# ## Initial system, reference engine settings, MD settings
#
# These settings were explained in the first tutorial.
#
# Here we use a new molecule (acetic acid), but we could also have changed the
→ temperature of the MD simulation, or any other setting.

mol = plams.from_smiles("CC(=O)O")
for at in mol:
    at.properties = {}
mol = plams.preoptimize(mol)
plams.plot_molecule(mol)

ref_s = plams.Settings()
ref_s.input.ForceField.Type = "UFF"
ref_s.runscript.nproc = 1

md_s = plams.AMSNVTJob(temperature=300, timestep=0.5, nsteps=10000).settings

# ## ParAMS ML training settings
# Here we set ``LoadModel = previous_params_path`` to load the model from the
→ previous job.
#
# This will also automatically load the previous training and validation data, unless
→ it's disabled in the Active Learning settings.

ml_s = plams.Settings()
ml_s.input.ams.MachineLearning.Backend = "M3GNet"
ml_s.input.ams.MachineLearning.CommitteeSize = 1
```

(continues on next page)

(continued from previous page)

```

ml_s.input.ams.MachineLearning.LoadModel = os.path.abspath(previous_params_path)
ml_s.input.ams.MachineLearning.MaxEpochs = 200

# ## Active Learning settings
#
# Here we use the same settings as before, but if the system is similar (or even the
# same!) as before, you may consider increasing the ``Start`` to let the system
# evolve a bit more before the first reference calculation.
#
# You can also set the ``ActiveLearning.InitialReferenceData.Load.Directory`` option
# instead of the ``MachineLearning.LoadModel`` option to load the data from the
# previous run. See the documentation for details about the difference between the
# two options.

al_s = plams.Settings()
al_s.input.ams.ActiveLearning.Steps.Type = "Geometric"
al_s.input.ams.ActiveLearning.Steps.Geometric.Start = 10 # 10 MD frames
al_s.input.ams.ActiveLearning.Steps.Geometric.NumSteps = 5 # 5 AL steps
# alternative to ml_s.input.ams.MachineLearning.LoadModel:
# al_s.input.ams.ActiveLearning.InitialReferenceData.Load.Directory = os.path.
# abspath(previous_params_path)
al_s.input.ams.ActiveLearning.InitialReferenceData.Generate.ReferenceMD.Enabled = "Yes"

# ## Simple Active Learning job
#
# We can run the active learning as before.
#
# Note that the training jobs now take longer than before since the training and
# validation sets are bigger.

settings = ref_s + md_s + ml_s + al_s
job = SimpleActiveLearningJob(settings=settings, molecule=mol, name="sal")
job.run(watch=True)

```

Case studies

4.5.6 Liquid water: diffusion coefficient, radial distribution function, density

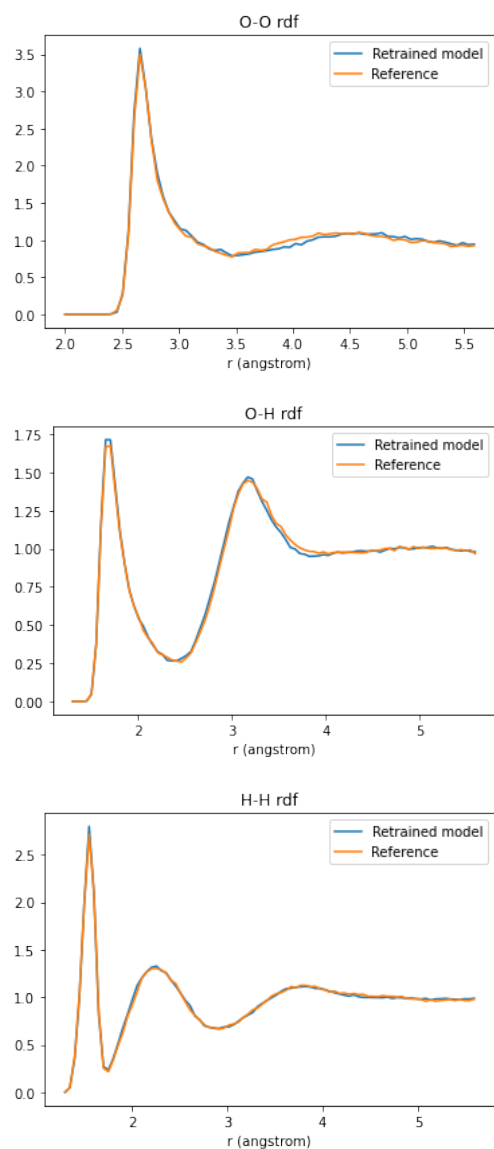
Trained model: M3GNet, starting from the Universal Potential (UP)

Reference method: ReaxFF Water2017.ff; J. Phys. Chem. B, 2017, 121 (24), pp 6021–6032 (<http://dx.doi.org/10.1021/acs.jpcb.7b02548>)

System: Liquid water at T = 300 K

Property	Reference	Retrained M3GNet	M3GNet-UP	Experiment
Self-diffusion ($10^{-9} \text{ m}^2 \text{ s}^{-1}$)	2.6	2.5	0.23	2.3
Density (g cm^{-3})	1.01	1.02	0.95	1.00

Water self-diffusion coefficient calculated for a small box size and not corrected for finite-size effects. The values may not be fully converged. When following this tutorial you may get different values.

Radial distribution functions (RDFs)

Expected duration: This notebook takes about 24 hours to run on the CPU. This includes both the active learning and the production simulations to get the diffusion coefficient, radial distribution functions, and density.

To follow along, either

- Download `liquid_water_training.py` (run as `$AMSBIN/amsipython liquid_water_training.py`).
- Download `liquid_water_training.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# When running active learning it's usually a good idea to start off "simple" and
# → make the system/structures gradually more complicated.
#
# To train liquid water, we here
#
# * first train a potential at slightly above room temperature and 1.0 g/cm^3
#
# * continue with a second active learning loop where the density is explicitly
# → scanned from a low to a high value
#
# ## Initial imports

import scm.plams as plams
from scm.simple_active_learning import SimpleActiveLearningJob
import matplotlib.pyplot as plt
from scm.external_engines.core import interface_is_installed

assert interface_is_installed("m3gnet"), "You must install the m3gnet backend before
# → following this tutorial!"

plams.init()

# ## Create an initial water box

water = plams.from_smiles("O")
for at in water:
    at.properties = plams.Settings()
plams.plot_molecule(water)

box = plams.packmol(water, n_molecules=48, density=1.0)
plams.plot_molecule(box, rotation="-5x,5y,0z")

# Let's run a short MD simulation with M3GNet-UP-2022 to make the structure more
# → realistic:

up_s = plams.Settings()
up_s.input.MLPotential.Model = "M3GNet-UP-2022"
up_s.runscript.nproc = 1 # always run AMS Driver in serial for MLPotential
up_md = plams.AMSNVTJob(
    molecule=box,
    settings=up_s,
    name="up_md",
    nsteps=1000,
    timestep=0.5,
    temperature=350,
)
up_md.run()
```

(continues on next page)

(continued from previous page)

```

# New structure:

starting_structure = up_md.results.get_main_molecule()
plams.plot_molecule(starting_structure, rotation="-5x,5y,0z")

# ## Simple Active Learning setup
#
# ### Reference engine settings
#
# Here, we choose to train against ReaxFF Water-2017.ff, which gives a good water_
↪structure.

fast_ref_s = plams.Settings()
fast_ref_s.input.ReaxFF.ForceField = "Water2017.ff"
fast_ref_s.runscript.nproc = 1

slow_ref_s = plams.Settings()
slow_ref_s.input.QuantumESPRESSO.Pseudopotentials.Family = "SSSP-Efficiency"
slow_ref_s.input.QuantumESPRESSO.Pseudopotentials.Functional = "PBE"
slow_ref_s.input.QuantumESPRESSO.K_Points._h = "gamma"
slow_ref_s.input.QuantumESPRESSO.System = plams.Settings(
    input_dft="revpbe", ecutwfc=40, vdw_corr="Grimme-D3", dftd3_version=4
)

# Change to slow_ref_s to train to revPBE-D3(BJ) instead:

ref_s = fast_ref_s.copy()

# ### NVT molecular dynamics settings

nvt_md_s = plams.AMSNVTJob(
    nsteps=20000,
    timestep=0.5,
    temperature=(270, 350, 350),
    tau=100,
    thermostat="Berendsen",
).settings

# ### ParAMS machine learning settings

ml_s = plams.Settings()
ml_s.input.ams.MachineLearning.Backend = "M3GNet"
ml_s.input.ams.MachineLearning.CommitteeSize = 1
ml_s.input.ams.MachineLearning.M3GNet.Model = "UniversalPotential"
ml_s.input.ams.MachineLearning.MaxEpochs = 200

# ### Active learning settings
#
# Liquid water is a "simple" homogeneous system, so we can expect the ML method to_
↪perform quite well. We therefore decrease the success criteria thresholds a bit_
↪compared to the default vvalues, to ensure that we get accurate results.

```

(continues on next page)

(continued from previous page)

```

#
# Since we will immediately continue with another active learning loop, we disable
# the "RerunSimulation" as we are not interested in the MD simulation per se.

al_s = plams.Settings()
al_s.input.ams.ActiveLearning.Steps.Type = "Geometric"
al_s.input.ams.ActiveLearning.Steps.Geometric.Start = 10
al_s.input.ams.ActiveLearning.Steps.Geometric.NumSteps = 8
al_s.input.ams.ActiveLearning.InitialReferenceData.Generate.M3GNetShortMD.Enabled =
# "Yes"
al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Relative = 0.003
al_s.input.ams.ActiveLearning.SuccessCriteria.Forces.MaxDeviationForZeroForce = 0.35
al_s.input.ams.ActiveLearning.AtEnd.RerunSimulation = "No"

# ### Complete job

settings = ref_s + nvt_md_s + ml_s + al_s
job = SimpleActiveLearningJob(settings=settings, molecule=starting_structure, name=
# "sal")
print(job.get_input())

# ### Run the simple active learning job

job.run/watch=True

# ## Validate trained model by RDF and MSD
#
# Note: You should skip this part if you trained to DFT since the reference MD
# calculation will take a very long time!

mol = job.results.get_main_molecule()
plams.plot_molecule(mol, rotation="-5x,5y,0z")

retrained_model_settings = job.results.get_params_job().results.get_production_engine_
# settings()
retrained_model_settings.runscript.nproc = 1

# ### Equilibration and production MD settings

eq_md_settings = plams.AMSNVTJob(
    nsteps=8000,
    timestep=0.5,
    thermostat="Berendsen",
    tau=100,
    temperature=300,
    samplingfreq=100,
).settings

prod_md_settings = plams.AMSNVTJob(
    nsteps=50000,
    timestep=0.5,
    thermostat="NHC",

```

(continues on next page)

(continued from previous page)

```

    tau=200,
    temperature=300,
    samplingfreq=100,
).settings

# ### Retrained model equilibration

retrained_model_eq_md_job = plams.AMSJob(
    settings=eq_md_settings + retrained_model_settings,
    molecule=mol,
    name="retrained_model_eq_md_dens_1",
)
retrained_model_eq_md_job.run()

# ### Retrained model production simulation

# Let's then run a production simulation from the final structure of the above_
↳ equilibration MD using both the retrained model and the reference engine:

retrained_model_prod_md_job = plams.AMSJob(
    settings=prod_md_settings + retrained_model_settings,
    name="retrained_model_prod_md_dens_1",
    molecule=retrained_model_eq_md_job.results.get_main_molecule(),
)
retrained_model_prod_md_job.run()

# ### Reference equilibration MD

reference_eq_md_job = plams.AMSJob(
    settings=eq_md_settings + ref_s,
    molecule=mol,
    name="reference_eq_md_dens_1",
)
reference_eq_md_job.run()

# ### Reference production MD

reference_prod_md_job = plams.AMSJob(
    settings=prod_md_settings + ref_s,
    name="reference_prod_md_dens_1",
    molecule=reference_eq_md_job.results.get_main_molecule(),
)
reference_prod_md_job.run()

# ### Mean squared displacement (MSD) helper functions

# For a detailed explanation of the MSD and RDF jobs, see the "Molecular dynamics_
↳ with Python" tutorial

def get_msd_job(job: plams.AMSJob, symbol: str = "O"):
    atom_indices = [i for i, at in enumerate(job.results.get_main_molecule(), 1) if

```

(continues on next page)

(continued from previous page)

```

↪at.symbol == symbol]

    msd_job = plams.AMSMSDJob(
        job,
        name="msd-" + job.name,
        atom_indices=atom_indices, # indices start with 1 for the first atom
        max_correlation_time_fs=4000, # max correlation time must be set before_
↪running the job
        start_time_fit_fs=2000, # start_time_fit can also be changed later in the_
↪postanalysis
    )
    msd_job.run()

    return msd_job

def plot_msd(job, start_time_fit_fs=None):
    """job: an AMSMSDJob"""
    time, msd = job.results.get_msd()
    fit_result, fit_x, fit_y = job.results.get_linear_fit(start_time_fit_fs=start_
↪time_fit_fs)
    # the diffusion coefficient can also be calculated as fit_result.slope/6 (ang^2/
↪fs)
    diffusion_coefficient = job.results.get_diffusion_coefficient(start_time_fit_
↪fs=start_time_fit_fs) # m^2/s
    plt.figure(figsize=(5, 3))
    plt.plot(time, msd, label="MSD")
    plt.plot(fit_x, fit_y, label="Linear fit slope={:.5f} ang^2/fs".format(fit_result.
↪slope))
    plt.legend()
    plt.xlabel("Correlation time (fs)")
    plt.ylabel("Mean square displacement (ang^2)")
    plt.title("MSD: Diffusion coefficient = {:.2e} m^2/s".format(diffusion_
↪coefficient))

# ### Temporarily turn off PLAMS logging
#
# Technically, the MSD and RDF jobs are normal PLAMS jobs. However, they are very_
↪fast to run. We can turn off the PLAMS logging to keep the Jupyter notebook a bit_
↪more tidy:

plams.config.log.stdout = 0

# ### Retrained model MSD, diffusion coefficient

retrained_model_msd_job = get_msd_job(retrained_model_prod_md_job, "O")
retrained_model_D = retrained_model_msd_job.results.get_diffusion_coefficient() #_
↪diffusion coefficient, m^2/s
plot_msd(retrained_model_msd_job)

# ### Reference MSD, diffusion coefficient

reference_msd_job = get_msd_job(reference_prod_md_job, "O")
reference_D = reference_msd_job.results.get_diffusion_coefficient() # diffusion_

```

(continues on next page)

(continued from previous page)

```

↪coefficient, m^2/s
plot_msd(reference_msd_job)

# **Conclusion for diffusion coefficient**: In this case, the retrained model gives 2.
↪53e-9 m^2/s and the reference method 2.62e-9 m^2/s. That is very good agreement!↪
↪Note: your results may be somewhat different.

# ### Retrained model and reference RDF
#
# Let's compare the calculated O-O, O-H, and H-H radial distribution functions (rdf):

def get_rdf(job, atom_indices, atom_indices_to, rmin, rmax, rstep):
    rdf = plams.AMSRDFJob(
        job,
        atom_indices=atom_indices,
        atom_indices_to=atom_indices_to,
        rmin=rmin,
        rmax=rmax,
        rstep=rstep,
    )
    rdf.run()

    return rdf.results.get_rdf()

final_frame = retrained_model_prod_md_job.results.get_main_molecule() # doesn't↪
↪matter if retrained model or reference
O_ind = [i for i, at in enumerate(final_frame, 1) if at.symbol == "O"]
H_ind = [i for i, at in enumerate(final_frame, 1) if at.symbol == "H"]
rmax = final_frame.lattice[0][0] / 2
rstep = 0.05

# ### O-O rdf

atom_indices, atom_indices_to = O_ind, O_ind
rmin = 2.0
pred_x, pred_y = get_rdf(retrained_model_prod_md_job, atom_indices, atom_indices_to,↪
↪rmin, rmax, rstep)
ref_x, ref_y = get_rdf(reference_prod_md_job, atom_indices, atom_indices_to, rmin,↪
↪rmax, rstep)
plt.plot(pred_x, pred_y, label="Retrained model")
plt.plot(ref_x, ref_y, label="Reference")
plt.xlabel("r (angstrom)")
plt.legend()
plt.title("O-O rdf")

# ### O-H rdf

atom_indices, atom_indices_to = O_ind, H_ind
rmin = 1.3
pred_x, pred_y = get_rdf(retrained_model_prod_md_job, atom_indices, atom_indices_to,↪
↪rmin, rmax, rstep)
ref_x, ref_y = get_rdf(reference_prod_md_job, atom_indices, atom_indices_to, rmin,↪

```

(continues on next page)

(continued from previous page)

```

    ↪rmax, rstep)
plt.plot(pred_x, pred_y, label="Retrained model")
plt.plot(ref_x, ref_y, label="Reference")
plt.xlabel("r (angstrom)")
plt.legend()
plt.title("O-H rdf")

# ### H-H rdf

atom_indices, atom_indices_to = H_ind, H_ind
rmin = 1.3
pred_x, pred_y = get_rdf(retrained_model_prod_md_job, atom_indices, atom_indices_to, ↪
    ↪rmin, rmax, rstep)
ref_x, ref_y = get_rdf(reference_prod_md_job, atom_indices, atom_indices_to, rmin, ↪
    ↪rmax, rstep)
plt.plot(pred_x, pred_y, label="Retrained model")
plt.plot(ref_x, ref_y, label="Reference")
plt.xlabel("r (angstrom)")
plt.legend()
plt.title("H-H rdf")

# ### Turn PLAMS logging back on

plams.config.log.stdout = 3 # default value

# ## Density and NPT
#
# ### Check the predicted vs. reference density

npt_md_s = plams.AMSNPTJob(
    nsteps=100000,
    timestep=0.5,
    thermostat="NHC",
    tau=100,
    temperature=300,
    barostat="MTK",
    barostat_tau=1000,
    equal="XYZ",
    pressure=1e5,
).settings

retrained_model_npt_job = plams.AMSJob(
    settings=npt_md_s + retrained_model_settings,
    name="retrained_model_npt",
    molecule=retrained_model_prod_md_job.results.get_main_molecule(),
)

retrained_model_npt_job.run()

retrained_model_density = (
    plams.AMSNPTJob.load_external(retrained_model_npt_job.results.rkfpath())
    .results.get_equilibrated_molecule()

```

(continues on next page)

(continued from previous page)

```

        .get_density()
    )
    print(f"Retrained model water density at 300 K: {retrained_model_density*1e-3:.2f} g/
    ↪cm^3")

    plams.config.jobmanager.hashing = None
    reference_npt_job = plams.AMSJob(
        settings=npt_md_s + ref_s,
        name="reference_npt",
        molecule=reference_prod_md_job.results.get_main_molecule(),
    )

    reference_npt_job.run()

    reference_density = (
        plams.AMSNPTJob.load_external(reference_npt_job.results.rkfpath()).results.get_
        ↪equilibrated_molecule().get_density()
    )
    print(f"Reference model water density at 300 K: {reference_density*1e-3:.2f} g/cm^3")

    # The above reference value for ReaxFF Water-2017.ff agrees exactly with the
    ↪published reference value of 1.01 g/cm^3.
    #
    # However, the retrained M3GNet model predicts a density of 0.95 g/cm^3. The
    ↪agreement is reasonable but not excellent. This can be explained by the fact that
    ↪almost all training data points were at 1.00 g/cm^3. Only a few points (from the
    ↪"M3GNetShortMD" initial reference data generator) were taken at other densities.
    #
    # Let's continue the active learning while sampling more densities. There are two
    ↪strategies:
    #
    # * Use an NPT simulation during the active learning
    # * Scan the density during the active learning
    #
    # Here, we choose the second approach in order to ensure that multiple different
    ↪densities are sampled.

    # ### Initial structure for scanning density
    #
    # Get the final frame from one of the previous MD simulations, and linearly scale the
    ↪density to 800 kg/m^3 = 0.8 g/cm^3. This will stretch out the O-H bonds so follow
    ↪up with a short UFF preoptimization.

    new_structure = final_frame.copy()
    new_structure.set_density(850)
    new_structure = plams.preoptimize(new_structure, model="uff", maxiterations=20)

    plams.plot_molecule(new_structure)

    # ### Second active learning job: scanning density
    #
    # Here we set Steps.Type = "Linear" to run reference calculations every 5000 MD steps.

```

(continues on next page)

(continued from previous page)

```

#
# To get an accurate density it's very important that the predicted energy is
↳ accurate. It is not enough to just get a good fit for the forces.
#
# Here, we decrease the success criteria for both the energy and forces compared to
↳ default values.

nsteps = 80000
scan_density_md_s = plams.AMSMDScanDensityJob(
    molecule=new_structure,
    scan_density_upper=1.15,
    nsteps=nsteps,
    tau=100,
    thermostat="Berendsen",
    temperature=300,
).settings
# we must explicitly set the StopStep, since the AL divides the simulation into
↳ multiple segments
scan_density_md_s.input.ams.MolecularDynamics.Deformation.StopStep = nsteps

# job = SimpleActiveLearningJob.load_external(plams.config.default_jobmanager.workdir,
↳ + "/sal.002")
scan_density_ml_s = ml_s.copy()
scan_density_ml_s.input.ams.MachineLearning.LoadModel = job.results.get_params_
↳ results_directory()
scan_density_ml_s.input.ams.MachineLearning.Target.Forces.MAE = 0.02
scan_density_ml_s.input.ams.MachineLearning.MaxEpochs = 200

scan_density_al_s = plams.Settings()

scan_density_al_s.input.ams.ActiveLearning.Steps.Type = "Linear"
scan_density_al_s.input.ams.ActiveLearning.Steps.Linear.Start = 500
scan_density_al_s.input.ams.ActiveLearning.Steps.Linear.StepSize = 5000
scan_density_al_s.input.ams.ActiveLearning.InitialReferenceData.Load.
↳ FromPreviousModel = "Yes"

scan_density_al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Relative = 0.001
scan_density_al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Total = 0.002
# because we do not set Normalization, the above Energy criteria are energies per atom
# scan_density_al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Normalization =

scan_density_al_s.input.ams.ActiveLearning.SuccessCriteria.Forces.
↳ MaxDeviationForZeroForce = 0.30

scan_density_al_s.input.ams.ActiveLearning.AtEnd.RerunSimulation = "No"
scan_density_al_s.input.ams.ActiveLearning.MaxReferenceCalculationsPerAttempt = 2

scan_density_al_job = SimpleActiveLearningJob(
    name="scan_density_al",
    settings=ref_s + scan_density_md_s + scan_density_ml_s + scan_density_al_s,
    molecule=new_structure,
)
scan_density_al_job.run(watch=True)

```

(continues on next page)

(continued from previous page)

```
# Let's recalculate the density again:

new_retrained_model_settings = scan_density_al_job.results.get_params_job().results.
    ↳ get_production_engine_settings()

new_retrained_model_npt_job = plams.AMSJob(
    settings=npt_md_s + new_retrained_model_settings,
    name="new_retrained_model_npt",
    molecule=retrained_model_prod_md_job.results.get_main_molecule(),
)

new_retrained_model_npt_job.run()

new_retrained_model_density = (
    plams.AMSNPTJob.load_external(new_retrained_model_npt_job.results.rkfpath())
    .results.get_equilibrated_molecule()
    .get_density()
)

print(f"New retrained model water density at 300 K: {new_retrained_model_density*1e-
    ↳ 3:.2f} g/cm^3")

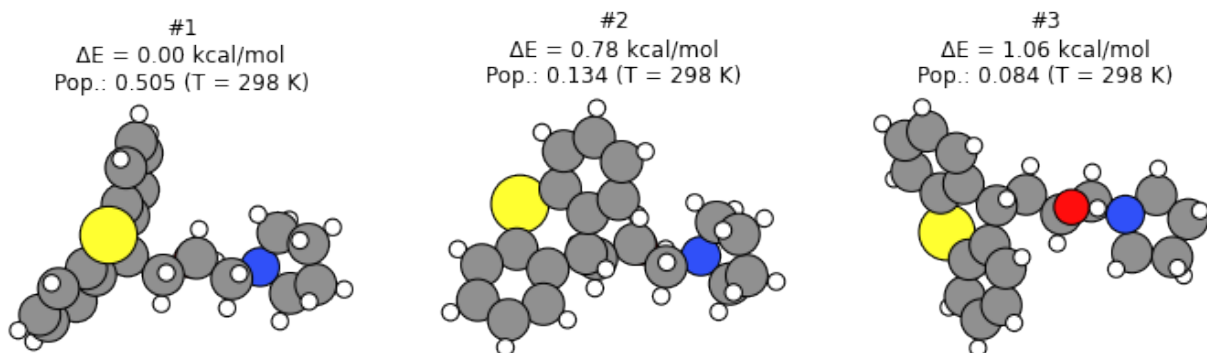
# **Conclusion for the density**: Using active learning when scanning the densities.
    ↳ makes sure that the predictions are accurate for all densities. Consequently the
    ↳ equilibrium density is also in better agreement with the reference value of 1.01 g/
    ↳ cm^3.
#
# Note that the density in general is quite difficult to fit accurately.
```

4.5.7 Conformers: Active learning with CREST metadynamics and custom addition of data points

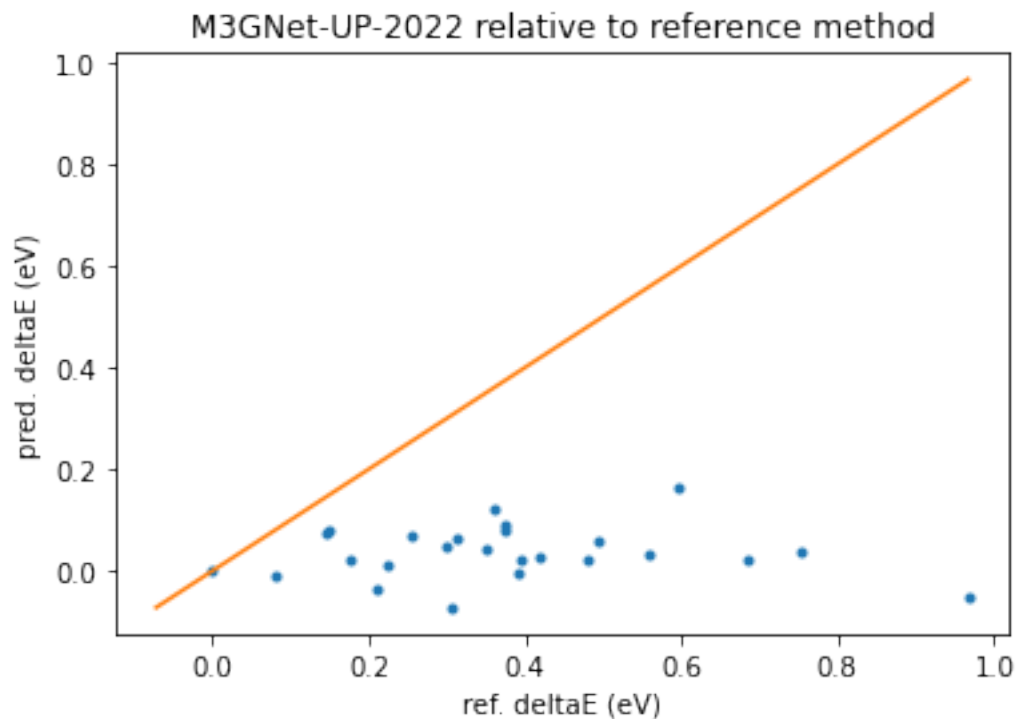
Trained model: M3GNet, starting from the Universal Potential (UP)

Reference method: GFN-1xTB (DFTB engine)

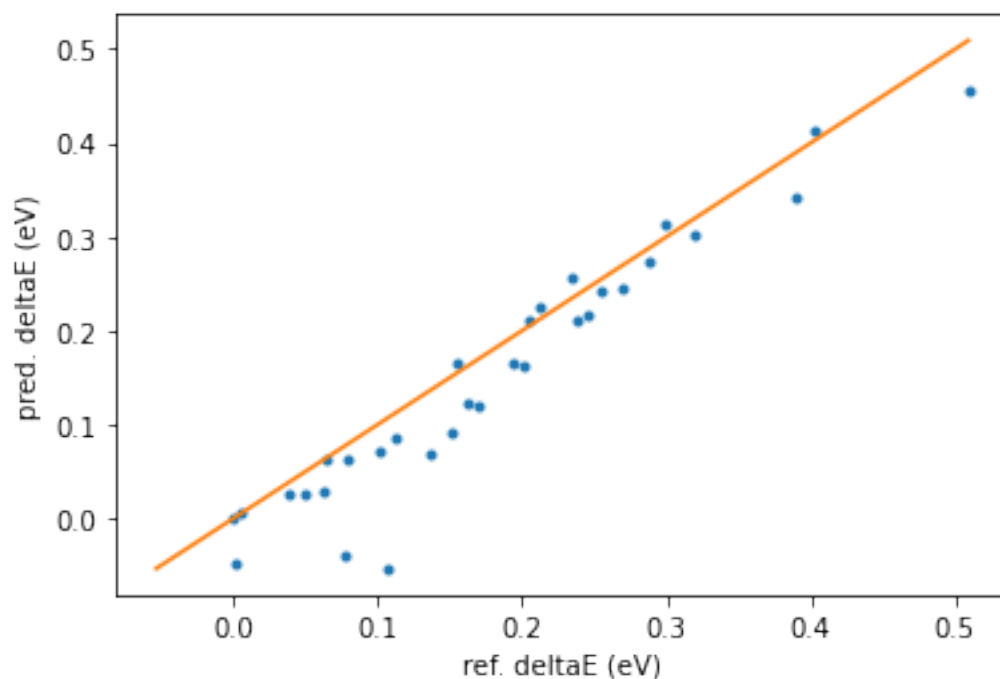
System: Organic molecule with SMILES code OC (CC1c2ccccc2Sc2ccccc21) CN1CCCC1



Problem: M3GNet-UP-2022 is not reliable for conformer stability prediction:



Solution: Retraining the model gives better agreement:



Expected duration: This notebook takes about 24 hours to run on the CPU. This includes both the active learning and the production simulations to generate new conformers.

To follow along, either

- Download `conformers_training.py` (run as `$AMSBIN/amsipython conformers_training.py`).

- Download `conformers_training.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# When running active learning it's usually a good idea to start off "simple" and
# → make the system/structures gradually more complicated.
#
# For getting a model which predicts conformers accurately, we may take the following
# → approach:
#
# * first train a potential at slightly above room temperature with NVT MD
#
# * continue training a second potential using CREST metadynamics
#
# * generate conformers with the previous model and also train to those
#
# ## Initial imports

import scm.plams as plams
import scm.params as params
from scm.simple_active_learning import SimpleActiveLearningJob
import matplotlib.pyplot as plt
from scm.external_engines.core import interface_is_installed
import os
import numpy as np
from typing import List
from scm.conformers import ConformersJob
from scm.conformers.plams.plot import plot_conformers

assert interface_is_installed("m3gnet"), "You must install the m3gnet backend before
# → following this tutorial!"

plams.init()

# ## Create the initial structure

molecule = plams.from_smiles("OC(CC1c2ccccc2Sc2ccccc21)CN1CCCC1", forcefield="uff")
molecule.delete_all_bonds()
for at in molecule:
    at.properties = plams.Settings()
plams.plot_molecule(molecule)
starting_structure = molecule

# ## Reference engine settings

fast_ref_s = plams.Settings()
fast_ref_s.input.DFTB.Model = "GFN1-xTB"

slow_ref_s = plams.Settings()
slow_ref_s.input.ADF.Basis.Type = "TZP"
slow_ref_s.input.ADF.Basis.Core = "None"
```

(continues on next page)

(continued from previous page)

```

slow_ref_s.input.ADF.XC.Hybrid = "B3LYP"
slow_ref_s.input.ADF.XC.DISPERSION = "GRIMME3 BJDAMP"

# Change to slow_ref_s to train to B3LYP-D3(BJ) instead:

ref_s = fast_ref_s.copy()
# ref_s = slow_ref_s.copy()
perform_expensive_tests = ref_s == fast_ref_s

# ## Problem statement: Generate a few conformers with M3GNet-UP-2022 and Score with
↳ reference method

m3gnet_up_s = plams.Settings()
m3gnet_up_s.input.MLPotential.Model = "M3GNet-UP-2022"

generate_conformers_m3gnet_up_job = ConformersJob(name="generate_conformers_m3gnet_up
↳", molecule=starting_structure)
generate_conformers_m3gnet_up_job.settings.input.ams.Task = "Generate"
generate_conformers_m3gnet_up_job.settings.input.ams.Generator.Method = "RDKit"
generate_conformers_m3gnet_up_job.settings.input.ams.Generator.RDKit.
↳InitialNConformers = 40
# generate_conformers_m3gnet_up_job.settings += crest_al_job.results.get_production
↳engine_settings()
generate_conformers_m3gnet_up_job.settings += m3gnet_up_s
# generate_conformers_m3gnet_up_job.settings.runscript.nproc = 1 # run in serial,
↳useful if you run out of memory when running M3GNet on the GPU

generate_conformers_m3gnet_up_job.run()

# Show the 4 most stable conformers with M3GNet-UP-2022:

plot_conformers(generate_conformers_m3gnet_up_job, 4, unit="eV")

# Score the generated conformers with the reference method:

score_conformers_ref_job = ConformersJob(name="score_conformers_ref")
score_conformers_ref_job.settings.input.ams.Task = "Score"
score_conformers_ref_job.settings.input.ams.InputConformersSet = generate_conformers_
↳m3gnet_up_job.results.rkfpath()
score_conformers_ref_job.settings.input += ref_s.input
score_conformers_ref_job.run()

plot_conformers(score_conformers_ref_job, 4, unit="eV")

# Here we can see that the ordering of conformers at the reference level is
↳completely different compared to M3GNet-UP-2022.
#
# The Score job reorders the conformers. To compare the energies more precisely, we
↳can use the minimum pairwise RMSD (which should be 0) to identify how the order of
↳conformers change:

```

(continues on next page)

(continued from previous page)

```

def get_pairwise_rmsds(molecules_ref: List[plams.Molecule], molecules_pred:
↳List[plams.Molecule]) -> np.ndarray:
    """Returns a len(molecules_ref)*len(molecules_pred) numpy array with pairwise_
↳rmsds (in angstrom) between the structures"""
    rmsds = np.zeros((len(molecules_ref), len(molecules_pred)))

    for i, ref_mol in enumerate(molecules_ref):
        for j, pred_mol in enumerate(molecules_pred):
            rmsds[i, j] = plams.Molecule.rmsd(ref_mol, pred_mol)

    return rmsds

def get_reordering(rmsds: np.ndarray) -> np.ndarray:
    """
    rmsds: numpy array with shape len(molecules_ref)*len(molecules_pred)

    Returns a len(molecules_ref) integer numpy array.
    The first element is the index (0-based) in molecules_pred corresponding to the_
↳first reference molecule, etc.
    """
    rmsds = get_pairwise_rmsds(molecules_ref, molecules_pred)
    reordering = np.argmin(rmsds, axis=1)
    return reordering

def print_reordering_table(molecules_ref, molecules_pred, energies_ref, energies_pred,
↳ax=None):
    """This functions prints the reordering table including relative energies. It_
↳also plots the predicted relative energies versus the reference relative energies"""
    print(f"Ref_i  Pred_i  RMSD  Ref_dE  Pred_dE")
    x, y = [], []
    rmsds = get_pairwise_rmsds(molecules_ref, molecules_pred)
    reordering = get_reordering(rmsds)
    rmsd_threshold = 0.7 # angstrom, for printing/plotting points
    for ref_i, pred_i in enumerate(reordering):
        rmsd = rmsds[ref_i, pred_i]
        ref_relative_e = energies_ref[ref_i]
        pred_relative_e = energies_pred[pred_i] - energies_pred[reordering[0]]
        if rmsd <= rmsd_threshold:
            print(f"{ref_i:6d} {pred_i:6d} {rmsd:4.1f} {ref_relative_e:7.2f} {pred_
↳relative_e:7.2f}")
            x.append(ref_relative_e)
            y.append(pred_relative_e)
        else:
            print(f"{ref_i:6d} {pred_i:6d} rmsd > {rmsd_threshold:1f} ang.")

    if ax is None:
        _, ax = plt.subplots()

    m, M = np.min([np.min(x), np.min(y)]), np.max([np.max(x), np.max(y)])
    ax.plot(x, y, ".")
    ax.plot([m, M], [m, M], "--")
    ax.set_xlabel("ref. deltaE (eV)")
    ax.set_ylabel("pred. deltaE (eV)")

```

(continues on next page)

(continued from previous page)

```

return ax

# In the below table the energies are given with respect to the structure that had
↳ the lowest reference energy. This is thus different compared to the first image
↳ above, where the predicted (m3gnet-up-2022) relative energies were given to
↳ predicted lowest energy conformer.

molecules_ref = score_conformers_ref_job.results.get_conformers()
energies_ref = score_conformers_ref_job.results.get_relative_energies(unit="eV")
molecules_pred = generate_conformers_m3gnet_up_job.results.get_conformers()
energies_pred = generate_conformers_m3gnet_up_job.results.get_relative_energies(unit=
↳ "eV")
ax = print_reordering_table(molecules_ref, molecules_pred, energies_ref, energies_
↳ pred)
ax.set_title("M3GNet-UP-2022 relative to reference method")

# Here we see that there is no correlation between the relative stabilities
↳ calculated with M3GNet-UP-2022 and the reference method.
#
# Example: The most stable conformer with the reference method (Ref_i=0) corresponds
↳ to the tenth most stable conformer with M3GNet-UP-2022 (Pred_i=10)
#
# Example 2: The second most stable conformer with the reference method (Ref_i=1)
↳ should be 0.08 eV *less* stable than the ref_i=0 conformer, but with M3GNet-UP-2022
↳ method it is actually 0.01 eV *more* stable!

# ## Optimize a few conformers for initial reference data
#
# Conformers are local minima on the potential energy surface. The Active Learning MD
↳ will sample off-equilibrium structures. So let's make sure that there are at least
↳ some local minima in the training set by optimizing some of the generated
↳ conformers.
#
# Here, we loop over the conformers and run GeometryOptimization jobs. This produces
↳ output files in the normal AMS format that can easily be imported into ParAMS

max_N = min(6, len(molecules_pred)) # at most 6 optimizations
opt_s = plams.Settings()
opt_s.input.ams.Task = "GeometryOptimization"
opt_s.input.ams.GeometryOptimization.Convergence.Quality = "Basic"
opt_jobs = [
    plams.AMSJob(settings=opt_s + ref_s, name=f"opt_{i}", molecule=mol) for i, mol in
↳ enumerate(molecules_pred[:max_N])
]
for opt_job in opt_jobs:
    opt_job.run()

# Now import the data into a ParAMS results importer and store in the directory ``my_
↳ initial_reference_data``:

yaml_dir = os.path.abspath("my_initial_reference_data")
ri = params.ResultsImporter(settings={"units": {"energy": "eV", "forces": "eV/angstrom
↳ "}}})

```

(continues on next page)

(continued from previous page)

```

for opt_job in opt_jobs:
    ri.add_singlejob(opt_job, properties=["energy", "forces"], task="SinglePoint")

ri.store(yaml_dir, backup=False)

# ## Simple Active Learning setup
#

# ### Molecular dynamics settings (temperature ramp)

nvt_md_s = plams.AMSNVTJob(
    nsteps=20000,
    timestep=0.5,
    temperature=(270, 350, 350),
    tau=100,
    thermostat="Berendsen",
).settings

# ### ParAMS machine learning settings

ml_s = plams.Settings()
ml_s.input.ams.MachineLearning.Backend = "M3GNet"
ml_s.input.ams.MachineLearning.CommitteeSize = 1
ml_s.input.ams.MachineLearning.M3GNet.Model = "UniversalPotential"
ml_s.input.ams.MachineLearning.MaxEpochs = 200

# ### Active learning settings
#
# Conformer search of a single molecule is quite simple, so we can expect the ML
↪ method to perform quite well. We therefore decrease the success criteria thresholds
↪ a bit compared to the default values, to ensure that we get accurate results.
#
# Since we will immediately continue with another active learning loop, we disable
↪ the "RerunSimulation" as we are not interested in the MD simulation per se.

al_s = plams.Settings()
al_s.input.ams.ActiveLearning.Steps.Type = "Geometric"
al_s.input.ams.ActiveLearning.Steps.Geometric.Start = 10
al_s.input.ams.ActiveLearning.Steps.Geometric.NumSteps = 8
al_s.input.ams.ActiveLearning.InitialReferenceData.Load.Directory = yaml_dir
al_s.input.ams.ActiveLearning.InitialReferenceData.Generate.M3GNetShortMD.Enabled =
↪ "Yes"
al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Relative = 0.002
al_s.input.ams.ActiveLearning.SuccessCriteria.Forces.MaxDeviationForZeroForce = 0.30
al_s.input.ams.ActiveLearning.AtEnd.RerunSimulation = "No"

# ### Initial structure
#
# Let's start with the lowest-energy conformer according to the reference method:

starting_structure_al = molecules_ref[0]

```

(continues on next page)

(continued from previous page)

```

# The bonds are not used, so delete them to make the input file less confusing:
starting_structure_al.delete_all_bonds()

# ### Complete job

settings = ref_s + nvt_md_s + ml_s + al_s
job = SimpleActiveLearningJob(settings=settings, molecule=starting_structure_al, name=
↳ "sal")
print(job.get_input())

# ### Run the simple active learning job

job.run(watch=True)

# ### Final structure from MD simulation
#

new_structure = job.results.get_main_molecule()
plams.plot_molecule(new_structure, rotation="-5x,5y,0z")

# ## Second active learning job: CREST metadynamics
#
# Here we set Steps.Type = "Linear" to run reference calculations every 2000 MD steps.

nsteps = 20000

crest_md_s = plams.Settings()
crest_md_s.input.ams.MolecularDynamics.CRESTMtd.Height = 0.138
crest_md_s.input.ams.MolecularDynamics.CRESTMtd.NGaussiansMax = 50
crest_md_s.input.ams.MolecularDynamics.CRESTMtd.NSteps = 200
crest_md_s.input.ams.MolecularDynamics.CRESTMtd.Width = 0.62

crest_complete_md_s = plams.AMSMDJob(
    molecule=new_structure,
    nsteps=nsteps,
    settings=crest_md_s,
    tau=10, # small time constant
    thermostat="NHC",
    temperature=300,
    timestep=0.5,
    samplingfreq=20,
).settings

# job = SimpleActiveLearningJob.load_external(plams.config.default_jobmanager.workdir,
↳ "/sal.002")
crest_ml_s = ml_s.copy()
crest_ml_s.input.ams.MachineLearning.LoadModel = job.results.get_params_results_
↳ directory()
crest_ml_s.input.ams.MachineLearning.Target.Forces.MAE = 0.04
crest_ml_s.input.ams.MachineLearning.MaxEpochs = 200

```

(continues on next page)

(continued from previous page)

```

crest_al_s = plams.Settings()

crest_al_s.input.ams.ActiveLearning.Steps.Type = "Linear"
crest_al_s.input.ams.ActiveLearning.Steps.Linear.Start = 500
crest_al_s.input.ams.ActiveLearning.Steps.Linear.StepSize = 2000
crest_al_s.input.ams.ActiveLearning.InitialReferenceData.Load.FromPreviousModel = "Yes
↳"

crest_al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Relative = 0.002
crest_al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Total = 0.010
# because we do not set Normalization, the above Energy criteria are energies per atom
# crest_al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Normalization =

crest_al_s.input.ams.ActiveLearning.SuccessCriteria.Forces.MaxDeviationForZeroForce = 0.30
↳

crest_al_s.input.ams.ActiveLearning.AtEnd.RerunSimulation = "No"
crest_al_s.input.ams.ActiveLearning.MaxReferenceCalculationsPerAttempt = 3

crest_al_job = SimpleActiveLearningJob(
    name="crest_al",
    settings=ref_s + crest_complete_md_s + crest_ml_s + crest_al_s,
    molecule=new_structure,
)
crest_al_job.run(watch=True)

# crest_al_job = SimpleActiveLearningJob.load_external("plams_workdir.003/crest_al")

new_retrained_model_settings = crest_al_job.results.get_params_job().results.get_
↳production_engine_settings()

# ## Generate conformers with the retrained M3GNet model and score with reference_
↳method

def generate_and_score(
    molecule: plams.Molecule,
    gen_name: str,
    gen_settings: plams.Settings,
    score_name: str,
    score_settings: plams.Settings,
):
    generate_job = ConformersJob(name=gen_name, molecule=molecule)
    generate_job.settings.input.ams.Task = "Generate"
    generate_job.settings.input.ams.Generator.Method = "RDKit"
    generate_job.settings.input.ams.Generator.RDKit.InitialNConformers = 40
    generate_job.settings.input += gen_settings.input
    generate_job.run()

    score_job = ConformersJob(name=score_name)
    score_job.settings.input.ams.Task = "Score"
    score_job.settings.input.ams.InputConformersSet = generate_job.results.rkfpath()
    score_job.settings.input += score_settings.input
    score_job.run()

```

(continues on next page)

(continued from previous page)

```

molecules_gen = generate_job.results.get_conformers()
energies_gen = generate_job.results.get_relative_energies(unit="eV")
molecules_score = score_job.results.get_conformers()
energies_score = score_job.results.get_relative_energies(unit="ev")

    return generate_job, molecules_gen, energies_gen, score_job, molecules_score, \
↳energies_score

(
    generate_conformers_m3gnet_retrained_job,
    molecules_pred,
    energies_pred,
    _,
    molecules_ref,
    energies_ref,
) = generate_and_score(
    starting_structure,
    gen_name="generate_conformers_m3gnet_retrained",
    gen_settings=crest_al_job.results.get_production_engine_settings(),
    score_name="score_conformers_ref2",
    score_settings=ref_s,
)

print_reordering_table(molecules_ref, molecules_pred, energies_ref, energies_pred)

# We can see a significant improvement compared to the M3GNet-UP-2022 results! \
↳However, the results are not perfect.

# ## Generate conformers with the reference method and score with the retrained \
↳M3GNet model
#
# As a second test, we can instead generate the conformers with the reference method \
↳and score them with the retrained m3gnet model.
#
# This is quite expensive if the reference method is DFT!

if perform_expensive_tests:
    generate_ref_job, molecules_ref, energies_ref, _, molecules_pred, energies_pred = \
↳generate_and_score(
        starting_structure,
        gen_name="generate_conformers_ref",
        gen_settings=ref_s,
        score_name="score_conformers_m3gnet_retrained",
        score_settings=crest_al_job.results.get_production_engine_settings(),
    )
    print_reordering_table(molecules_ref, molecules_pred, energies_ref, energies_pred)
    plot_conformers(generate_ref_job, 3)

# ## Compare the RMSD between the different conformer sets
#
# The Conformer "Score" task performs single-point calculations.
#

```

(continues on next page)

(continued from previous page)

```

# If we instead change the task to "Optimize" we can see how similar the reference-
→method-optimized conformers are to the retrained-M3GNet-optimized conformers by_
→comparing the RMSD.
#
# The below is quite computationally expensive for a DFT reference engine.

if perform_expensive_tests:
    opt_conformers_ref_job2 = ConformersJob(name="opt_conformers_ref2")
    opt_conformers_ref_job2.settings.input.ams.Task = "Optimize"
    opt_conformers_ref_job2.settings.input.ams.InputConformersSet = (
        generate_conformers_m3gnet_retrained_job.results.rkfpath()
    )
    opt_conformers_ref_job2.settings.input.ams.InputMaxEnergy = (
        5.0 # only conformers in the lowest 5.0 kcal/mol = 0.2 eV
    )
    opt_conformers_ref_job2.settings.input += ref_s.input
    opt_conformers_ref_job2.run()

    molecules_ref = opt_conformers_ref_job2.results.get_conformers()
    energies_ref = opt_conformers_ref_job2.results.get_relative_energies(unit="eV")
    molecules_pred = generate_conformers_m3gnet_retrained_job.results.get_conformers()
    energies_pred = generate_conformers_m3gnet_retrained_job.results.get_relative_
→energies(unit="eV")
    print_reordering_table(molecules_ref, molecules_pred, energies_ref, energies_pred)

# In the above table a few entries have "rmsd > 0.7 ang.". This means that the_
→reference geometry optimization causes the structure to change significantly_
→compared to the retrained-m3gnet-optimized geometry.
#
# In such cases it is not so meaningful to compare the relative energies between the_
→reference and prediction, so those points are excluded from the table and from the_
→plot.

# ## Custom active learning loop: add newly generated conformers to training set
#
# The Simple Active Learning module in AMS only works for MD simulations, so it_
→cannot automatically add optimized conformers to the training or validation sets.
#
# However, you can do it yourself!
#
# The Conformers "Score" function does not store or calculate the forces. So let's_
→set up an AMS "Replay" job to recalculate the energies and forces to add to the_
→training set:

replay_s = plams.Settings()
replay_s.input.ams.Task = "Replay"
replay_s.input.ams.Replay.File = generate_conformers_m3gnet_retrained_job.results.
→rkfpath()
replay_s.input.ams.Properties.Gradients = "Yes"
replay_s += ref_s
replay_job = plams.AMSJob(settings=replay_s, name="replay_new_conformers")
replay_job.run()

# Now import the data into a results importer:

```

(continues on next page)

(continued from previous page)

```

ri = params.ResultsImporter.from_yaml(crest_al_job.results.get_reference_data_
↳directory())
ri.add_trajectory_singlepoints(replay_job, properties=["energy", "forces"])
yaml_dir = "data_with_conformer_singlepoints_yaml"
ri.store(yaml_dir, backup=False)

# Then launch ParAMS:

params_job = params.ParAMSJob.from_yaml(yaml_dir, name="params_with_conformer_
↳singlepoints")
params_job.settings.input += ml_s.input.ams
params_job.settings.input.MachineLearning.LoadModel = crest_al_job.results.get_params_
↳results_directory()
params_job.settings.input.Task = "MachineLearning"
params_job.settings.input.MachineLearning.LossCoeffs.Energy = 50
params_job.settings.input.MachineLearning.Target.Forces.Enabled = "No"
params_job.settings.input.MachineLearning.MaxEpochs = 100
params_job.run()

# If the job failed print the error message:

if not params_job.check():
    print(params_job.get_errormsg())

# Generate conformers with the new model and score with the reference method:

_, molecules_pred, energies_pred, _, molecules_ref, energies_ref = generate_and_score(
    starting_structure,
    gen_name="generate_conformers_m3gnet_retrained_again",
    gen_settings=params_job.results.get_production_engine_settings(),
    score_name="score_conformers_ref2",
    score_settings=ref_s,
)

# And print/plot the results:

print_reordering_table(molecules_ref, molecules_pred, energies_ref, energies_pred)

# Here we see even better agreement than before.
#
# **Conclusion**: By manually adding retrained-ml-optimized conformers to the_
↳training set, you can improve the conformer prediction even more. This means to do_
↳your own "active learning" outside of the Simple Active Learning module in AMS.

```

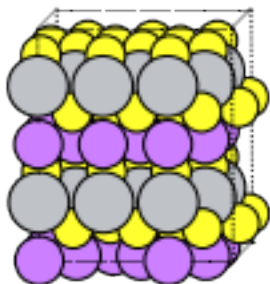
4.5.8 Li-vacancy diffusion in a solid electrolyte

Trained model: M3GNet, starting from the Universal Potential (UP)

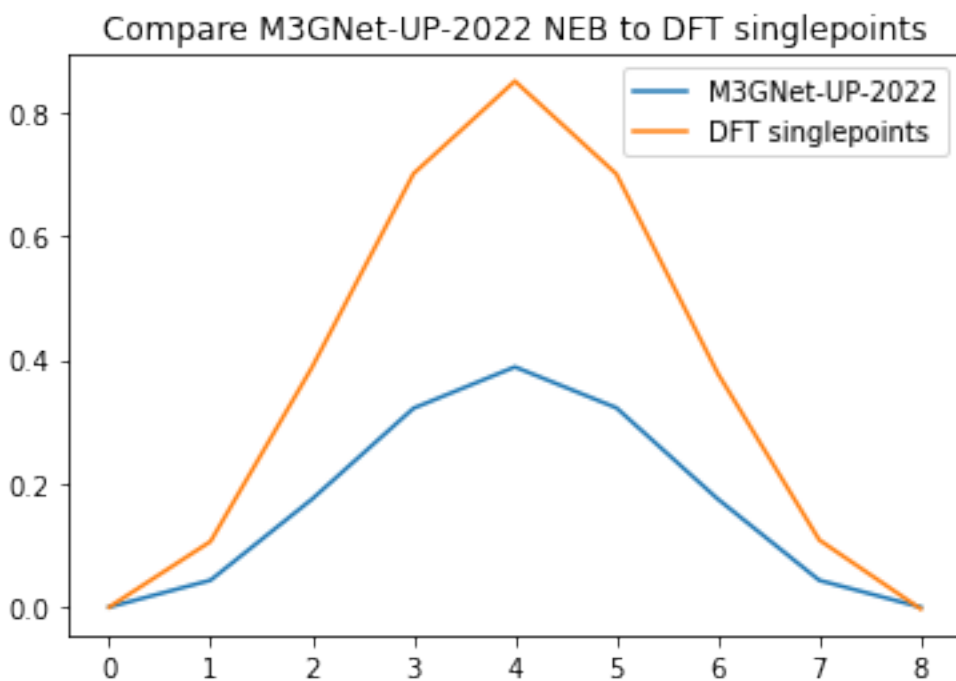
Reference method: DFT (PBE, engine Quantum ESPRESSO)

System: LiTiS_2 and $\text{Li}_{23}\text{Ti}_{24}\text{S}_{48}$ (crystal with one Li vacancy)

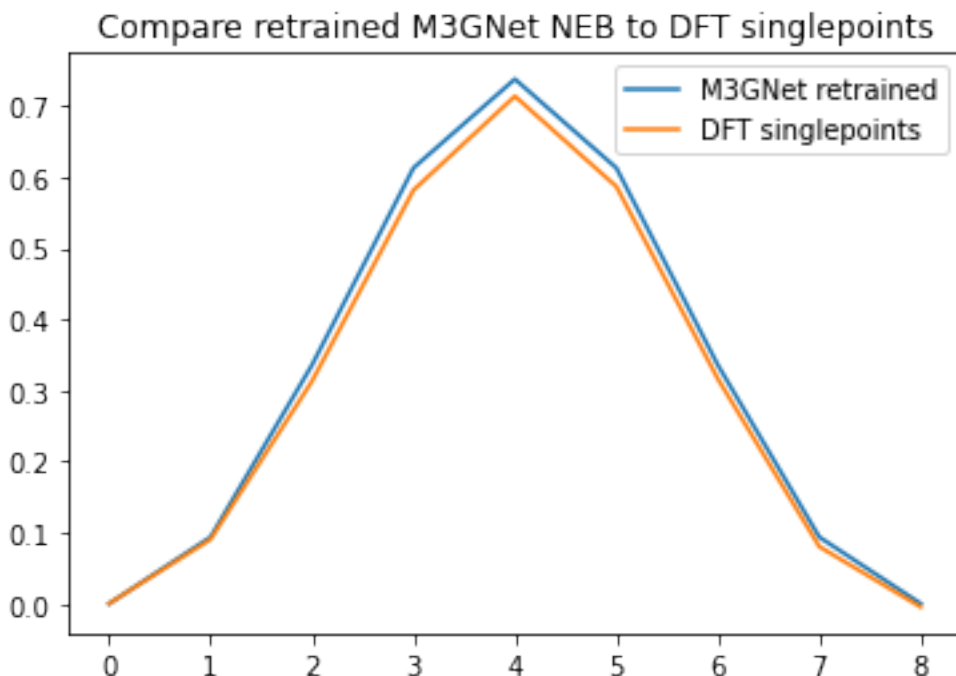
Initial structure



Problem: M3GNet-UP-2022 is underestimates the Li diffusion barrier



Solution: Retraining the model gives better agreement:



Expected duration: This notebook takes about 24 hours to run on the CPU. This includes both the active learning and the various NEB calculations.

To follow along, either

- Download `litis2_diffusion_neb.py` (run as `$AMSBIN/amsipython litis2_diffusion_neb.py`).
- Download `litis2_diffusion_neb.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# ## Initialization

import scm.plams as plams
import os
import numpy as np
import matplotlib.pyplot as plt
import scm.params as params
from scm.simple_active_learning import SimpleActiveLearningJob

plams.init()

# ## Create structures
#
# ### Create primitive cell from coordinates
# for plotting in Jupyter notebooks
```

(continues on next page)

(continued from previous page)

```

rotation = "-85x,-5y,0z"

# create structure
primitive = plams.AMSJob.from_input(
    """
    System
    Atoms
        Li 0. 0. 0.
        Ti 0. 0. 3.0874524999999999
        S 0. 1.985103430533333 4.551328274799999
        S 1.71915 0.9925517153000001 1.6235767252
    End
    Lattice
        3.4383 0.0 0.0
        -1.71915 2.977655145833333 0.0
        0.0 0.0 6.174905
    End
End
    """
).molecule[""]
plams.plot_molecule(primitive, rotation=rotation)

# ### Create a supercell.
#
# Here we a reasonably-sized supercell. If the cell is very small, it is actually
# → quite inefficient for training the machine learning potential. If every atom sees
# → its own periodic image then there will not be sufficient variety in the atomic
# → environments for the ML potential to be able to extrapolate to unseen environments.
# → **It is recommended to use a larger supercell**.

supercell = primitive.supercell([[3, 0, 0], [2, 4, 0], [0, 0, 2]])
for at in supercell:
    at.properties = plams.Settings()
plams.plot_molecule(supercell, rotation=rotation)

print("Lattice vectors")
print(supercell.lattice)
print(f"Number of atoms: {len(supercell)}")

# ### Create Li vacancy in two different places

li_indices = [i for i, at in enumerate(supercell, 1) if at.symbol == "Li"]
print(f"Li indices (starting with 1): {li_indices}")

# Pick the first Li atom, and get its nearest Li neighbor.
#
# Note that the PLAMS distance_to function ignores periodic boundary conditions,
# → which in this case is quite convenient as it will make the visualization easier if
# → the Li atom diffuses inside the unit cell and doesn't cross the periodic boundary.

first_index = li_indices[0]
li1 = supercell[first_index]
li1.properties.region = "DiffusingLi"

```

(continues on next page)

(continued from previous page)

```

distances = [li1.distance_to(supercell[x]) for x in li_indices[1:]]
nearest_neighbor_index = li_indices[np.argmin(distances) + 1]
li2 = supercell[nearest_neighbor_index]
target_coords = li2.coords
print(f"First Li atom: {li1}")
print(f"Second Li atom: {li2}")

# The goal is to make a Li atom diffuse between these two positions.
#
# For this, we will first delete the second Li atom, and then create a new structure_
↳ in which the first Li atom is translated to the second Li atom coordinates.

defective_1 = supercell.copy()
defective_1.delete_atom(defective_1[nearest_neighbor_index])
defective_2 = supercell.copy()
defective_2.delete_atom(defective_2[nearest_neighbor_index])
defective_2[first_index].coords = li2.coords

plams.plot_molecule(defective_1, rotation=rotation)
plt.title("Initial structure")

# The Li atom in the bottom left corner will diffuse to the right (the vacancy will_
↳ diffuse to the left):

plams.plot_molecule(defective_2, rotation=rotation)
plt.title("Final structure")

# ## Initial validation of diffusion barrier with NEB and M3GNet-UP-2022
#
# ### M3GNet-UP-2022 NEB job

m3gnet_up_s = plams.Settings()
m3gnet_up_s.input.MLPotential.Model = "M3GNet-UP-2022"

neb_s = plams.Settings()
neb_s.input.ams.Task = "NEB"
neb_s.input.ams.NEB.PreoptimizeWithIDPP = "Yes"
neb_s.input.ams.NEB.Images = 7

m3gnet_up_neb_job = plams.AMSJob(
    settings=m3gnet_up_s + neb_s,
    molecule={"": defective_1, "final": defective_2},
    name="m3gnet_up_neb",
)
m3gnet_up_neb_job.run()

# ### M3GNet-UP-2022 NEB results

m3gnet_up_res = m3gnet_up_neb_job.results.get_neb_results(unit="eV")
print(f"Left barrier: {m3gnet_up_res['LeftBarrier']:.3f} eV")
print(f"Right barrier: {m3gnet_up_res['RightBarrier']:.3f} eV")

```

(continues on next page)

(continued from previous page)

```

# ref_results = [0.0, 0.144, 0.482, 0.816, 0.963, 0.816, 0.482, 0.144, 0.0] # from_
↳ Quantum ESPRESSO DFT calculation
nImages = m3gnet_up_res["nImages"]
relative_energies = np.array(m3gnet_up_res["Energies"]) - m3gnet_up_res["Energies"][0]
plt.plot(relative_energies)
plt.xlabel("Image")
plt.ylabel("Relative energy (eV)")
plt.title("M3GNet-UP-2022 NEB Li diffusion")

fig, axes = plt.subplots(1, len(m3gnet_up_res["Molecules"]), figsize=(15, 5))

for i, mol in enumerate(m3gnet_up_res["Molecules"]):
    plams.plot_molecule(mol, ax=axes[i], rotation=rotation)
    axes[i].set_title(f"Image {i}")

# ## DFT reference engine settings
#
# Are the above M3GNet-UP-2022 results any good? Let's compare to DFT calculations_
↳ using the AMS "Replay" task.
#
# "Replay" is just a series of singlepoints on the previous NEB structures. It is not_
↳ a NEB calculation.
#
# Here we use the Quantum ESPRESSO engine available in AMS2024.

dft_s = plams.Settings()
dft_s.input.QuantumESPRESSO.System.occupations = "Smearing"
dft_s.input.QuantumESPRESSO.System.degauss = 0.005
# for production purposes always manually check that ecutwfc and ecutrho are high_
↳ enough
# here we set a fairly low ecutwfc to speed up the calculation
dft_s.input.QuantumESPRESSO.System.ecutwfc = 30.0
dft_s.input.QuantumESPRESSO.System.ecutrho = 240.0
# decrease mixing_beta for more robust SCF convergence
dft_s.input.QuantumESPRESSO.Electrons.mixing_beta = 0.3
dft_s.input.QuantumESPRESSO.Electrons.conv_thr = 1.0e-5 * len(supercell)
# for a small cell one should ideally use more k-points than just the gamma point
# by settings K_Points._h = "gamma" we use the faster gamma-point-only
# implementation in Quantum ESPRESSO
dft_s.input.QuantumESPRESSO.K_Points._h = "gamma"
# SCM_DISABLE_MPI launches ams in serial, but will still run Quantum ESPRESSO in_
↳ parallel
dft_s.runscript.preamble_lines = ["export SCM_DISABLE_MPI=1"]

# ## Run DFT calculations on the NEB points from M3GNet-UP-2022
#
# Note: The DFT calculations may take a few minutes to complete.

replay_s = plams.Settings()
replay_s.input.ams.Task = "Replay"
replay_s.input.ams.Replay.File = os.path.abspath(m3gnet_up_neb_job.results.rkfpath())
replay_s.input.ams.Properties.Gradients = "Yes"

```

(continues on next page)

(continued from previous page)

```

replay_dft_job = plams.AMSJob(settings=replay_s + dft_s, name="replay_m3gnet_up_neb_
↳with_dft")
replay_dft_job.run(watch=True)

# When using "Replay" on a NEB job, the results are stored in the normal NEB format.
↳Thus we can use the ``.get_neb_results()`` method also on this job:

dft_energies = replay_dft_job.results.get_neb_results(unit="eV") ["Energies"]
dft_relative_energies = np.array(dft_energies) - dft_energies[0]

plt.plot(relative_energies)
plt.plot(dft_relative_energies)
plt.legend(["M3GNet-UP-2022", "DFT singlepoints"])
plt.title("Compare M3GNet-UP-2022 NEB to DFT singlepoints")

# **Conclusion**: M3GNet-UP-2022 significantly underestimates the diffusion barrier.
↳compared to DFT calculations. **This motivates the reparametrization**

# ## Store DFT results for later
#
# Since we already performed some DFT calculations, we may as well add them to the
↳training set.

ri = params.ResultsImporter(settings={"units": {"energy": "eV", "forces": "eV/angstrom
↳"}})
ri.add_trajectory_singlepoints(
    replay_dft_job,
    properties=["energy", "forces"],
    indices=[0, 1, 2, 3, 4],
    data_set="training_set",
)
ri.add_trajectory_singlepoints(replay_dft_job, properties=["energy", "forces"],
↳indices=[5], data_set="validation_set")
yaml_dir = "lidiffusion_initial_reference_data"
ri.store(yaml_dir, backup=False)

# ## Preliminary active learning jobs with equilibrium MD
#
# Let's first run some simple NVT MD with active learning for both the
# * stoichiometric system (``supercell``), and
# * defective system (``defective_1``)
#
# This is just to get a good general sampling before setting up the reaction boost to
↳explicitly sample the diffusion process.

# ### Active Learning for stoichiometric system
#
# The stoichiometric system is a perfect crystal with not so many atoms. The forces
↳will be very close to 0 also after a few MD frames. However, even if the ML model
↳predicts forces close to 0, the R^2 between predicted and reference forces may be
↳quite low.
#
# In a case like this, it is reasonable to either

```

(continues on next page)

(continued from previous page)

```

#
# * decrease the MinR2 success criterion, and/or
# * perturb the atomic coordinates of the initial structure a bit, so that the forces
↪aren't extremely small
#
# Here, we do both of these:

nvt_md_s = plams.AMSNVTJob(nsteps=5000, timestep=0.5, temperature=400, tau=50,
↪thermostat="Berendsen").settings

prelim_al_s = plams.Settings()
prelim_al_s.input.ams.ActiveLearning.InitialReferenceData.Load.Directory = os.path.
↪abspath(yaml_dir)
prelim_al_s.input.ams.ActiveLearning.Steps.Type = "Geometric"
prelim_al_s.input.ams.ActiveLearning.Steps.Geometric.NumSteps = 7
prelim_al_s.input.ams.ActiveLearning.SuccessCriteria.Forces.MinR2 = 0.5
prelim_al_s.input.ams.ActiveLearning.AtEnd.RerunSimulation = "No"

ml_s = plams.Settings()
ml_s.input.ams.MachineLearning.Backend = "M3GNet"
ml_s.input.ams.MachineLearning.M3GNet.Model = "UniversalPotential"
ml_s.input.ams.MachineLearning.MaxEpochs = 100
ml_s.input.ams.MachineLearning.Target.Forces.MAE = 0.04

perturbed_supercell = supercell.copy()
perturbed_supercell.perturb_atoms(0.1)

prelim_al_job = SimpleActiveLearningJob(
    name="al_supercell",
    molecule=perturbed_supercell,
    settings=prelim_al_s + ml_s + nvt_md_s + dft_s,
)
prelim_al_job.run(watch=True)

# ### Active learning for defective system

prelim_al_s2 = prelim_al_s.copy()
prelim_al_s2.input.ams.ActiveLearning.InitialReferenceData.Load.Directory = (
    prelim_al_job.results.get_reference_data_directory()
)
prelim_al_s2.input.ams.ActiveLearning.InitialReferenceData.Load.FromPreviousModel =
↪"No"
prelim_al_s2.input.ams.ActiveLearning.SuccessCriteria.Forces.MaxDeviationForZeroForce
↪= 0.35

ml_s2 = plams.Settings()
ml_s2.input.ams.MachineLearning.Backend = "M3GNet"
ml_s2.input.ams.MachineLearning.LoadModel = prelim_al_job.results.get_params_results_
↪directory()
ml_s2.input.ams.MachineLearning.MaxEpochs = 200
ml_s2.input.ams.MachineLearning.Target.Forces.MAE = 0.03

prelim_al_job2 = SimpleActiveLearningJob(
    name="al_defective", molecule=defective_1, settings=prelim_al_s2 + ml_s2 + nvt_md_
↪s + dft_s
)

```

(continues on next page)

(continued from previous page)

```

prelim_al_job2.run(watch=True)

# ## Set up ReactionBoost MD simulation with M3GNet-UP-2022
#
# ReactionBoost simulations can be a bit tricky to set up.
#
# Before using ReactionBoost inside the Active Learning, it is best to verify that
# the simulation behaves reasonably when using M3GNet-UP-2022.
#
# So let's just run a normal ReactionBoost simulation with M3GNet-UP-2022:

md_s = plams.Settings()
md_s.Thermostat.Temperature = 300
md_s.Thermostat.Type = "Berendsen"
md_s.Thermostat.Tau = 5
md_s.InitialVelocities.Type = "Random"
md_s.InitialVelocities.Temperature = 300
md_s.ReactionBoost.Type = "RMSD"
md_s.ReactionBoost.NSteps = 480
md_s.ReactionBoost.Region = "DiffusingLi"
md_s.ReactionBoost.TargetSystem = "final"
md_s.ReactionBoost.RMSDRestraint.Type = "Harmonic"
md_s.ReactionBoost.RMSDRestraint.Harmonic.ForceConstant = 0.1
md_s.NSteps = 500
md_s.TimeStep = 0.5
md_s.Trajectory.SamplingFreq = 20
rb_s = plams.Settings()
rb_s.input.ams.Task = "MolecularDynamics"
rb_s.input.ams.MolecularDynamics = md_s

m3gnet_rb_job = plams.AMSJob(
    settings=m3gnet_up_s + rb_s,
    name="m3gnet_up_rb",
    molecule={"": defective_1, "final": defective_2},
)
m3gnet_rb_job.run()

N = 5 # show 5 structures
fig, axes = plt.subplots(1, N, figsize=(12, 5))
nEntries = m3gnet_rb_job.results.readrkf("History", "nEntries")
print(f"There are {nEntries} frames in the trajectory")
ind = np.linspace(1, nEntries, N, endpoint=True, dtype=np.int64)
for ax, i_frame in zip(axes, ind):
    plams.plot_molecule(m3gnet_rb_job.results.get_history_molecule(i_frame), ax=ax,
    rotation="-85x, 5y, 0z")
    ax.set_title(f"Frame {i_frame}")

# It looks like the Li atom is diffusing to the correct place, but it is easiest to
# visualize in AMSmovie:

# **Conclusion**: We have reasonable reaction boost settings for M3GNet-UP-2022. This
# does not guarantee that the settings will be appropriate with the retrained model

```

(continues on next page)

(continued from previous page)

```

→ (for example because we expect the barrier to be higher), but it is likely to work.

# ## Run simple active learning using ReactionBoost MD

al_s = plams.Settings()
al_s.input.ams.ActiveLearning.SuccessCriteria.Energy.Relative = 0.0012
al_s.input.ams.ActiveLearning.SuccessCriteria.Forces.MaxDeviationForZeroForce = 0.25
al_s.input.ams.ActiveLearning.Steps.Type = "Linear"
al_s.input.ams.ActiveLearning.Steps.Linear.Start = 15
al_s.input.ams.ActiveLearning.Steps.Linear.StepSize = 50
al_s.input.ams.ActiveLearning.InitialReferenceData.Load.Directory = (
    prelim_al_job2.results.get_reference_data_directory()
)
al_s.input.ams.ActiveLearning.InitialReferenceData.Load.FromPreviousModel = "No"

ml_s = plams.Settings()
ml_s.input.ams.MachineLearning.Backend = "M3GNet"
ml_s.input.ams.MachineLearning.LoadModel = prelim_al_job2.results.get_params_results_
→directory()
ml_s.input.ams.MachineLearning.MaxEpochs = 200
ml_s.input.ams.MachineLearning.Target.Forces.MAE = 0.03

sal_job = SimpleActiveLearningJob(
    name="sal_lidiffusion_rb",
    settings=al_s + dft_s + ml_s + rb_s,
    molecule={"": defective_1, "final": defective_2},
)
print(sal_job.get_input())

# plams.config.jobmanager.hashing = None
sal_job.run(watch=True)

m3gnet_new_s = sal_job.results.get_production_engine_settings()
m3gnet_new_s = prelim_al_job2.results.get_production_engine_settings()

# ## Run NEB with retrained M3GNet

# constraint_s = plams.Settings()
# constraint_s.input.ams.Constraints.Atom = 8
defective_1_perturbed = defective_1.copy()
defective_1_perturbed.perturb_atoms()
defective_2_perturbed = defective_2.copy()
defective_2_perturbed.perturb_atoms()
m3gnet_new_neb_job = plams.AMSJob(
    settings=m3gnet_new_s + neb_s, # + constraint_s,
    molecule={"": defective_1, "final": defective_2},
    name="m3gnet_new_neb",
)
m3gnet_new_neb_job.run()

m3gnet_new_res = m3gnet_new_neb_job.results.get_neb_results(unit="eV")

```

(continues on next page)

(continued from previous page)

```

print(f"Left barrier: {m3gnet_new_res['LeftBarrier']:.3f} eV")
print(f"Right barrier: {m3gnet_new_res['RightBarrier']:.3f} eV")

# ref_results = [0.0, 0.144, 0.482, 0.816, 0.963, 0.816, 0.482, 0.144, 0.0] # from
↳ Quantum ESPRESSO DFT calculation
nImages = m3gnet_new_res["nImages"]
relative_energies = np.array(m3gnet_new_res["Energies"]) - m3gnet_new_res["Energies
↳"][0]
plt.plot(relative_energies)
plt.xlabel("Image")
plt.ylabel("Relative energy (eV)")
plt.title("M3GNet retrained NEB Li diffusion")

fig, axes = plt.subplots(1, len(m3gnet_new_res["Molecules"]), figsize=(15, 5))

for i, mol in enumerate(m3gnet_new_res["Molecules"]):
    plams.plot_molecule(mol, ax=axes[i], rotation="-85x,-5y,0z")
    axes[i].set_title(f"Image {i}")

# Let's run DFT calculations on these frames to compare. We can do this using the
↳ "Replay" task.
#
# Note: The DFT calculations may take a few minutes to complete.

replay_s = plams.Settings()
replay_s.input.ams.Task = "Replay"
replay_s.input.ams.Replay.File = os.path.abspath(m3gnet_new_neb_job.results.rkfpath())
replay_s.input.ams.Properties.Gradients = "Yes"

replay_dft_job = plams.AMSJob(settings=replay_s + dft_s, name="replay_m3gnet_new_neb_
↳ with_dft")
replay_dft_job.run(watch=True)

# When using "Replay" on a NEB job, the results are stored in the normal NEB format.
↳ Thus we can use the ``.get_neb_results()`` method also on this job:

dft_energies = replay_dft_job.results.get_neb_results(unit="eV")["Energies"]
dft_relative_energies = np.array(dft_energies) - dft_energies[0]

plt.plot(relative_energies)
plt.plot(dft_relative_energies)
plt.legend(["M3GNet retrained", "DFT singlepoints"])
plt.title("Compare retrained M3GNet NEB to DFT singlepoints")

```

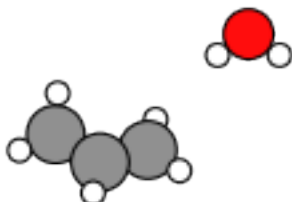
4.5.9 Active Learning with uncertainties predicted from committee models

Trained model: A committee model consisting of 2 custom M3GNet models, starting from random weight initialization

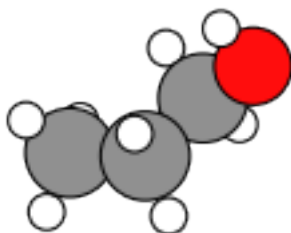
Reference method: DFT, PBE-D3(BJ), engine ADF

System: $\text{H}_2\text{O} + \text{propene} \rightarrow \text{1-propanol}$

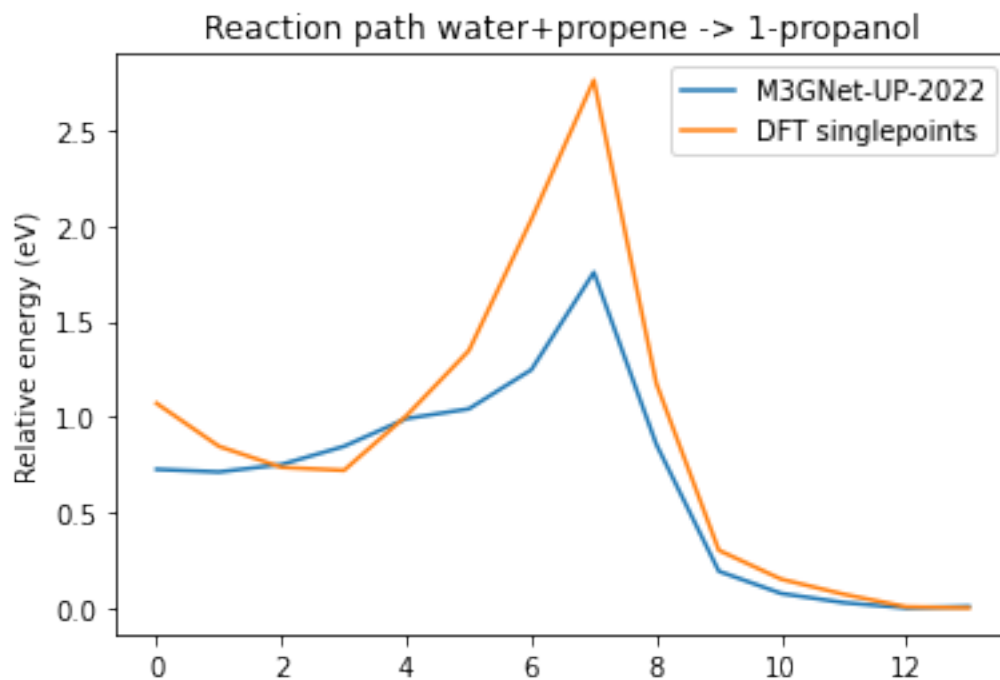
System 1: propene + H₂O



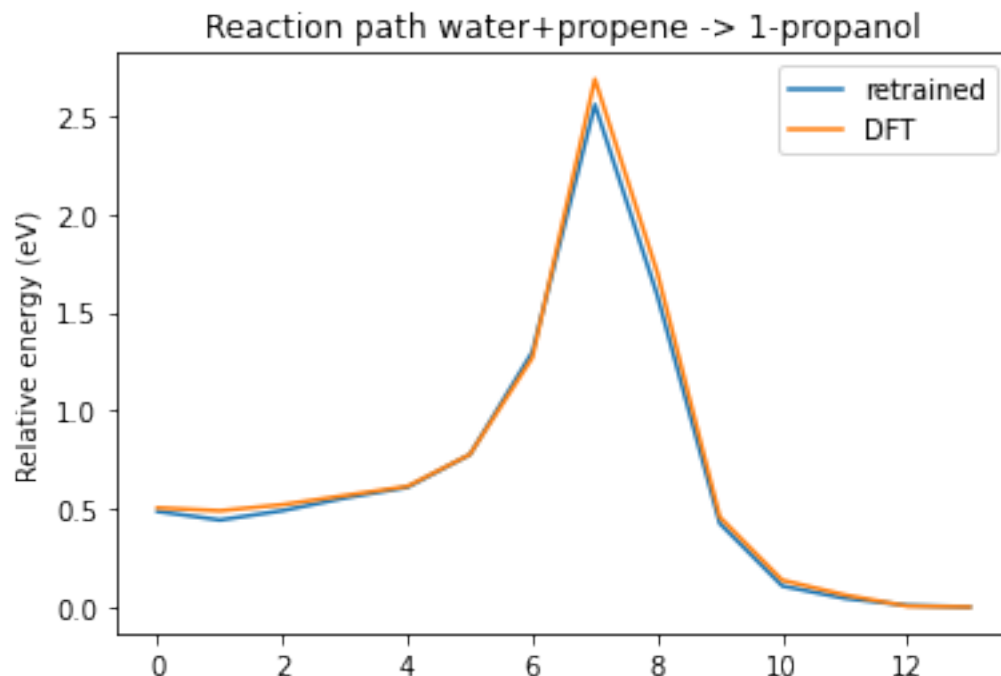
System 2: 1-propanol



Problem: M3GNet-UP-2022 underestimates the reaction barrier



Solution: Retraining the model gives better agreement:



Expected duration: This notebook takes about 48 hours to run on the CPU. This includes both the active learning and the various NEB calculations.

Approach: In addition to dividing an MD simulation into several active learning steps, we will also exploit the uncertainty that ML models can give when trained as committee models. By setting up a ReactionBoost MD simulation to force the desired reaction and applying uncertainty-based exit conditions, we effectively choose training points where the model is “uncertain”.

Key takeaways:

1. Training M3GNet models from scratch typically requires more training data than starting from the Universal Potential
2. ReactionBoost can be effectively used to sample structures during a chemical reaction but may not sample the minimum energy path.
3. In particular for models trained from scratch like this, to get a good result for the reaction barrier we also need to create a separate “mini-active-learning-loop” in Python that runs NEB, recalculates with DFT, retrains the model, runs NEB, etc., a few times before we get a good result also for NEB calculations.

Technical note: This example uses the new “Python Input System for AMS” input classes instead of the normal PLAMS settings object. Both methods work.

To follow along, either

- Download `sal_uncertainty.py` (run as `$AMSBIN/amsipython sal_uncertainty.py`).
- Download `sal_uncertainty.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports

import scm.plams as plams
from scm.simple_active_learning import SimpleActiveLearningJob
from scm.reactmap.tools import reorder_plams_mol
import matplotlib.pyplot as plt
from scm.input_classes import drivers, engines
from scm.libbase import Units
import scm.params as params
import os
import numpy as np

plams.init()

system1 = plams.from_smiles("CC=C", forcefield="uff") # propene

# add a water molecule *at least* 1 angstrom away from all propene atoms
system1.add_molecule(plams.from_smiles("O"), margin=1)

for at in system1:
    at.properties = plams.Settings()
system1.delete_all_bonds()

plams.plot_molecule(system1)
plt.title("System 1: propene + H2O")

system2 = plams.from_smiles("CCCO") # 1-propanol
for at in system2:
    at.properties = plams.Settings()
system2.delete_all_bonds()

# reorder atoms in system2 to match the order in system1
# this only takes bond breaking and forming into account, the order is not guaranteed.
# → to match exactly for all atoms
system2 = reorder_plams_mol(system1, system2)

# Rotate system2 so that the RMSD with respect to system1 is minimized
system2.align2mol(system1)

plams.plot_molecule(system2)
plt.title("System 2: 1-propanol")

# sanity-check that at least the order of elements is identical
assert list(system1.symbols) == list(system2.symbols), f"Something went wrong!"

# Note that this does not guarantee that the atom order is completely the same.
# For example the order of the hydrogen atoms in the CH3 group might be different.
```

(continues on next page)

(continued from previous page)

```

# This means that we cannot just run NEB directly. So let's first run MD_
↳ReactionBoost.

# ## Initial Reaction Boost to get reactant and product

# ### Engine settings
#
# Here we use ``e_up`` to refer to the M3GNet Universal Potential.
#
# For the ADF DFT engine we set an electronic temperature and the OptimizeSpinRound_
↳option. This helps with SCF convergence, and can converge the SCF to a different_
↳spin state when applicable.

e_up = engines.MLPotential()
e_up.Model = "M3GNet-UP-2022"

e_dft = engines.ADF()
e_dft.XC.GGA = "PBE"
e_dft.XC.Dispersion = "GRIMME3 BJDAMP"
e_dft.Basis.Type = "TZP"
e_dft.Unrestricted = True
e_dft.Occupations = "ElectronicTemperature=300 OptimizeSpinRound=0.05"

def set_reaction_boost(driver, nsteps=3000):
    driver.Task = "MolecularDynamics"
    md = driver.MolecularDynamics
    md.InitialVelocities.Temperature = 100
    md.NSteps = nsteps
    md.ReactionBoost.Type = "Pair"
    md.ReactionBoost.BondBreakingRestrains.Type = "Erf"
    md.ReactionBoost.BondBreakingRestrains.Erf.MaxForce = 0.05
    md.ReactionBoost.BondMakingRestrains.Type = "Erf"
    md.ReactionBoost.BondMakingRestrains.Erf.MaxForce = 0.12
    md.ReactionBoost.InitialFraction = 0.05
    md.ReactionBoost.Change = "LogForce"
    md.ReactionBoost.NSteps = nsteps
    md.ReactionBoost.TargetSystem[0] = "final"
    md.Trajectory.SamplingFreq = 10
    md.Trajectory.WriteBonds = False
    md.Trajectory.WriteMolecules = False
    md.TimeStep = 0.25
    md.Thermostat[0].Tau = 5
    md.Thermostat[0].Temperature = [100.0]
    md.Thermostat[0].Type = "Berendsen"

def get_reaction_boost_job(engine, molecule, name: str = "reaction_boost") -> plams.
↳AMSJob:
    d = drivers.AMS()
    set_reaction_boost(d)
    d.Engine = engine
    job = plams.AMSJob(settings=d, name=name, molecule=molecule)
    job.settings.runscript.nproc = 1
    return job

```

(continues on next page)

(continued from previous page)

```

molecule_dict = {"": system1, "final": system2}
prelim_job = get_reaction_boost_job(e_up, molecule_dict, "prelim_md")

prelim_job.run()

# Let's check that the final molecule corresponds to the target system (1-propanol):
#

engine_energies = prelim_job.results.get_history_property("EngineEnergy")
N_frames = len(engine_energies)
max_index = np.argmax(engine_energies)
reactant_index = max(0, max_index - 50) # zero-based
system1_correct_order = prelim_job.results.get_history_molecule(reactant_index + 1)
system1_correct_order.delete_all_bonds()
plams.plot_molecule(system1_correct_order)

product_index = min(max_index + 50, N_frames - 1) # zero-based
system2_correct_order = prelim_job.results.get_history_molecule(product_index + 1)
system2_correct_order.delete_all_bonds()
plams.plot_molecule(system2_correct_order)

# We now have the product molecule with the correct atom order, which means we can
→run an initial NEB with M3GNet and compare to the DFT reference:

# ## Initial NEB calculation

molecule_dict = {"": system1_correct_order, "final": system2_correct_order}

def get_neb_job(engine, name: str = "neb") -> plams.AMSJob:
    d = drivers.AMS()
    d.Task = "NEB"
    d.GeometryOptimization.Convergence.Quality = "Basic"
    d.NEB.Images = 12
    d.Engine = engine

    neb_job = plams.AMSJob(name=name, settings=d, molecule=molecule_dict)
    return neb_job

neb_job = get_neb_job(e_up, name="neb_up")
neb_job.run()

# Let's then replay with the ADF DFT engine.

def get_replay_job(rkf, name="replay"):
    d_replay = drivers.AMS()
    d_replay.Task = "Replay"
    d_replay.Replay.File = os.path.abspath(rkf)
    d_replay.Properties.Gradients = True

```

(continues on next page)

(continued from previous page)

```

d_replay.Engine = e_dft

replay_job = plams.AMSJob(name=name, settings=d_replay)
return replay_job

replay_job = get_replay_job(neb_job.results.rkfpath(), "replay_neb")
replay_job.run()

def get_relative_energies(neb_job):
    e = neb_job.results.get_neb_results()["Energies"]
    e = np.array(e) - np.min(e)
    e *= Units.get_factor("hartree", "eV")
    return e

def plot_neb_comparison(neb_job, replay_job, legend=None, title=None):
    energies_up = get_relative_energies(neb_job)
    energies_dft = get_relative_energies(replay_job)
    fig, ax = plt.subplots()
    ax.plot(energies_up)
    ax.plot(energies_dft)
    ax.legend(legend or ["M3GNet-UP-2022", "DFT singlepoints"])
    ax.set_ylabel("Relative energy (eV)")
    ax.set_title(title or "Reaction path water+propene -> 1-propanol")
    return ax

plot_neb_comparison(neb_job, replay_job)

# So we can see that either M3GNet-UP-2022 underestimates the barrier or it NEB path
# is different from the DFT one. Let's use these datapoints as a starting point for
# the active learning.
#
# Let's also run replay on some of the frames from the prelim_md reaction boost job:

replay_md = get_replay_job(prelim_job.results.rkfpath(), "replay_md")
N_frames_to_replay = 10
replay_md.settings.input.Replay.Frames = list(
    np.linspace(reactant_index, product_index, N_frames_to_replay, dtype=np.int64)
)
replay_md.run()

# ## Simple Active Learning using Uncertainties

# ## Create the initial reference data

yaml_dir = "my_neb_data"
ri = params.ResultsImporter.from_ase() # use ASE units
ri.add_trajectory_singlepoints(replay_job.results.rkfpath(), properties=["energy",
    "forces"], data_set="training_set")
ri.add_trajectory_singlepoints(
    replay_md.results.rkfpath(),
    properties=["energy", "forces"],

```

(continues on next page)

(continued from previous page)

```

    data_set="training_set",
    indices=list(range(1, N_frames_to_replay - 1)),
)
ri.add_trajectory_singlepoints(
    replay_md.results.rkfpath(),
    properties=["energy", "forces"],
    indices=[0, N_frames_to_replay - 1],
    data_set="validation_set",
)
ri.store(yaml_dir, backup=False)

# When we have initial reference data like this, it's often most convenient to run a
↳ separate ParAMS training before starting the active learning.
#
# This lets us sanity-check the training parameters, and more easily try different
↳ Active Learning settings without having to retrain the initial model every time.

def get_params_job(yaml_dir, load_model=None, name="paramsjob"):
    committee_size = 2
    paramsjob = params.ParAMSJob.from_yaml(yaml_dir, use_relative_paths=True,
↳ name=name)
    paramsjob.settings.input.Task = "MachineLearning"
    ml = paramsjob.settings.input.MachineLearning
    ml.Backend = "M3GNet"
    if load_model:
        ml.LoadModel = load_model
        ml.MaxEpochs = 200
        ml.M3GNet.LearningRate = 5e-4
    else:
        ml.M3GNet.Model = "Custom"
        ml.M3GNet.Custom.NumNeurons = 32
        ml.MaxEpochs = 300
        ml.M3GNet.LearningRate = 1e-3

    ml.CommitteeSize = committee_size
    paramsjob.settings.input.ParallelLevels.CommitteeMembers = committee_size

    return paramsjob

paramsjob = get_params_job(yaml_dir, name="custom_initial_training")
paramsjob.run()

# ## Set up the active learning job
#
# Here the key new setting is the ``ReasonableSimulationCriteria.
↳ GradientsUncertainty``. This setting will cause the MD simulation to instantly stop
↳ if the uncertainty is greater than 1.0 eV/angstrom.
#
# This is useful since the ML model is unlikely to give good predictions for the new
↳ types of structures encountered during the reactive MD.
#
# In the summary log, such an event will be marked as "FAILED" with the reason
↳ "GRADIENTS_UNCERTAINTY".

```

(continues on next page)

(continued from previous page)

```

#
# In order to use ML uncertainties, you need to train a committee model with at least
# 2 members. Here we set the committee size to 2. We also choose to train the 2
# committee members in parallel. By default, they would be trained in sequence.
#
# It is a good idea to train them in parallel if you have the computational resources
# to do so (for example, enough GPU memory).
#
# When using uncertainty-based criteria, you may consider increasing the
# MaxAttemptsPerStep. Here, we stick with the default value of 15.

d_al = drivers.SimpleActiveLearning()
d_al.ActiveLearning.InitialReferenceData.Load.FromPreviousModel = True
d_al.ActiveLearning.Steps.Type = "Geometric"
d_al.ActiveLearning.Steps.Geometric.NumSteps = 5
d_al.ActiveLearning.Steps.Geometric.Start = 10
d_al.ActiveLearning.ReasonableSimulationCriteria.GradientsUncertainty.Enabled = True
d_al.ActiveLearning.ReasonableSimulationCriteria.GradientsUncertainty.MaxValue = 1.0
# eV/ang
d_al.ActiveLearning.SuccessCriteria.Forces.MinR2 = 0.4
d_al.ActiveLearning.MaxReferenceCalculationsPerAttempt = 3
d_al.ActiveLearning.MaxAttemptsPerStep = 15
d_al.MachineLearning.Backend = "M3GNet"
d_al.MachineLearning.LoadModel = os.path.abspath(params.job.results.path)
d_al.MachineLearning.CommitteeSize = 2
d_al.MachineLearning.MaxEpochs = 120
d_al.MachineLearning.M3GNet.LearningRate = 5e-4
d_al.MachineLearning.RunAMSEAtEnd = False
d_al.ParallelLevels.CommitteeMembers = 2
set_reaction_boost(d_al)
d_al.Engine = e_dft

sal_job = SimpleActiveLearningJob(name="sal", driver=d_al, molecule=molecule_dict)
print(sal_job.get_input())

sal_job.run(watch=True)

# Above we see that during step 5, several attempts failed with the message GRADIENTS_
# UNCERTAINTY. It is during step 5 that the actual reaction happens. We do not know
# exactly at what time the reaction will happen (since the ReactionBoost gradually
# increases the applied force).
#
# In such a case it is useful to have the GradientsUncertainty reasonable simulation
# criterion. This will immediately stop the simulation when the uncertainty is too
# high and follow it with a retraining of the model.

# ## New NEB validation

# Let's now evaluate with a second-round NEB and replay. ``sal_job.results.get_
# production_engine_settings()`` returns the engine settings in the PLAMS Settings
# format. Let's first convert it to a PISA Engine:

def settings2engine(settings):

```

(continues on next page)

(continued from previous page)

```

temporary_d = drivers.AMS.from_settings(settings)
return temporary_d.Engine

e_new = settings2engine(sal_job.results.get_production_engine_settings())

# Let's now create our own active learning loop for NEB where we run the NEB_
↪ calculation with our trained potential, replay, add points to training set, retrain,
↪ rerun NEB etc.
#
# We need to do this since MD SAL may not exactly sample the minimum energy path.

n_loop = 3
ri = params.ResultsImporter.from_yaml(sal_job.results.get_reference_data_directory())
params_results_dir = sal_job.results.get_params_results_directory()
for i in range(n_loop):
    neb = get_neb_job(e_new, f"new_neb{i}")
    neb.run()
    replay = get_replay_job(neb.results.rkfpath(), name=f"new_replay{i}")
    replay.run()
    plot_neb_comparison(neb, replay, legend=["retrained", "DFT"])
    yaml_dir = f"new_yaml_dir{i}"
    ri.add_trajectory_singlepoints(replay.results.rkfpath(), properties=["energy",
↪ "forces"])
    ri.store(yaml_dir, backup=False)
    paramsjob = get_params_job(yaml_dir, load_model=params_results_dir, name=f"new_
↪ params{i}")
    paramsjob.run()
    e_new = settings2engine(paramsjob.results.get_production_engine_settings())
    params_results_dir = os.path.abspath(paramsjob.results.path)

```

Ru/H case study

4.5.10 Ru/H introduction

Trained model: M3GNet, starting from the Universal Potential (UP)

Reference method: PBE-D3(BJ) with engine Quantum ESPRESSO

System: H atoms depositing onto Ru surfaces

Problem: M3GNet-UP-2022 is not very reliable for high-temperature surface chemistry.

Solution: Retraining the model gives better agreement.

Expected duration: This example takes several days to run on a modern compute node.

This is a very thorough example which shows how to

- construct initial reference data using PES Scans like volume scans, cartesian coordinate scans, and bond scans as well as MD simulations
- training an initial model to the reference data before the active learning loop
- running an active learning loop with the molecule gun

Important: This tutorial is only compatible with AMS2024.102 or later. AMS2024.101 will not work.

To run this example, you may download `common_ru_h.py`, `01_Ru_volume_scan_H2_bond_scan.py`, `02_surface_pes_scans.py`, `03_Ru_H2_gas_snapshots.py`, `04_Ru_H_initial_training.py`, `05_active_learning_molecule_gun_md.py` and run

```
#!/bin/sh

# also make sure that common_ru_h.py is in the current directory

"$AMSBIN/amspython" 01_Ru_volume_scan_H2_bond_scan.py || exit 1
"$AMSBIN/amspython" 02_surface_pes_scans.py || exit 1
"$AMSBIN/amspython" 03_Ru_H2_gas_snapshots.py || exit 1
"$AMSBIN/amspython" 04_Ru_H_initial_training.py || exit 1
"$AMSBIN/amspython" 05_active_learning_molecule_gun_md.py || exit 1
```

Important: The `common_ru_h.py` file contains a variable `TESTING_MODE`.

Set `TESTING_MODE = True` to not use DFT reference calculations but instead a custom-trained M3GNet model for the reference calculations. This will let you run through the workflow quickly without running any expensive DFT reference calculations.

Alternatively, you can follow the individual Jupyter notebooks below:

- *Ru/H Part 1: Initial reference data from lattice optimization, volume scan, bond scan* (page 162)
- *Ru/H Part 2: Initial reference data from cartesian coordinate scans and bond scans* (page 165)
- *Ru/H Part 3: Initial reference data MD simulation + single-point replays* (page 169)
- *Ru/H Part 4: Initial training* (page 171)
- *Ru/H Part 5: Active learning for molecule gun MD* (page 173)

4.5.11 Ru/H Part 1: Initial reference data from lattice optimization, volume scan, bond scan

Important: First read *Ru/H introduction* (page 161) and follow all parts in order.

To follow along,

- Download `01_Ru_volume_scan_H2_bond_scan.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```
#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports

import scm.plams as plams
from scm.params import ResultsImporter
from scm.plams import Settings, AMSJob, log, Molecule

# common_ru_h.py must exist in the current working directory
from common_ru_h import (
    rotation,
    dft_settings,
    QEKPointsConfig,
    lattice_optimization_settings,
    plot_pesscan,
    check_installation,
)

check_installation()

# ## Initialize PLAMS working directory

plams.init()

# ## Bulk structure: hcp Ru

initial_bulk = plams.Molecule()
a = 2.7 # hexagonal lattice parameter, angstrom
c = 4.2768 # hexagonal lattice parameter, angstrom
initial_bulk.add_atom(plams.Atom(symbol="Ru", coords=(0.0, 0.0, 0.0)))
initial_bulk.add_atom(plams.Atom(symbol="Ru", coords=(0.0, a / 3**0.5, c / 2)))
initial_bulk.lattice = [[a, 0, 0], [-a / 2, a * 3**0.5 / 2, 0], [0, 0, c]]
log("Initial structure")
log(initial_bulk)

plams.plot_molecule(initial_bulk, rotation=rotation)

# ## Lattice optimization of bulk Ru with DFT

lattopt_job = plams.AMSJob(
    settings=dft_settings(QEKPointsConfig(11, 11, 11)) + lattice_optimization_
    ↪ settings(),
    name="hcp_lattopt_Ru_dft",
    molecule=initial_bulk,
)
lattopt_job.run()

optimized_bulk: Molecule = lattopt_job.results.get_main_molecule() # type: ignore
log(optimized_bulk)
```

(continues on next page)

(continued from previous page)

```
log(f"Density: {optimized_bulk.get_density():.2f} kg/m^3")

# ## Volume scan of bulk Ru with DFT

from common_ru_h import (
    dft_settings,
    QEKPointsConfig,
    pesscan_settings,
    CellVolumeScalingRangeScanCoordinate,
)

s = dft_settings(QEKPointsConfig(11, 11, 11))
s += pesscan_settings([CellVolumeScalingRangeScanCoordinate(0.85, 1.15)], n_points=7)
volume_scan_job = AMSJob(
    settings=s,
    molecule=optimized_bulk,
    name="bulk_hcp_Ru_volume_scan_dft",
)
volume_scan_job.run()

plot_pesscan(volume_scan_job)

# ## Bond scan of H2 with DFT

h2_mol = plams.from_smiles("[HH]")
h2_mol.lattice = [[5, 0, 0], [0, 5, 0], [0, 0, 5]]
plams.plot_molecule(h2_mol, rotation=rotation)

from common_ru_h import (
    dft_settings,
    QEKPointsConfig,
    pesscan_settings,
    DistanceScanCoordinate,
)

scan_coordinate = DistanceScanCoordinate(atom1=1, atom2=2, start=0.55, end=1.0)
s = dft_settings(QEKPointsConfig(1, 1, 1))
s += pesscan_settings([scan_coordinate], n_points=7)

h2_bond_scan_job = AMSJob(settings=s, molecule=h2_mol, name="h2_bond_scan_dft")
h2_bond_scan_job.run()

plot_pesscan(h2_bond_scan_job)

# ## Store results

ri = ResultsImporter.from_ase()
properties = ["energy", "forces"]
ri.add_pesscan_singlepoints(volume_scan_job, properties=properties)
ri.add_pesscan_singlepoints(h2_bond_scan_job, properties=properties)
```

(continues on next page)

(continued from previous page)

```

ri.add_singlejob(lattopt_job, task="SinglePoint", properties=properties)

# Also add as PES Scans - these will not be used during training but
# will plot the energy-volume curve and bond-scan curve at the end
# of the training
ri.add_singlejob(volume_scan_job, task="PESScan", properties=["pes"])
ri.add_singlejob(h2_bond_scan_job, task="PESScan", properties=["pes"])

ri.store("reference_data_1")

```

4.5.12 Ru/H Part 2: Initial reference data from cartesian coordinate scans and bond scans

Important: First read *Ru/H introduction* (page 161) and follow all parts in order.

To follow along,

- Download `02_surface_pes_scans.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```

#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports

import scm.plams as plams
from scm.params import ResultsImporter
from scm.plams import Settings, AMSJob, log, Molecule
from pathlib import Path
import matplotlib.pyplot as plt

# common_ru_h.py must exist in the current working directory
from common_ru_h import rotation, check_installation

# ## Initialize PLAMS working directory

old_ref_dir = "reference_data_1"
check_installation(old_ref_dir)
new_ref_dir = "reference_data_2"
ri = ResultsImporter.from_yaml(old_ref_dir)
plams.init()

# ## Load the optimized bulk Ru structure from the job collection
#
# The lattice was optimized in the previous notebook, and the structure was stored in
# the job collection.
#
# Let's retrieve it from the job collection and use it to construct Ru surface slabs.

```

(continues on next page)

(continued from previous page)

```

optimized_bulk = ri.job_collection["hcp_lattopt_Ru_dft"].molecule
plams.plot_molecule(optimized_bulk, rotation=rotation)
plt.title("Bulk hcp Ru")

# ## Cut out Ru(0001) and Ru(10-10) slabs
#
# The PLAMS Molecule class does not have any methods for creating slabs. However, we
→ can use the UnifiedChemicalSystem in AMS2024+. The details can be found in the
→ ``slice_slab()`` function in common_ru_h.py.

from common_ru_h import slice_slab

slab_100 = slice_slab(
    optimized_bulk,
    miller=(1, 0, 0),
    thickness=7.0,
    cell_z=15,
    ref_atom=0,
)
slab_100 = slab_100.supercell(3, 2, 1)
for at in slab_100:
    at.properties = Settings() # remove details about supercell generation
plams.plot_molecule(slab_100, rotation=rotation)
plt.title("Ru(10-10)")

from common_ru_h import slice_slab

slab_001 = slice_slab(
    optimized_bulk,
    miller=(0, 0, 1),
    thickness=7.0,
    cell_z=15,
    ref_atom=0,
)
slab_001 = slab_001.supercell([[3, 0, 0], [2, 4, 0], [0, 0, 1]])
for at in slab_001:
    at.properties = Settings() # remove details about supercell generation
plams.plot_molecule(slab_001, rotation=rotation)
plt.title("Ru(0001)")

# This slab is perhaps too thick, so let's remove the top atomic layer:

for at in slab_001:
    if at.coords[2] > 10:
        slab_001.delete_atom(at)

plams.plot_molecule(slab_001, rotation=rotation)
plt.title("Ru(0001)")

# ## Add hydrogens

from common_ru_h import add_adsorbate

```

(continues on next page)

(continued from previous page)

```

slab_100_H = add_adsorbate(slab_100, "H", frac_x=0, frac_y=0, delta_z=2.0)
# add second adsorbate, now delta_z=2.0 means 2 angstrom above the previous H
slab_100_H2 = add_adsorbate(slab_100_H, "H", frac_x=0.5, frac_y=0.5, delta_z=2.0)
plams.plot_molecule(slab_100_H2, rotation=rotation, radii=0.6)
plt.title("Ru(10-10) with H")

from common_ru_h import add_adsorbate

slab_001_H = add_adsorbate(slab_001, "H", frac_x=0, frac_y=0, delta_z=2.0)
slab_001_H2 = add_adsorbate(slab_001_H, "H", frac_x=0.5, frac_y=0.5, delta_z=2.0)
plams.plot_molecule(slab_001_H2, rotation=rotation, radii=0.6)
plt.title("Ru(0001) with H")

# ## Set up and run the jobs

# ### Ru-H distance bond scan on Ru(10-10)

from common_ru_h import get_surface_bond_scan_coordinates
from common_ru_h import (
    get_bottom_atom_index,
    pesscan_settings,
    m3gnet_up_settings,
    constraints_settings,
    plot_pesscan,
)

system = slab_100_H

scan_coordinates = get_surface_bond_scan_coordinates(system, atom_index=len(system),
    ↪start=1.2, end=2.5)
pesscan_Ru_H_distance = plams.AMSJob(
    settings=(
        pesscan_settings(scan_coordinates, n_points=6)
        + m3gnet_up_settings()
        + constraints_settings(get_bottom_atom_index(system))
    ),
    name="PESScan_Ru_H_distance_m3gnet",
    molecule=system,
)
pesscan_Ru_H_distance.run()
# !amsmovie {pesscan_Ru_H_distance.results.rkfpath()} # open in AMSmovie
plot_pesscan(pesscan_Ru_H_distance)

# ### Surface and bulk diffusion H on/in Ru(10-10)
#
# Note: These PES scans are just meant to sample H atoms diffusing through the slab_
    ↪and on the surface.
#
# You can also set up more intelligent PES scans (for example, to sample diffusion_
    ↪along specific pathways like hcp-bridge-fcc etc.). In these examples, the H atom_
    ↪simply diffuses from one corner of the cell to the opposite corner.

from common_ru_h import (
    get_surface_diffusion_scan_coordinates,

```

(continues on next page)

(continued from previous page)

```

        get_bulk_diffusion_scan_coordinates,
    )
    from common_ru_h import (
        get_bottom_atom_index,
        pesscan_settings,
        m3gnet_up_settings,
        constraints_settings,
        plot_pesscan,
    )

    system = slab_100_H2

    scan_coordinates = get_surface_diffusion_scan_coordinates(system, atom_
    ↪index=len(system) - 1)
    scan_coordinates += get_bulk_diffusion_scan_coordinates(system, atom_
    ↪index=len(system), delta_z=2.0)

    pesscan_100_H2 = plams.AMSJob(
        settings=(
            pesscan_settings(scan_coordinates, n_points=20)
            + m3gnet_up_settings()
            + constraints_settings(get_bottom_atom_index(system))
        ),
        name="PESScan_100_H2_m3gnet",
        molecule=system,
    )
    pesscan_100_H2.run()
    # !amsmovie {pesscan_100_H2.results.rkfpath()} # open in AMSmovie
    plot_pesscan(pesscan_100_H2)

    # ### Surface and bulk diffusion H on/in Ru(0001)

    from common_ru_h import (
        get_surface_diffusion_scan_coordinates,
        get_bulk_diffusion_scan_coordinates,
    )
    from common_ru_h import (
        get_bottom_atom_index,
        pesscan_settings,
        m3gnet_up_settings,
        constraints_settings,
        plot_pesscan,
    )

    system = slab_001_H2

    scan_coordinates = get_surface_diffusion_scan_coordinates(system, atom_
    ↪index=len(system) - 1)
    scan_coordinates += get_bulk_diffusion_scan_coordinates(system, atom_
    ↪index=len(system), delta_z=2.0)

    pesscan_001_H2 = plams.AMSJob(
        settings=(
            pesscan_settings(scan_coordinates, n_points=20)
            + m3gnet_up_settings()
            + constraints_settings(get_bottom_atom_index(system))

```

(continues on next page)

(continued from previous page)

```

    ),
    name="PESScan_001_H2_m3gnet",
    molecule=system,
)
pesscan_001_H2.run()
# !amsmovie {pesscan_001_H2.results.rkfpath()} # open in AMSmovie
plot_pesscan(pesscan_001_H2)

# ## Run DFT singlepoints on PES Scan structures and store on disk

from common_ru_h import dft_settings, replay_settings, QEKPointsConfig

replay_jobs = dict()
for pesscan_job in [pesscan_Ru_H_distance, pesscan_100_H2, pesscan_001_H2]:
    name = f"dft_replay_{pesscan_job.name}"
    settings = dft_settings(QEKPointsConfig(3, 3, 1)) + replay_settings(pesscan_job.
→results.rkfpath())
    replay_jobs[name] = plams.AMSJob(
        settings=settings,
        name=name,
    )

ri = ResultsImporter.from_yaml(old_ref_dir)
for name, job in replay_jobs.items():
    job.run()
    ri.add_trajectory_singlepoints(job, properties=["energy", "forces"])
    ri.store(new_ref_dir)
    plot_pesscan(job)

```

4.5.13 Ru/H Part 3: Initial reference data MD simulation + single-point replays

Important: First read *Ru/H introduction* (page 161) and follow all parts in order.

To follow along,

- Download `03_Ru_H2_gas_snapshots.ipynb` (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```

#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports

import scm.plams as plams
from scm.params import ResultsImporter
from scm.plams import Settings, AMSJob, log, Molecule, packmol_on_slab
from pathlib import Path
import matplotlib.pyplot as plt

```

(continues on next page)

(continued from previous page)

```

# common_ru_h.py must exist in the current working directory
from common_ru_h import (
    rotation,
    dft_settings,
    QEKPointsConfig,
    m3gnet_up_settings,
    replay_settings,
    slice_slab,
    check_installation,
)

# ## Initialize PLAMS working directory

old_ref_dir = "reference_data_2"
check_installation(old_ref_dir)
new_ref_dir = "reference_data_3"
ri = ResultsImporter.from_yaml(old_ref_dir)
plams.init()

# ## Construct the Ru(10-10)/H2(gas) interface
#
# For details about the construction of the slab, see the previous notebook.

optimized_bulk = ri.job_collection["hcp_lattopt_Ru_dft"].molecule
slab_100 = slice_slab(optimized_bulk, miller=(1, 0, 0), thickness=7.0, cell_z=15, ref_
    ↪atom=0)
slab_100 = slab_100.supercell(3, 2, 1)
for at in slab_100:
    at.properties = Settings() # remove details about supercell generation
plams.plot_molecule(slab_100, rotation=rotation)
plt.title("Ru(10-10)")

# Now use the ``packmol_slab`` function to add hydrogen molecules:

from scm.plams import packmol_on_slab

h2_mol = plams.from_smiles("[HH]")
density = 0.3 # approximate density of the gas phase in g/cm^3
slab_100_H2_gas_raw = packmol_on_slab(slab_100, h2_mol, density=density)
slab_100_H2_gas = plams.preoptimize(slab_100_H2_gas_raw, settings=m3gnet_up_
    ↪settings(), maxiterations=100)

plams.plot_molecule(slab_100_H2_gas, rotation=rotation, radii=0.8)

# Now let's run some MD:

mdjob = plams.AMSNVTJob(
    settings=m3gnet_up_settings(),
    name="md_ru10-10_h2_m3gnet",
    temperature=501,
    nsteps=10000,
    molecule=slab_100_H2_gas,

```

(continues on next page)

(continued from previous page)

```

        samplingfreq=100,
    )
mdjob.run()

from scm.params import ResultsImporter

ri = ResultsImporter.from_yaml(old_ref_dir)

settings = dft_settings(QEKPointsConfig(3, 3, 1))
settings += replay_settings(mdjob.results.rkfpath(), frames=[10, 30, 50, 70, 90])

dft_replay_job = plams.AMSJob(
    settings=settings,
    name="snapshots_from_md_ru10-10_h2_dft",
)
dft_replay_job.run(watch=True)

ri.add_trajectory_singlepoints(dft_replay_job, properties=["energy", "forces"])
ri.store(new_ref_dir)

```

4.5.14 Ru/H Part 4: Initial training

Important: First read *Ru/H introduction* (page 161) and follow all parts in order.

To follow along,

- Download 04_Ru_H_initial_training.ipynb (see also: how to install [Jupyterlab](#) in AMS)

Complete Python code

```

#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports

import scm.plams as plams
from scm.params import ResultsImporter, ParAMSJob
from scm.plams import Settings, AMSJob, log, Molecule, packmol_on_slab
from pathlib import Path
import matplotlib.pyplot as plt

# common_ru_h.py must exist in the current working directory
from common_ru_h import rotation, check_installation

# ## Initialize PLAMS working directory

old_ref_dir = "reference_data_3"
check_installation(old_ref_dir)

```

(continues on next page)

(continued from previous page)

```

new_ref_dir = "reference_data_4"
plams.init()

# ## Perform training/validation split

## Create a training/validation split
ri = ResultsImporter.from_yaml(old_ref_dir)
log("Performing training/validation split")
training_set, validation_set = ri.get_data_set("training_set").split_by_jobids(0.95,
↪0.05, seed=314)
ri.data_sets = {"training_set": training_set, "validation_set": validation_set}
log(f"{len(training_set)} training set entries; {len(validation_set)} validation set↪
↪entries.")
log(f"Storing in {new_ref_dir}")
ri.store(new_ref_dir)

# ## Create a ParAMS Job for transfer learning on the M3GNet universal potential

job = ParAMSJob.from_yaml(new_ref_dir)
job.name = "initial_training"
inp = job.settings.input
inp.Task = "MachineLearning"
inp.MachineLearning.CommitteeSize = 1 # train only a single model
inp.MachineLearning.MaxEpochs = 250
inp.MachineLearning.LossCoeffs.Energy = 10.0
inp.MachineLearning.LossCoeffs.Forces = 1.0
inp.MachineLearning.Backend = "M3GNet"
inp.MachineLearning.M3GNet.LearningRate = 1e-3
inp.MachineLearning.M3GNet.Model = "UniversalPotential"
inp.MachineLearning.M3GNet.UniversalPotential = Settings(
    Featurizer="No", # must use strings here, not Python booleans
    ThreeDInteractions1="No",
    GraphLayer1="No",
    ThreeDInteractions2="No",
    GraphLayer2="No",
    ThreeDInteractions3="Yes",
    GraphLayer3="Yes",
    Final="Yes",
)
inp.MachineLearning.Target.Forces.Enabled = "Yes"
inp.MachineLearning.Target.Forces.MAE = 0.05
inp.MachineLearning.RunAMSAAtEnd = "Yes"
# Larger batch sizes require more (GPU) memory but will also typically train faster
# The amount of memory also depends on the number of atoms in the structures
# So set the batch size to some appropriate number
inp.DataSet[0].BatchSize = 10 # training set batch size
inp.DataSet[1].BatchSize = 10 # validation set batch size
print(job.get_input())

job.run()

# ## Plot some results of the training

```

(continues on next page)

(continued from previous page)

```

job.results.plot_simple_correlation("forces")

job.results.plot_all_pes()
plt.subplots_adjust(top=2, hspace=0.5)

# ## Copy the results directory to a known place

import shutil

orig_training_results_dir = str(job.results.path)
new_training_results_dir = Path("initial_training_results").resolve()
log(f"Copying {orig_training_results_dir} to {new_training_results_dir}")
shutil.copytree(orig_training_results_dir, new_training_results_dir, dirs_exist_
    ↳ok=True)
log(f"Use {new_training_results_dir} as the LoadModel in upcoming active learning.")

```

4.5.15 Ru/H Part 5: Active learning for molecule gun MD

Important: First read *Ru/H introduction* (page 161) and follow all parts in order.

To follow along,

- Download 05_active_learning_molecule_gun_md.ipynb (see also: how to install Jupyterlab in AMS)

Complete Python code

```

#!/usr/bin/env amspython
# coding: utf-8

# ## Initial imports

import scm.plams as plams
from scm.params import ResultsImporter, ParAMSJob
from scm.plams import Settings, AMSJob, log, Molecule
from scm.simple_active_learning import SimpleActiveLearningJob
from pathlib import Path
import matplotlib.pyplot as plt

# common_ru_h.py must exist in the current working directory
from common_ru_h import (
    rotation,
    dft_settings,
    QEKPointsConfig,
    slice_slab,
    check_installation,
)

# ## Initialize PLAMS working directory

```

(continues on next page)

(continued from previous page)

```

load_model_dir = "initial_training_results"
check_installation(load_model_dir)
plams.init()

# ## Load the optimized bulk Ru structure from the job collection
#
# The lattice was optimized in the previous notebook, and the structure was stored in
# the job collection.
#
# Let's retrieve it from the job collection and use it to construct Ru surface slabs.

job_collection = ParAMSJob.load_external(load_model_dir).results.get_job_collection()
optimized_bulk = job_collection["hcp_lattopt_Ru_dft"].molecule
slab = slice_slab(optimized_bulk, miller=(1, 0, 0), thickness=7.0, cell_z=15, ref_
atom=0)
min_z = min(at.coords[2] for at in slab)
slab.translate((0, 0, -min_z + 2.0))
slab = slab.supercell(3, 2, 1)
plams.plot_molecule(slab, rotation=rotation)
plt.title("Ru(10-10)")

min_z = min(at.coords[2] for at in slab)
for i, at in enumerate(slab, 1):
    at.properties = Settings() # remove details about supercell generation
    if at.coords[2] == min_z:
        at.properties.region = "very_cold"
    else:
        at.properties.region = "thermostatted"

h_atom = plams.Molecule()
h_atom.add_atom(plams.Atom(symbol="H", coords=(0.0, 0.0, 0.0)))
h_atom.atoms[0].properties.region = "hydrogen"

main_system_name = "" # must be empty string
projectile_name = "projectile"
molecules_dict = {main_system_name: slab, projectile_name: h_atom}

# ## Set up the MD settings
#
# Before starting the active learning, let's set up a molecule gun simulation using
# the initially trained potential.
#
# This is just to see that that simulation settings are somewhat reasonable.

s = Settings()
s.input.ams.Task = "MolecularDynamics"
md_s = s.input.ams.MolecularDynamics
md_s.NSteps = 5000 # will be increased later for active learning
md_s.Trajectory.SamplingFreq = 10 # for testing purposes to check the trajectory
md_s.InitialVelocities.Temperature = 100
md_s.Thermostat = [

```

(continues on next page)

(continued from previous page)

```

    Settings(
        Region="thermostatted",
        Type="NHC",
        Temperature=[300],
        Tau=100.0,
    ),
    Settings(
        Region="very_cold",
        Type="NHC",
        Temperature=[2.0],
        Tau=10.0,
    ),
]
md_s.RemoveMolecules.Frequency = 1
md_s.RemoveMolecules.Formula = "*"
md_s.RemoveMolecules.SinkBox.FractionalCoordsBox = "0 1 0 1 0.90 0.99"
md_s.AddMolecules.System = projectile_name
md_s.AddMolecules.Frequency = 1000
md_s.AddMolecules.StartStep = 100
# insert H atoms 4.5 angstrom above the surface
max_z = max(at.coords[2] for at in slab)
projectile_insertion_z = (4.5 + max_z) / slab.lattice[2][2]
md_s.AddMolecules.FractionalCoordsBox = f"0 1 0 1 {projectile_insertion_z}
↪{projectile_insertion_z+0.01}"
md_s.AddMolecules.VelocityDirection = "0 0 -1" # shoot down towards slab (decrease z ↪
↪coordinate)
md_s.AddMolecules.DeviationAngle = 0.0
md_s.AddMolecules.Velocity = 0.03

test_md_job = AMSJob(
    settings=s + ParAMSJob.load_external(load_model_dir).results.get_production_
↪engine_settings(),
    molecule=molecules_dict,
    name="test_molecule_gun",
)

test_md_job.run()

# Open the trajectory in AMSmovie to check if it is reasonable. We'd expect some ↪
↪combination of the following events:
#
# * H atoms adsorbing on the Ru surface
# * H atoms diffusing into the subsurface
# * H atoms desorbing from the Ru surface
# * H atoms combining into H2 molecules and desorbing from the surface
#
# The simulation seems reasonable, so let's couple it to the active learning with on- ↪
↪the-fly retraining.
#
# ## Active Learning for Ru/H molecule gun simulation

plams.config.jobmanager.hashing = None
al_s = plams.Settings()
al_s.input.ams.ActiveLearning.Steps.Type = "Linear"

```

(continues on next page)

(continued from previous page)

```

al_s.input.ams.ActiveLearning.Steps.Linear.Start = 1000
al_s.input.ams.ActiveLearning.Steps.Linear.StepSize = 5000
# H atoms at high temperature, so let's decrease the minimum allowed distance a bit
al_s.input.ams.ActiveLearning.ReasonableSimulationCriteria.Distance.MinValue = 0.50
al_s.input.ams.ActiveLearning.MaxReferenceCalculationsPerAttempt = 1
al_s.input.ams.ActiveLearning.SuccessCriteria.Forces.MaxDeviationForZeroForce = 0.65

ml_s = plams.Settings()
ml_s.input.ams.MachineLearning.Backend = "M3GNet"
ml_s.input.ams.MachineLearning.LoadModel = Path(load_model_dir).resolve()
ml_s.input.ams.MachineLearning.MaxEpochs = 50
ml_s.input.ams.MachineLearning.RunAMSAtEnd = "No"

ref_s = dft_settings(QEKPointsConfig(3, 3, 1), conv_thr=1e-4)

new_md_s = s.copy()
new_md_s.input.ams.MolecularDynamics.NSteps = 100000
new_md_s.input.ams.MolecularDynamics.Trajectory.SamplingFreq = 100

al_job = SimpleActiveLearningJob(name="sal", settings=al_s + ml_s + ref_s + new_md_s,
↪molecule=molecules_dict)

al_job.run()

```

See also:

- [Getting Started with PLAMS](#)
- [PLAMS Examples](#)
- [ParAMS Python tutorial](#)
- [Simple Active Learning Python API](#) (page 176)
- [Simple Active Learning tutorial using the Graphical User Interface](#)

4.6 Python API

Python APIs for the `SimpleActiveLearningJob` (a type of PLAMS Job) and `SimpleActiveLearningResults` classes.

Import it like this:

```

#!/usr/bin/env amspython

from scm.simple_active_learning import SimpleActiveLearningJob

```

See also:

- [Getting Started with PLAMS](#)
- [Python Input System for AMS \(PISA\)](#) (*new in AMS2024*)
- [Simple Active Learning Python Examples](#) (page 109)

4.6.1 SimpleActiveLearningJob

class SimpleActiveLearningJob (*name='simple_active_learning_job', driver=None, settings=None, molecule=None, **kwargs*)

PLAMS Job class for running Simple Active Learning.

This class inherits from the PLAMS SingleJob class. For usage, see the SingleJob documentation.

If you supply a Settings object to the constructor, it will be converted to a PISA (Python Input System for AMS) object.

Attributes:

- **input**: an alias for `self.settings.input`

__init__ (*name='simple_active_learning_job', driver=None, settings=None, molecule=None, **kwargs*)

Initialize the SimpleActiveLearningJob.

name

[str] The name of the job

driver

[scm.input_classes.drivers.SimpleActiveLearning] PISA object describing the input to the SimpleActiveLearning program

settings: scm.plams.Settings

All settings for the job. Input settings in the PLAMS settings format under `settings.input` are automatically converted to the PISA format. You can specify `settings.runscript.nproc` to set the total number of cores to run on.

molecule: scm.plams.Molecule or Dict[str, scm.plams.Molecule]

The initial system in PLAMS Molecule format, or if the simulation requires multiple input system, given as a dictionary where the main system has an empty string "" as the key.

classmethod load_external (*path, finalize=False*)

Load a previous SimpleActiveLearning job from disk.

Parameters

- **path** (*Union[str, Path]*) – A reactions discovery results folder.
- **finalize** (*bool, optional*) – See SingleJob, defaults to False

Raises

FileError – When the path does not exist.

Returns

An initialized SimpleActiveLearningJob

Return type

SimpleActiveLearningJob (page 177)

classmethod from_rkf (*path*)

Initialize a job from a simple_active_learning.rkf file.

Parameters

path (*str*) – Path to a simple_active_learning.rkf file

Returns

A new SimpleActiveLearningJob instance based on the information found in path.

Return type

SimpleActiveLearningJob (page 177)

classmethod `from_input` (*text_input*)

Initialize a job from text input.

Parameters

text_input (*str*) – A multiline text input

Returns

A SimpleActiveLearningJob

Return type

[SimpleActiveLearningJob](#) (page 177)

classmethod `restart_from` (*job*, *name*='simple_active_learning_job', *molecule*=None, *keep_initial_reference_data_settings*=False, *job_prefix*=None)

Returns a SimpleActiveLearningJob with LoadModel set appropriately.

Parameters

- **job** (*Union*[[SimpleActiveLearningJob](#) (page 177), *str*, *Path*]) – A previously finished SimpleActiveLearningJob (or path to its results folder)
- **molecule** (*Optional*[*Molecule*], *default* None) – Input molecule for the new job. If None, use the final molecule from job.
- **keep_initial_reference_data_settings** (*bool*, *default* False) – Whether to keep the ActiveLearning%InitialReferenceData block from the original job.

Returns

Returns a new SimpleActiveLearningJob.

Return type

[SimpleActiveLearningJob](#) (page 177)

get_errormsg ()

Returns the contents of the jobname.err file if it exists. If the file does not exist an empty string is returned.

Returns

The error message

Return type

str

get_input ()

Obtain the input string.

Returns

An input string.

Return type

str

get_runscript ()

Generates the runscript. Use `self.settings.runscript.preamble_lines = ['line1', 'line2']` or similarly for `self.settings.runscript.postamble_lines` to set custom settings.

`self.settings.runscript.nproc` controls the total number of cores to run on.

check ()

Returns True if “NORMAL TERMINATION” is given in the General section of simple_active_learning.rkf.

ok()
 Synonym for check()

run (*jobrunner=None, jobmanager=None, watch=False, **kwargs*)
 Runs the job

property input
 PISA format input

4.6.2 SimpleActiveLearningResults

Note: You should not initialize SimpleActiveLearningResults yourself. Instead always access it as `job.results`, where `job` is of type SimpleActiveLearningJob.

```
class SimpleActiveLearningResults(job)
    Results class for SimpleActiveLearningJob

    get_errormsg()
        Returns the error message of this calculation if any were raised.

        Returns
            String containing the error message.

        Return type
            str

    rkfpath(file='simple_active_learning')
        Returns path to simple_active_learning.rkf

        Returns
            Path to simple_active_learning.rkf

        Return type
            str

    readrkf(section, variable)
        Reads simple_active_learning.rkf

    get_simulation_directory(step=None, attempt=None, allow_final=True)
        Returns the absolute path to a simulation directory.

        step: optional, int
            The step number. If not specified will be autodetected to the last step.

        attempt: optional, int
            The attempt number. If not specified will be autodetected to the last attempt.

        allow_final: bool
            If True and step=None and attempt=None, then it will return final_production_simulation if it exists.

    get_main_molecule(allow_final=True)
        Returns AMSResults.get_main_molecule() on the main simulation job.

        Parameters
            allow_final(bool, optional) – _description_, defaults to True

        Returns
            _description_
```

Return type

Union[Molecule, Dict[str, Molecule], None]

get_params_results_directory (*step=None, attempt=None, allow_final=True*)

Returns the absolute path to a ParAMS results directory that can be loaded with ParAMSJob.load_external or used as LoadModel in ParAMS or SimpleActiveLearning input.

step: optional, int

The step number. If not specified will be autodetected to the last step.

attempt: optional, int

The attempt number. If not specified will be autodetected to the last attempt.

allow_final: bool

If True and step=None and attempt=None, then it will return final_training/results if it exists.

get_params_job (*step=None, attempt=None, allow_final=True*)

Returns the latest ParAMSJob. This can be used to analyze results from the parametrization.

Parameters

- **step** (*Optional[int], optional*) – _description_, defaults to None
- **attempt** (*Optional[int], optional*) – _description_, defaults to None
- **allow_final** (*bool, optional*) – _description_, defaults to True

Returns

description

Return type

ParAMSJob

get_production_engine_settings (*step=None, attempt=None, allow_final=True*)

Returns the production engine settings from the ParAMSJob

get_reference_data_directory (*step=None, attempt=None*)

Returns the absolute path to a reference data directory that can be opened in the ParAMS GUI or which lets you initialize a ParAMSJob with ParAMSJob.from_yaml()

step: optional, int

The step number. If not specified will be autodetected to the last step.

attempt: optional, int

The attempt number. If not specified will be autodetected to the last attempt.

allow_final: bool

If True and step=None and attempt=None, then it will return final_training/results if it exists.

4.7 Frequently Asked Questions

4.7.1 What kind of properties can I fit?

In AMS2024 you can only train to

- energies
- forces

4.7.2 Can I run on the GPU?

Yes, both the training and production simulations can be run on a compatible GPU when training/running M3GNet.

Running on the GPU results in significant speedup so it is recommended to do so.

To use the GPU you must install a GPU-compatible version of M3GNet or other machine learning backends.

See the [MLPotential](#) and [ParAMS](#) documentations.

Note: The engines typically used for DFT reference calculations, like ADF, BAND, and the AMS-bundled version of Quantum ESPRESSO, cannot run on the GPU.

The Active Learning procedure repeatedly switches between ML training, ML production simulations, and reference calculations.

4.7.3 What kinds of MD simulations can I run?

Simple Active Learning is compatible with (almost) all types of [MD simulations](#) supported by the AMS Driver.

Type	Comment
✓ NVE	
✓ NVT	example (page 109) tutorial
✓ NPT	
✓ Temperature ramping	example (page 120)
✓ Multiple thermostats (thermal conductivity)	
✓ Molecule gun or sink (CVD, ALD)	
✓ CREST metadynamics	example (page 131)
✓ Constraints (e.g., fixed positions)	
✓ Apply force or velocity (friction and viscosity)	
✓ Lattice deformation linear	Use Steps%Type = Linear example (page 120)
✓ Reaction boost force	Use committee uncertainties example (page 153)
✓ Reaction boost target coordinate	Use Steps%Type = Linear example (page 143)
✓ Force-bias Monte Carlo (fbMC)	
(✓) Lattice deformation periodic	Difficult setup to get good training/validation sets
(✓) Nanoreactor	Difficult setup to get good training/validation data
✗ Very high temperatures (combustion)	The workflow automatically considers very high temperatures (> 5000 K) to be unrealistic and assumes that they are artifacts of the model

4.7.4 Can VASP be used with Simple Active Learning?

Simple Active Learning does not support VASP as the engine. Instead, you may use ADF, BAND or Quantum ESPRESSO. Alternatively you can try to set up any other reference method through [engine ASE](#).

Non-alphabetical

`__init__()` (*OLEDPropertiesSummary* method), 31
`__init__()` (*ReactionsDiscoveryJob* method), 77
`__init__()` (*SimpleActiveLearningJob* method), 177
`__str__()` (*OLEDPropertiesSummary* method), 31
`_extract_mol_from_pisa()` (*ReactionsDiscoveryJob* static method), 77

A

`as_yaml()` (*OLEDPropertiesSummary* method), 32

C

`check()` (*ReactionsDiscoveryJob* method), 78
`check()` (*SimpleActiveLearningJob* method), 178

F

`from_input()` (*ReactionsDiscoveryJob* class method), 77
`from_input()` (*SimpleActiveLearningJob* class method), 178
`from_rkf()` (*ReactionsDiscoveryJob* class method), 77
`from_rkf()` (*SimpleActiveLearningJob* class method), 177

G

`get_errormsg()` (*ReactionsDiscoveryJob* method), 77
`get_errormsg()` (*ReactionsDiscoveryResults* method), 75
`get_errormsg()` (*SimpleActiveLearningJob* method), 178
`get_errormsg()` (*SimpleActiveLearningResults* method), 179
`get_input()` (*ReactionsDiscoveryJob* method), 78
`get_input()` (*SimpleActiveLearningJob* method), 178
`get_main_molecule()` (*SimpleActiveLearningResults* method), 179
`get_md_jobs()` (*ReactionsDiscoveryJob* method), 78
`get_md_jobs()` (*ReactionsDiscoveryResults* method), 75
`get_network()` (*ReactionsDiscoveryResults* method), 76

`get_network_rd()` (*ReactionsDiscoveryResults* method), 76
`get_num_md_simulations()` (*ReactionsDiscoveryResults* method), 76
`get_params_job()` (*SimpleActiveLearningResults* method), 180
`get_params_results_directory()` (*SimpleActiveLearningResults* method), 180
`get_production_engine_settings()` (*SimpleActiveLearningResults* method), 180
`get_reference_data_directory()` (*SimpleActiveLearningResults* method), 180
`get_runscript()` (*ReactionsDiscoveryJob* method), 78
`get_runscript()` (*SimpleActiveLearningJob* method), 178
`get_simulation_directory()` (*SimpleActiveLearningResults* method), 179

I

`input` (*ReactionsDiscoveryJob* property), 78
`input` (*SimpleActiveLearningJob* property), 179

L

`load_external()` (*ReactionsDiscoveryJob* class method), 78
`load_external()` (*SimpleActiveLearningJob* class method), 177

O

`ok()` (*ReactionsDiscoveryJob* method), 78
`ok()` (*SimpleActiveLearningJob* method), 178
OLEDPropertiesSummary (class in *scm.oledtools*), 31

R

ReactionsDiscovery, 41
ReactionsDiscoveryJob (class in *scm.reactions_discovery.plams_job*), 76
ReactionsDiscoveryResults (class in *scm.reactions_discovery.plams_job*), 75
`readrkf()` (*SimpleActiveLearningResults* method), 179

`restart_from()` (*SimpleActiveLearningJob* class method), 178
`rkfpath()` (*ReactionsDiscoveryResults* method), 76
`rkfpath()` (*SimpleActiveLearningResults* method), 179
`run()` (*SimpleActiveLearningJob* method), 179

S

Simple Active Learning, 81
SimpleActiveLearningJob (class in *scm.simple_active_learning.plams.simple_active_learning_job*), 177
SimpleActiveLearningResults (class in *scm.simple_active_learning.plams.simple_active_learning_job*), 179