MLIP: a user manual

Alexander V. Shapeev (e-mail: a.shapeev@skoltech.ru)
Evgeny V. Podryabinkin (e-mail: e.podryabinkin@skoltech.ru)
Ivan S. Novikov (e-mail: i.novikov@skoltech.ru)

October 13, 2017

Abstract

MLIP is a software package implementing several machine learning interatomic potentials (aka force fields) for calculating the energy, forces and stresses of atomic configurations. Such potentials seek to combine the accuracy of quantum mechanics with the computational efficiency of the empirical potentials.

Contents

1 Some important info

1.1 Licensing and distribution

This package is developed by the group of Alexander Shapeev (`a.shapeev@skoltech.ru`) at Skoltech (Skolkovo Institute of Science and Technology), Moscow, Russia. It is distributed with the New BSD license. This means you can use this software for practically any purpose provided that you give credit to the authors of this software—see the `LICENSE` file for the details.

1.2 A Quickstart guide (aka TL;DR)

You like pushing buttons but not reading long texts? Try:

- Download the source through git by executing

  ```
  git clone http://gitlab.skoltech.ru/shapeev/mlip.git
  ```

  This will create the `mlip/` folder with the sources.

- Install: Execute `make mlp` in the `make/` folder to build the `mlp` tool. Run the shell script in `make/LAMMPS/` to install MLIP with LAMMPS. The `lmp_serial` and `lmp_mpi` binaries will be created in the `LAMMPS/src/` folder and copied over to the `make/` folder.

- Try out examples, see `doc/examples/readme.txt`.

2 Theory

You are advised to read Section 2.1, but you can skip the rest of Section 2 if you have read [2, 1].

2.1 Basic terms and definitions

- In this document, we will refer to this package as *MLIP*, while the machine learning interatomic potentials will be just called "machine learning potentials" or just "potentials".
- The empirical (classical) interatomic potentials, such as Lennard-Jones or EAM, will be specifically referred to as the *empirical potentials*.
- *Ab initio model* (AIM) means any interatomic interaction model or potential which is considered to reproduce sufficiently accurately the behavior of the real atomic systems. Such models are typically computationally expensive. The main examples are Density Functional Theory (DFT), Orbital-free DFT, Hartree-Fock, etc. This package allows for using LAMMPS as an ab initio model.
- We will call an atomistic *configuration* a system with N atoms described by a collection of their types (presently, MLIP handles configurations with only one type, i.e, single-component configurations), their positions, and the supercell given by three lattice vectors ($L = \begin{pmatrix} l_1 & l_2 & l_3 \end{pmatrix} \in \mathbb{R}^{3\times 3}$). A configuration is typically denoted by x. The supercell, thus, defines the periodic boundary conditions. A very large supercell can be given to simulate non-periodic boundary conditions.
- The *neighborhood* of atom i, $1 \le i \le m$, is defined as a collection (tuple) of vectors pointing from atom i to all the atoms and their periodic extensions, excluding the atom itself, that are not farther than $R_{\rm cut}$ (inside a cutoff sphere).
- The "EFS" abbreviation will be used for the collection of the energy $E(x)$, forces $(f_i(x))_{1\le i\le N}$ acting on atoms, and stresses $\sigma(x) = (\sigma_{ij}(x))_{1\le i,j\le 3}$ of the atomic configuration x.
- We will use the term *driver* as the algorithmic component of an atomistic simulation that generates a set (or a sequence) of configurations. The examples of drivers are: molecular dynamics (MD), structure relaxation, Monte-Carlo sampling, nudge elastic band, etc. Also, reading configurations from database is also considered a driver in MLIP.

2.2 Machine learning potentials

The main purpose of machine learning potentials is to reproduce the behavior of ab initio models (AIMs) at a tiny fraction of the computational cost. Machine learning potentials postulate a partitioning of energy into individual contributions of atoms, each contribution depends on the neighborhood of the atom, and assume a flexible functional form for such contributions depending on the positions and types of surrounding atoms within a certain cutoff radius, typically with hundreds or more parameters. These parameters are found by requiring the energy, forces and/or stresses predicted by a potential to be close to those obtained by an AIM on some atomic configurations. These configurations are called the *training set*, and finding the parameters of a potential is known as *training* or *fitting*. One of the important features (requirements) of a potential is its ability to approximate potential energy surfaces with an arbitrary accuracy (at least theoretically) by increasing the number of parameters and the training set. Due to this, one can improve the accuracy of the potential at the cost of reducing its computational efficiency and vice versa.

2.3 Moment tensor potentials

Moment Tensor Potentials (MTPs) [2] are machine learning potentials implemented in the MLIP package. MTPs adopt a linear regression model with polynomial-like functions of atomic coordinates as the basis functions. Potential energy (contribution) of an atom (site energy) provided by MTP as a function of atomic environment within a cutoff sphere (neighborhood) is invariant with respect to all Euclidian transformations and permutations of the chemically equivalent atoms. It has been proved theoretically and shown computationally that MTP allows for systematically improving the accuracy by increasing the number of basis functions.

MTP has a complex functional form that depends on the number of parameters and the particular basis functions. Therefore MTPs, as a rule, can not be written in a plain compact form and require a complex algorithms for their efficient evaluation and fitting. Such algorithms are implemented in MLIP. The current implementation of MTPs is limited to systems with one atomic type.

2.4 Fitting

Mathematically, training (fitting) of a potential is determining its unknown parameters from a (typically, overdetermined) system of linear equations, which expresses that the energy, forces and stresses (EFS in short) predicted by the potential are close to the given (ab initio) ones:

$$\begin{aligned}
E\big(x^{(k)}\big) &= E^{\rm ai}\big(x^{(k)}\big)\\
f_i\big(x^{(k)}\big) &= f_i^{\rm ai}\big(x^{(k)}\big), \quad 1 \le i \le N^{(k)}\\
\sigma\big(x^{(k)}\big) &= \sigma^{\rm ai}\big(x^{(k)}\big)
\end{aligned}$$

for the configuration $x^{(k)}, 1 \le k \le K$ from the training set [2, 1].

The EFS enter the least squares functional with different coefficients:

$$\sum_{k=1}^{K}\Bigg[\ \frac{C_E}{N^{(k)}}\Big(E\big(x^{(k)}\big)-E^{\rm ai}\big(x^{(k)}\big)\Big)^2\ + \\ +\ C_f\sum_{i}^{N^{(k)}}\Big|f_i\big(x^{(k)}\big)-f_i^{\rm ai}\big(x^{(k)}\big)\Big|^2\ + \\ +\ \frac{C_\sigma}{N^{(k)}}\sum_{i,j=1}^{3}\Big(\sigma_{ij}\big(x^{(k)}\big)-\sigma_{ij}^{\rm ai}\big(x^{(k)}\big)\Big)^2\ \Bigg]\longrightarrow \min, \tag{1}$$

where $N^{(k)}$ is the size of (i.e., the number of atoms in) $x^{(k)}$. Note that the scaling by $N^{(k)}$ is chosen so that the same configuration with a larger unit cell has the same relative contributions of EFS. The coefficients C_E, C_f, C_σ are the adjustable parameters of the fitting. The default values are $C_E = 1$, $C_f = 0.01\text{Å}^2$, $C_\sigma = 0.001$.

There is an alternative form of fitting, where the small forces are fitted more accurately than the large

$$\sum_{k=1}^{K}\Bigg[\ \frac{C_E}{N^{(k)}}\Big(E\big(x^{(k)}\big)-E^{\rm ai}\big(x^{(k)}\big)\Big)^2\ + \\ +\ C_f\sum_{i}\frac{\big|f_i\big(x^{(k)}\big)-f_i^{\rm ai}\big(x^{(k)}\big)\big|^2}{\big|f_i^{\rm ai}\big(x^{(k)}\big)\big|^2+\epsilon_f^2}\ + \\ +\ \frac{C_\sigma}{N^{(k)}}\sum_{i,j=1}^{3}\Big(\sigma_{ij}\big(x^{(k)}\big)-\sigma_{ij}^{\rm ai}\big(x^{(k)}\big)\Big)^2\ \Bigg]\longrightarrow \min, \tag{2}$$

where ϵ_f is an additional adjustable parameter.

2.5 Active learning

There are two approaches to the fitting of the potentials, known as passive learning and active learning (AL). In contrast to passive learning in which a potential learns every configuration in the training set, the concept of active learning assumes some algorithm analyzing and selecting an optimal (D-optimal) training subset [1]. The key component of any AL method is, thus, its *query strategy*—an algorithmic criterion for deciding whether a given configuration should be added in the training subset. The query strategy does not use ab initio EFS (otherwise the whole purpose of AL is defeated).

There are three query strategies currently implemented in MLIP based on an estimate of degree of extrapolation when a machine learning potential is used for calculation of (1) configuration energy, (2) site energy of atomic neighborhoods, and (3) forces and stresses. It has been shown that active learning does not lead to a decrease of approximation accuracy (moreover the maximal errors typically diminish) whereas its transferability (i.e. ability to predict EFS reliably outside the training domain) improves.

2.6 Learning on the fly

Active learning is the key component of our version of learning on the fly, where the training set selection, the fitting of the potential, and an atomistic simulation are combined in a single process (learning on the fly is schematically shown in Fig. 1).

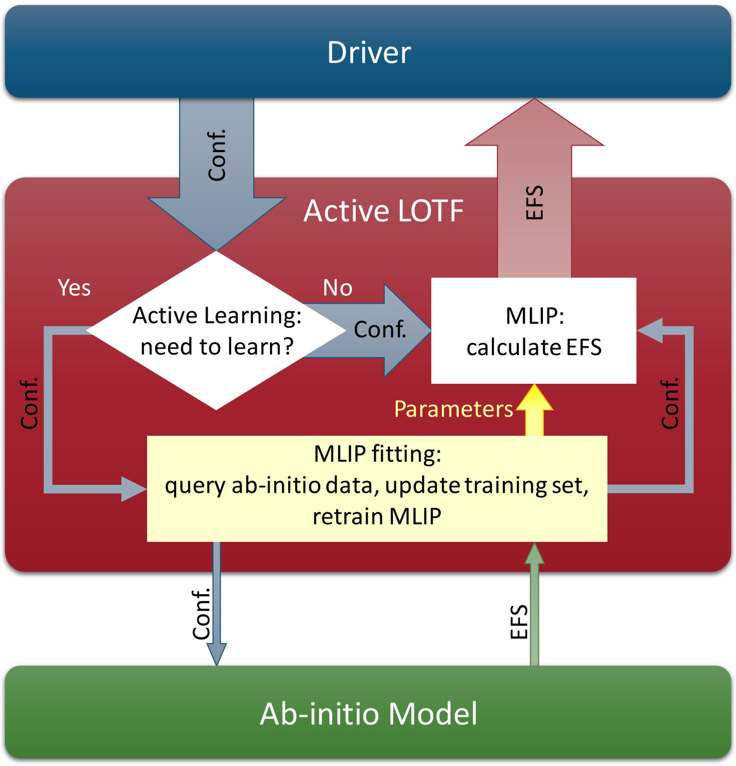
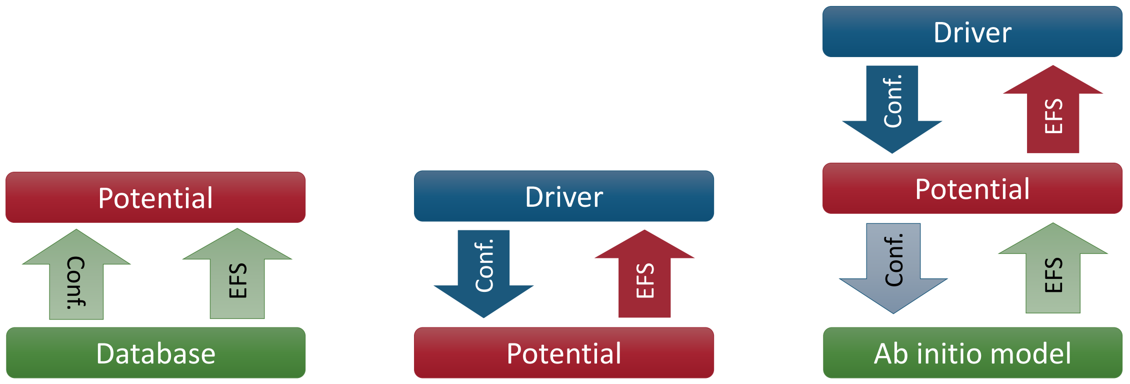

Driver
Conf.
EFS
Active LOTF
Yes
No
Active Learning: need to learn?
Conf.
MLIP: calculate EFS
Conf.
Conf.
Parameters
MLIP fitting: query ab-initio data, update training set, retrain MLIP
Conf.
EFS
Ab-initio Model

Figure 1: Workflow in actively learning a potential on the fly. In the course of iterations an AL potential receives from MD a configuration and estimates the reliability of EFS calculation. If it detects sufficient extrapolation when calculating EFS, MLIP queries ab initio EFS, add this configuration to the training set, and retrains the potential. Then it calculates the EFS and passes it to the driver.

3 MLIP concept

MLIP, roughly speaking, provides tools for fitting potentials and using them in atomistic simulations. MLIP, thus, can be used:

- as a stand-alone package `mlp`, allowing for fitting the potentials and doing some simple atomistic simulations (see Section 4.1 for installation), and
- as a LAMMPS plugin, for doing most of the things that LAMMPS does (see Section 4.2 for installation).



(a) fitting on a database

(b) simple atomisitic simulation

(c) a learning-on-the-fly simulation

Figure 2: Diagrams showing different ways of using MLIP: fitting and atomistic simulation.

The following is the typical use cases of MLIP:

(a) Fitting a potential on a configuration database (see Figure 2(a)). This is achieved through `mlp train`. The fitting errors (and validation errors) can be computed using `mlp calc-errors`.

(b) A simple atomistic simulation with a trained potential (see Figure 2(b)). Typically, this is done through LAMMPS.

(c) A learning-on-the-fly type of simulation (see Figure 2(c)). This can be done both, through LAMMPS and `mlp run`.

In fact, the scenario (c) encompasses all the other scenarios (at least in the serial version of MLIP). For example, (b) corresponds to (c) when Potential decides not to do any ab initio calculations, while (a) corresponds to (c) when Driver="reading from a database" and AIM="preserve EFS as read from the database".

More details are given in Section 7.

4 Installation

MLIP uses OpenBLAS (`http://www.openblas.net`) as a static library and is distributed as a part of this package (namely, OpenBLAS v.0.2.19 in the `blas/` folder). MLIP's makefile is configured to build and link to it automatically; if you want to link to your version of BLAS (for any reason) you should modify the makefile accordingly.

MLIP can be build and used (1) as stand-alone package; and (2) as a part of LAMMPS.

4.1 Building as a stand-alone package

Let `MLIP/` be a directory with MLIP sources pulled from `http://gitlab.skoltech.ru/shapeev/mlip`.

1. Specify your C++ compiler in the makefile:

   In the makefile (located in `MLIP/make/`) specify the compiler supporting the `C++11` standard (in a line beginning with "`CC =`"). The default is `g++`.

2. In the directory `MLIP/make/` build MLIP by executing `make mlp`. The executable `mlp` should then appear in the same directory.

4.2 Building as a LAMMPS module

You should have the LAMMPS sources in a directory, say, `LAMMPS/` and MLIP sources in `MLIP/`.

1. MLIP and LAMMPS should be build with the same compiler. So, make sure that the proper compiler is set in the makefile (see the previous section).

2. For building MLIP with LAMMPS we have written a script doing all the installation routines (but presently not very helpful if errors occur).

   In `MLIP/make/LAMMPS/` execute `install4LAMMPS.sh LAMMPS/` passing LAMMPS path as a parameter for the script.

The script should build MLIP to a static library, link it to LAMMPS, and build both the serial and parallel versions of LAMMPS. The executables `lmp_serial` and `lmp_mpi` should appear in `LAMMPS/src/` and be moved to `MLIP/make/`. These executables will have MLIP embedded in them.

4.3 Building on Windows

Visual Studio projects of MLIP are available in `MLIP/dev/` folder.

5 Usage examples

The examples demonstrate the most frequently used routines. Each example contains a brief description in the `readme.txt` file, the input files, the settings file `mlp.ini` and the expected output in the `sample_out` folder. The output (normally) is written to the `out/` folder. All examples are available in `MLIP/doc/examples/`.

6 MLIP file formats

6.1 Internal format of configuration files

MLIP supports two internal formats for storing configurations in database files, typically having the "`.cfg`" extension: textual and binary. We recommend using the binary format when the speed of input/output operations is critical. The textual format, in contrast, is useful when configurations are prepared manually or converted from an unsupported format.

The example of configuration in the textual file format is bellow:

```
BEGIN_CFG
 Size
     12
 Supercell
        5.804540      0.000000      0.000000
       -0.000052      4.304488      0.000000
 AtomData: id type  cartes_x  cartes_y  cartes_z        fx        fy        fz
            1    0  -0.06668   1.48178   2.12328   0.00116   0.00035   0.00023
            2    1   1.23010   2.65535   1.49390  -0.00259  -0.00072   0.00170
            3    0  -0.54459   4.36682   3.13176   0.00168   0.00059   0.00035
            4    0   3.12308   2.65537   1.51976  -0.00075   0.00044  -0.00077
            5    0   4.41954   1.48201   0.89064   0.00001  -0.00156  -0.00136
            6    1   3.58878   3.89084   2.64425   0.00008   0.00077  -0.00044
            7    0   3.44883   4.27728   0.83155   0.00105   0.00136   0.00081
            8    0   0.46582   2.80689   3.10745  -0.00320  -0.00174  -0.00034
            9    0   0.76409   3.89093   0.36959  -0.00025  -0.00035  -0.00032
           10    1   3.30134   2.21479   3.25088   0.00283  -0.00171   0.00059
           11    0   0.90387   4.27708   2.18219   0.00016   0.00292  -0.00053
           12    0   2.29143   4.95920   3.27520  -0.00019  -0.00036   0.00006
 Energy
        -76.697992341920
 Stress:  xx          yy          zz          yz          xz          xy
    -0.00121     0.00075     0.00233     0.00191     0.00246    -0.00156
 Feature    EFS_by    VASP
 Feature    from      relaxation_driver
 Feature    min_dist  1.734226
 Feature    type_0    B
 Feature    type_1    Al
 Feature    comment
 Feature    comment    ((`'-"""-'`))
 Feature    comment     )  -   -  (
 Feature    comment    /   o _ o   \
 Feature    comment    \   ( 0 )   /
 Feature    comment     '-.._^_..-'
END_CFG
```

The description of each configuration in the database starts with `BEGIN_CFG` and ends with `END_CFG`. No symbols except whitespace (which is space, tab, and the newline characters '\n' and '\r') are allowed between configurations. After `BEGIN_CFG` several compulsory and optional fields follow. Bellow is the list of recognizable fields.

1. `Size` (compulsory field) is followed by one integer number, the number of atoms in the configuration. `Size` should appear before "`AtomData:`".

2. `Supercell` (optional field) is followed by 3, 6, or 9 numbers, the 3 coordinates of the first lattice vector, then (optionally) 3 coordinates of the second lattice vector, and (optionally) 3 coordinates of the third lattice vector. If `Supercell` is missing then no lattice vectors are given (this is a open-shell molecule).

3. "`AtomData:`" (compulsory field) contains the per-atom data (atomic coordinates, forces, atom types, etc.). `AtomData:` is followed by one or more field names (on the same line):

   (a) `id` (optional field) means the ordinal number of atom in the configuration (1, 2, ...). If `id`'s are given, but are different from 1, 2, 3 in exactly in this order, it is an error.

   (b) `type` (optional field) are the element types (0, 1, . . .); if they are empty then all atoms are considered as of 0 type and this is a single-component configuration.

   (c) `cartes_x`, `cartes_y`, `cartes_z` (all three are compulsory fields) means the Cartesian coordinates (measured in Å). There is an alternative option to provide the configuration with direct (fractional) coordinates. In this case `direct_...` caption should be specified instead of `cartes_...`.

   (d) `fx`, `fy`, `fz` (optional fields, but all three are required if present), are the force components in Cartesian coordinates (measured in eV/Å).

   (e) `site_en` (optional field) is the site energy (in a rare occasion it is defined).

   (f) `charge` (optional field) may be used for storing of atomic partial charges.

4. `Energy` (optional field) is followed by one number (measured in eV).

5. "`Stress:`" (optional field) must be immediately (on the same line) followed by the following 6 (not more and not less) field names (in any order), `xx`, `yy`, `zz`, `yz`, `xz`, and `xy`. On the next line are the 6 numbers, corresponding to those stresses. These are virial stresses multiplied by the cell volume (measured in eV). Positive stresses correspond to over-stretched configurations (in other words, "stress = −force on the supercell")

6. "`Feature`" (optional fields, multiple entries are allowed) is/are additional (textual) attributes of a configuration (such as what chemical elements corresponds to the atom indexes, what generated it, what computed its energy, whether it is an ideal crystal or has a defect, etc.). Each "Feature" has a name (one word without whitespace)

and value (a string separated from the name by a space or a tab character). A configuration may contain several features. If two lines with the same name appear, the newline (‘`\n`’) character and the second line are appended to the first line; similarly for the third line, etc. The features do not have universal meanings, they are used by different tools.

6.2 MTP file

The functional form of MTP and, optionally, the parameters are encoded in the `.mtp` files. MTPs may differ by:

- particular basis functions of the atomic environment;
- the number of basis functions;
- the minimal allowed interatomic distance;
- the cut-off radius;
- values of the parameters.

Some of these properties can be seen directly in an `.mtp` file, and the minimal and cutoff distances can be edited directly (if you do it you have to re-train the potential). Fitted MTPs can be distinguished from the non-fitted ones by the presence of the “`moment_coeffs`” string followed by the list of parameters in the `.mtp` file.

6.3 Settings file

MLIP can be configured to work with external codes, learn on the fly, etc. These settings are the given in the settings file (typically `mlip.ini`) and are described in Section 7.6.

7 A driver-potential-AIM framework

Many use cases fall into the driver-potential-ab initio model framework as described below. The interaction between an atomistic simulation (driver) and MLIP can be viewed as the following data exchange (according to the scheme shown in Fig. 2(c)):

1. the driver generates a configuration and passes it to the model;
2. the model optionally queries EFS for this configuration from the ab initio model;
3. the model calculates EFS for this configuration and passes it to the driver;
4. the driver processes EFS data from the model and generates new (or modify in some way) the configuration and repeats the first step.

This scheme obviously describes the plain atomistic simulation scenario (with no communication between MLIP and ab initio model) and the learning-on-the-fly scenario. Fitting and active learning from the database are also described by this scheme when the driver simply reads configurations from a database while the ab initio model simply returns the EFS data that are read from the database. This point of view unifies many typical operations with configurations and machine learning potentials into the general scheme shown in Figure 2(c).

The particular operational mode of MLIP is determined from the input settings file. The scope of the main operations with configurations and machine learning potentials determining the mode are listed in Section 7.1, the structure of the settings file is described in Section 7.6.

Generally the driver is some external package which uses MLIP as interatomic interaction model, however MLIP has two embedded drivers: (1) reading configurations from a database file and (2) relaxation (Section 7.3). When MLIP is operating under external driver it should be compiled as a part of the package providing that driver. This version of MLIP distributive provides an interface to LAMMPS package (see Section 4 describing compiling the joint software and Section 7.4 describing MLIP usage from LAMMPS).

MLIP also supports working with a number of internal and external EFS calculators:

1. DFT by VASP (external), see Section 7.2 for more details;
2. any empirical potential available in LAMMPS (external) described in the Section 7.2;
3. Lennard-Jones potential, usually used for testing purposes (internal);
4. internal dummy EFS "calculator", keeping the EFS data provided with the configurations in a database file;
5. another well-trained machine learning potential, which can be used, for example, for the fitting of "light" potentials (internal).

Communication between MLIP and external EFS calculators is implemented through the exchange of files. Internal AIMs are set in the settings file (details of configuring are given in Section 7.2).

In addition, MLIP provides a set of tools for self-testing, converting files into the ".cfg" format (for more details about the format see Section 6.1), parallel MTP fitting and parallel error calculation. These tools can be executed from the command line. All tools are described in a detail in Section 7.7.

7.1 Capabilities of MLIP

MLIP is a multifunctional tool communicating with a driver and an AIM as shown in Fig. 2. A particular operational mode is fully determined by the combination of the following options (each can be enabled or disabled independently from others).

O1. Calculation EFS for the configurations provided by the driver.

O2. Learning on the configurations (EFS for these configurations are queried from the ab initio model).

O3. Active selection of the configurations provided by driver;

O4. Comparison of EFS calculated by the potential and the AIM and accumulation of the average errors of approximation;

O5. Writing the processed configurations to a file (together with the EFS that are passed to the driver);

O6. Writing the log with the following information: the total number of configurations processed to the moment, the number of ab initio and MTP calculations of EFS.

All these options also have their own settings and parameters (that are described in detail in Section 7.6). Table 1 presents the most important scenarios that are defined by the combination of O1, O2 and O3. Options O4, O5, O6 provide the scenarios 1–8 from Table 1 with the additional functionality but do not change the scenarios themselves.

Note:

- When O4 is switched on, the EFS that are passed to the driver (the EFS predicted by the potential) are compared with ab initio EFS and the errors are calculated. In some scenarios (see Table 1) they are the same and hence the approximation error is zero.
- O5 writes the configurations together with their EFS that are passed to the driver. They may not coincide with ab initio EFS.

7.2 Communication with AIM

MLIP is able to query ab initio EFS data that are required in some scenarios (see Section 7.1). The current MLIP version is capable to communicate with the two external packages as AIMs:

- VASP implementing the DFT model (`https://www.vasp.at/`);
- LAMMPS that gives access to a large number of empirical potentials (`lammps.sandia.gov/`) useful mostly for testing purposes. Note that LAMMPS can be linked to MLIP in two roles: as the AIM and as the driver. The former interface is described below in Section 7.4.

The communication with the external packages is implemented through exchanging files as follows.

1. MLIP prepares an input configuration file in a special format (POSCAR for VASP, and dump-file for LAMMPS). Some other input files, required by the external package (e.g. INCAR and POTCAR for VASP), should be prepared manually.

Table 1: Description of the main scenarios defined by the combination of O1, O2 and O3 options

Mode	Calc. EFS (O1)	Learn (O2)	Select (O3)	Scenario description
1	no	no	no	EFS is calculated by AIM. No action is performed with MTP.
2	yes	no	no	Calculation mode. MLIP just calculates EFS by MTP for each configuration provided by the driver.
3	no	yes	no	Learning mode. MLIP fits MTP (according to the fitting settings) on the all the configurations provided by driver. The training is done on EFS provided by the AIM. The same EFS are returned to the driver.
4	no	no	yes	Selection mode. MLIP just selects (according to the selection settings) a D-optimal set of the configurations among all those provided by driver. EFS returned to the driver depend on the setting `mlip:select:efs_ignored`: if false then the EFS are taken from the AIM, otherwise EFS have unpredictable values (see Section 7.6 for more details).
5	yes	yes	no	Learning and calculation mode. The same as 3, but the EFS are calculated by MTP and returned to the driver.
6	yes	no	yes	Selection and calculation mode. The same as 4, but the EFS are calculated by MTP and returned to the driver.
7	no	yes	yes	Active learning mode. MTP is fitted on a D-optimal training set (according to the selection and fitting settings). EFS returned to the driver depend on the setting `mlip:select:efs_ignored`: if false then the EFS are taken from the AIM, otherwise EFS have unpredictable values (see Section 7.6 for more details).
8	yes	yes	yes	Learning on the fly mode. For each configuration given by the driver MLIP decides whether it needs to be learned by MTP. The EFS are calculated by MTP (after re-training, if necessary) and returned to the driver. (If `mlip:select:efs_ignored` is true then ab initio EFS are returned whenever a configuration is trained on, see also Section 7.6.)

2. MLIP invokes the ab initio package through a system call with the command specified in the settings file `mlip.ini` (all other input files, such as pseudopotentials, required by the external package, should be already prepared);

3. After the system call has finished, MLIP reads the EFS data from the output file (OUTCAR in the case of VASP, and the dump-file in the case of LAMMPS).

In addition to the external ab initio packages MLIP also has three internal EFS calculators:

- internal dummy EFS "calculator" used only with configuration reading driver (see Section 7.3), preserving the EFS data as read from the configuration database;
- Lennard-Jones potential, usually used for testing purposes (internal);
- another well-trained machine learning potential, which can be used, for example, for the fitting of "light" potentials (internal).

Selecting a specific model and setting its parameters is configured in the settings file (see Section 7.6).

7.3 Internal drivers

The current MLIP version has two internal drivers:

1. reading configurations from a file;
2. internal structure relaxation driver.

Both drivers can be chosen and configured from the settings file (see Section 7.6). To run the internal driver the following command should be executed from the command line:

```
> ./mlp run mlip.ini
```

Here:

`mlp` is the name of executable (it should be build as stand-alone package, see Section 4 for more details),
`run` is a special command word in mlp (other commands are described in Section 7.7),
`mlip.ini` is the settings file configuring the driver and MLIP settings.

7.4 Integration with the LAMMPS driver

LAMMPS is a popular code implementing a broad range of algorithms for atomistic simulations. The MLIP package provides an interface to LAMMPS that allows using practically all drivers from LAMMPS with our machine learning potentials. The interface is implemented as an optional user-provided potential for LAMMPS (for more information see compiling and installation details in Section 4).

From LAMMPS, MLIP looks like a "pair potential" (even though MTP is a many body potential!). Thus, to use MLIP as an interatomic potential one should set in the LAMMPS script the following pair style:

```
pair_style    mlip  mlip.ini
pair_coeff    * *
```

The MLIP's operating mode is determined by the settings from `mlip.ini` file (for more information see Section 7.6), as specified in the `pair_style` command. The examples of the LAMMPS input script, `mlip.ini` and the MTP file can be found in the `doc/examples/` folder of the MLIP package and described in the Section 5.

Note:

- The current version of MLIP does not support some of LAMMPS features. For example, trying to resume a computation after writing a restart file may lead to incorrect behavior. Some of the other known limitations are mentioned in Section 7.5.
- LAMMPS can be used by MLIP in two roles: as a driver and as an AIM. The former interface is described in Section 7.2.

7.5 Serial and parallel modes

The current implementation of the internal drivers supports only a serial version, whereas the interface to LAMMPS supports also a parallel version. Parallelization is performed by LAMMPS and is the same as for any other "pair" potentials. However, the "serial" and "parallel" versions have partially different capabilities and limitations, see Table 2. Thus, for treating small and moderate-size configurations, the serial version may be preferable.

Table 2: Comparison of capabilities and limitations of the two MLIP implementations

Version	Compatibility with serial LAMMPS	Compatibility with parallel LAMMPS	Learning on the fly	Configuration selection	Error calculation	I/O, logging
Serial MLIP	+	−	+	+	+	+
Parallel MLIP	+	+	−	−	−	−

By default, MLIP is linked in the serial mode in LAMMPS. To enable linking in parallel mode the following requirements should be met:

1. MLIP is compiled with MPI;
2. no fitting is configured in settings;
3. no selection is configured in settings;
4. no ab initio calculation is configured in settings;
5. no error calculation is configured in settings;
6. no logging is configured in settings;
7. EFS calculation is enabled;

If parallel mode is activated the message `MLIP linked in neighborhoods mode` will appear, otherwise the message `MLIP linked in configuration mode` will appear.

7.6 Settings file

All settings determining the operational mode of MLIP are set in the configuring file typically named as `mlip.ini`. This file is organized according the following rules:

- Each setting parameter must be given in a separate line and have the form

  ```
  identifier    value
  ```

  where `identifier` is the string without whitespace and `value` is another string without whitespace (a string can be a number). They must be separated by any number of spaces or tab symbols. Whitespace before `identifier` as well as any symbols after `value` are allowed but ignored by the parser.

- Other than identifiers and values, the lines of the settings file can contain comments. Comments begin with "`#`" and all the foregoing symbols are ignored in this line.

- Settings have a tree-like structure in which the parameters are grouped in blocks. Each block contains the settings and parameters for a particular module (e.g., fitting). The names of the parameters in a block start with the same prefix that corresponds to the name of the block. Each block can be either active or inactive. The parameters related to inactive blocks have no effect. In the example below, the blocks are highlighted by indents only for convenience (i.e., whitespace does not matter).

- The order of lines is not important; the settings are grouped for convenience.

- If a parameter is not present (or commented out), its default value is used. The blocks are switched off by default.

- If a setting is not supported (i.e. has no association with the internal variable) it has no effect (without warning).

An example of the settings file

```
abinitio                                        void
# void - EFS data should be provided
# lj - use embedded Lennard-Jones pair potential
    abinitio:lj:r_min                       2.0
    abinitio:lj:scale                       1.0
    abinitio:lj:cutoff                      5.0
# vasp - VASP
    abinitio:vasp:poscar                    vasp_tmp/POSCAR
    abinitio:vasp:outcar                    vasp_tmp/OUTCAR
    abinitio:vasp:start_command             ./vasp
# lammps - LAMMPS
    abinitio:lammps:onput_file              dump.inp
```

```
        abinitio:lammps:output_file             dump.out
        abinitio:lammps:start_command           ./lmp.sh
    # mtp - use pre-trained MTP
        abinitio:mtp:filename                   MTP999_83.mtp

mlip                                            MTP
mlip:load_from MTP.mtp
        mlip:calculate_efs                      1
        mlip:fit                                1
                mlip:fit:save                   MTP.mtp
                mlip:fit:energy_weight          1.0
                mlip:fit:force_weight           0.001
                mlip:fit:stress_weight          0.1
                mlip:fit:scale_by_force         0.0
                mlip:fit:log                    fitting.log
        mlip:select                             1
                mlip:select:site_en_weight      1.0
                mlip:select:energy_weight       0.0
                mlip:select:force_weight        0.0
                mlip:select:stress_weight       0.0
                mlip:select:threshold           1.1
                mlip:select:save_ts             TS.cfgs
                mlip:select:save_state          selection.mvs
                mlip:select:load_state          selection.mvs
                mlip:select:efs_ignored         false
                mlip:select:log                 selection.log
        mlip:check_errors                       errors.log
        mlip:write_cfgs                         record.cfgs
                mlip:write_cfgs:skip            0
        mlip:log                                lotf.log

driver                                          1
    # 0 - no driver or external MD driver (e.g. LAMMPS)
    # 1 - read configurations from database file
        driver:cfg_reader:filename              Li_NVT300.cfgs
        driver:cfg_reader:limit                 1
        driver:cfg_reader:log                   stdout
    # 2 - relaxation
        driver:relaxation:pressure              0.0
        driver:relaxation:iteration_limit       500
        driver:relaxation:min_dist              1.0
        driver:relaxation:force_tolerance       0.0001
        driver:relaxation:stress_tolerance      0.001
        driver:relaxation:max_step              0.5
        driver:relaxation:min_step              1.0e-8
        driver:relaxation:bfgs_wolfe_c1         1.0e-3
```

```
driver:relaxation:bfgs_wolfe_c2         0.7
driver:relaxation:input_cfgs_filename   4Relax.cfg
driver:relaxation:save_relaxed          relaxed.cfgs
driver:relaxation:log                   relaxation.log
driver:relaxation:save_unrelaxed        unrelaxed.cfg
```

Another example of settings file can be found in `doc/mlip.ini`. This file contains the full list of recognizable settings and comments explaining the meaning of the commands. All the settings are also listed in the Table below

Identifier	Type	Description
`abinitio`	string	Switches between embedded ab initio models: "void" - No EFS change (configurations should be provided with EFS); "lj" - embedded Lennard-Jones pair potential; "vasp" - VASP; "lammps" - LAMMPS; "mtp" - pretrained MTP
`abinitio:lj:r_min`	double, positive	Distance to the minimum of the pair potential function (in Angstroms)
`abinitio:lj:scale`	double	Value of the pair function at minimum (in eV)
`abinitio:lj:cutoff`	double, positive	Cutoff radius (in Angstroms)
`abinitio:vasp:poscar`	string	"POSCAR" with (optionally) path to VASP calculaion folder
`abinitio:vasp:outcar`	string	"OUTCAR" with (optionally) path to VASP calculaion folder
`abinitio:vasp:start_command`	string	System command that starts VASP (Hint: it can be useful to specify here the command for launching a script that launches VASP)
`abinitio:lammps:input_file`	string	File with the configuration to be read by LAMMPS
`abinitio:lammps:output_file`	string	File with the configuration and EFS data to be read by MLIP
`abinitio:lammps:start_command`	string	System command for starting LAMMPS (Hint: it can be useful to specify here the command for launching a script that launches LAMMPS)
`abinitio:mtp:filename`	string	MTP file name that is used as AIM
`mlip`	string	"MTP" or "void". If "void" specified then the driver operates directly with AIM (without additional MLIP routines)

`mlip:load_from`	string	Filename with MTP
`mlip:calculate_efs`	bool	Enables/disables EFS calculation by MTP
`mlip:fit`	bool	Enables/disables MTP fitting
`mlip:fit:save`	string	MTP output file name (for the fitted MTP)
`mlip:fit:scale_by_force`	double	Coefficient ϵ_f in (2) if greater then zero, otherwise MTP is fitted according (1)
`mlip:fit:energy_weight`	double	Coefficient C_E in (1) or (2)
`mlip:fit:force_weight`	double	Coefficient C_f in (1) or (2)
`mlip:fit:stress_weight`	double	Coefficient C_σ in (1) or (2)
`mlip:fit:log`	string	File/stream to write the fitting log. No logging if not specified; ”stdout” and ”stderr” corresponds to standard output streams; otherwise the log will be saved to the file with that name
`mlip:select`	bool	Activates/deactivates selection (active learning engine)
`mlip:select:site_en_weight`	double	Weight for site energy equations in selection procedure
`mlip:select:energy_weight`	double	Weight for energy equation in selection procedure
`mlip:select:force_weight`	double	Weight for forces equations in selection procedure
`mlip:select:stress_weight`	double	Weight for stresses equations in selection procedure
`mlip:select:threshold`	double, ≥ 1	Selection threshold - maximum allowed extrapolation level
`mlip:select:save_selected`	string	Selected configurations will be saved in this file after selection is complete. No configurations are saved if not specified.
`mlip:select:save_state`	string	The state of the selection will be saved in this file after selection is complete (or not saved if not specified). Can be used for restarting selection/learning on the fly

`mlip:select:load_state`	string	The state of the selection will be loaded (if specified) from this file before selection starts. Can be used for restarting selection/learning on the fly
`mlip:select:efs_ignored`	bool	Indicates that the driver actually does not need EFS to be calculated (e.g. in the fitting scenario). "true" value may speed up processing by skipping some extra EFS calculations
`mlip:select:log`	string	Where to write the selection log. No logging if not specified; "stdout" and "stderr" corresponds to standard output streams; otherwise the log will be saved to the file with that name
`mlip:check_errors`	string	If present, enables comparison and accumulation of error statistics for EFS calculated by AIM and MTP. Requires that AIM is specified in the settings. The value indicates where to write the log of learning on-the-fly. No logging if not specified; "stdout" and "stderr" corresponds to standard output streams; otherwise the log will be written to the file with that name
`mlip:write_cfgs`	string	File for writing all(or some, depending on `mlip:write_cfgs:skip`) processed configurations. No confuguration recording if not specified
`mlip:write_cfgs:skip`	int	Skip this number of configurations while writing
`mlip:log`	string	Where to write MLIP log. No logging if not specified; "stdout" and "stderr" corresponds to standard output streams; otherwise the log will be saved to file with that name
`driver`	integer, 0–3	Defines the configuration driver. Ignored if MLIP is linked to an external driver (e.g., LAMMPS). 0 - no driver or external MD driver; 1 - read configurations from database file; 2 - embedded relaxation
`driver:cfg_reader:filename`	string	Configuration file name
`driver:cfg_reader:max_count`	int	Maximal number of configurations to read
`driver:cfg_reader:log`	string	Where to write the reading log. No logging if not specified; "stdout" and "stderr" corresponds to standard output streams; otherwise the log will be saved to file with that name

`driver:relaxation:pressure`	double	External pressure (in GPa). If not zero enthalpy is minimized
`driver:relaxation:iteration_limit`	int	Maximal number of iteration of the relaxation process
`driver:relaxation:min_dist`	double	Minimal interatomic distance constraint (in Angstroms)
`driver:relaxation:force_tolerance`	double	Forces on atoms in relaxed configuration should be smaller than this value (in eV/Angstroms). Zero or negative value disables atoms relaxation (changing fractional coordinates of atoms)
`driver:relaxation:stress_tolerance`	double	Stresses in relaxed configuration should be smaller than this value (in GPa). Zero or negative value disables supercell relaxation (changing lattice vectors of the supercell)
`driver:relaxation:max_step`	double	Maximal allowed displacement of atoms and lattice vectors in Cartesian coordinates (in Angstroms) during a single step
`driver:relaxation:min_step`	double	Minimal displacement of atoms and lattice vectors in Cartesian coordinates (in Angstroms) during a single step. If all actual displacements are less then the relaxation stops
`driver:relaxation:bfgs_wolfe_c1`	double	Wolfe condition constant on the function decrease (linesearch stopping criterea)
`driver:relaxation:bfgs_wolfe_c2`	double	Wolfe condition constant on the gradient decrease (linesearch stopping criterea)
`driver:relaxation:cfg_filename`	string	Filename with configurations for the relaxation
`driver:relaxation:save_relaxed`	string	Filename for output results of relxation (configurations). No configuration will be saved if not specified
`driver:relaxation:log`	string	Where to write the relaxation log. No logging if not specified; if ”stdout” and ”stderr” corresponds to standard output streams; otherwise the log will be saved to file with that name

All settings are divided into three main blocks:

1. The `abinitio` block defining ab initio model;

2. The `mlip` block defining the operational mode of MLIP;
3. The `driver` block defining configuration driver (make sense only if running under internal driver);

There is a number of different available ab initio models (see Section 7.2):

0 ab initio model is not used,

1 dummy EFS calculator extracting EFS from .cfg file (applicable only with configuration reading driver),

2 embedded Lennard-Jones potential,

3 VASP (external),

4 LAMMPS (external),

5 pre-learned MTP,

and internal drivers (see Section 7.3):

0 no driver or external driver,

1 reading configurations from a file,

2 embedded relaxation driver.

The choice of a particular number after `abinitio/driver` activates the corresponding model/driver. `abinitio` and `driver` blocks can be missing from the settings file (that is equivalent to the "0" option) if a scenario does not require ab initio EFS (for example when MLIP just calculates EFS) or driven by an external driver.

7.7 More stand-alone tools

`mlp` is a self-explanatory binary that embeds several commands (tools). To list all the commands, execute "`mlp list`". To get an extended help page on a command, execute "`mlp help <command>`". The list of commands embedded into `mlp` include:

`convert-cfg` (serial only): a command that converts configuration files from one format (textual, binary, vasp outcar, vasp poscar) into another

`train` (serial and MPI): fits an MTP

`calc-errors` (serial and MPI): compares EFS calculated by MTP with values stored in the database

`self-test` (serial and MPI): performs a number of unit tests

8 Feedback

If you run into problems with MLIP or if you have a some related questions, please contact Alexander Shapeev (a.shapeev@skoltech.ru).

References

[1] E. V. Podryabinkin and A. V. Shapeev. Active learning of linear interatomic potentials. *arXiv preprint arXiv:1611.09346*, 2016.

[2] A. Shapeev. Moment tensor potentials: a class of systematically improvable interatomic potentials. *Multiscale Model. Simul.*, 14(3):1153–1173, 2016.