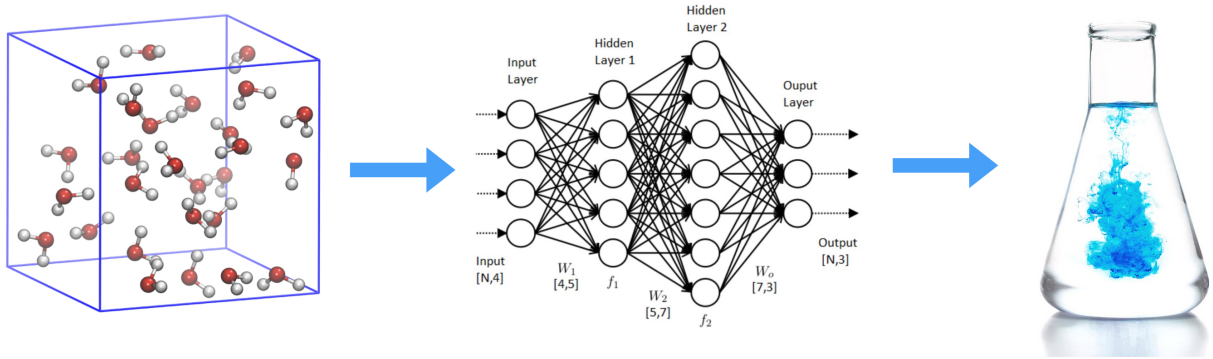
Molecular Dynamics meets Neural Networks: TUTORIAL



Veronika JURÁSKOVÁ
Frédéric CELERSE
Rubén LAPLAZA
Clémence CORMINBOEUF

Contents

1 Introduction

This tutorial aims to guide the building of neural network-based potentials (NNPs) and their application in molecular dynamics (MD). As the *ab initio* MD is too computationally demanding to obtain statistically converged simulations of chemical reactions and processes in large systems, the MD with NNPs benefits from the much lower cost of the simulation while the accuracy of the AIMD is conserved.

Part 1 of this tutorial is dedicated to the generation of the Behler-Parrinello NNPs, including the preparation of a robust data set, selection of the descriptors, and monitoring of the accuracy of the training. **Part 2** demonstrates how to use the NNPs using the i-PI interface to LAMMPS and other codes. **Part 3** introduces the concept of baselined NNPs, showcases its advantages compared to NNPs trained directly on DFT data, and provides a manual to multiple-time-step scheme with NNPs.

2 Neural Networks-based potentials: Crash course

Neural Networks are statistical models which can reproduce complex non-linear functions using large amounts of the training points. The availability of so-called *Big data* in a broad range of human activities resulted in the harvesting of neural networks in the development of game-playing models, speech and text recognition software, or analysis of patients' data in modern health care, to name a few. Recently, the power of NN to reproduce highly complex data has been exploited in the development of a new generation of potentials for molecular dynamics. The so-called Neural Network-based Potentials (NNPs) are trained to mimic the shape of the corresponding potential energy surface using the **data sets** containing coordinates and *ab initio* energy and possibly forces. The resulting NNPs are then used to propagate molecular dynamics with the accuracy of the underlying *ab initio* method but with a fraction of its original cost. Several NN architectures to construct the NNPs have been proposed during the last years. In this tutorial, we focus on Behler-Parrinello NNPs based on feedforward NN.

2.1 Feedforward Neural Networks

The example of the feedforward NN is depicted in Fig. 1. It is made from several nodes organized in interconnected layers. The potential energy E corresponds to the node in the **output layer** and the atomic coordinates of a frame are provided in the **input layer** as vectors of input coordinates $G = \{G_i\}$. The input and output layers are connected via several so-called **hidden layers** with no physical meaning. The number of layers and the nodes define the flexibility of the final NN. Each node is connected to the nodes of the next layer with a specific weight w_{ij}^{kl}, where i is the index of a node of layer k, which is connected to the node with index j of layer l. For instance, the w_{11}^{01} connects the nodes G_1 ($k = 0$, $i = 1$) and x_1^1 ($l = 1$, $j = 1$). Nodes in hidden and output layers are connected to a so-called bias node, which scales their values by a bias weight b_i^j, where j is the layer of the target node and i is the node index. In our scheme, the bias node is omitted for simplicity.

The value x_i^j of any node is computed as:

$$x_i^j = f_i^j(b_i^j + \sum_{k=1}^{N_{j-1}} w_{k,i}^{j-1,j} x_k^{j-1}) \tag{1}$$

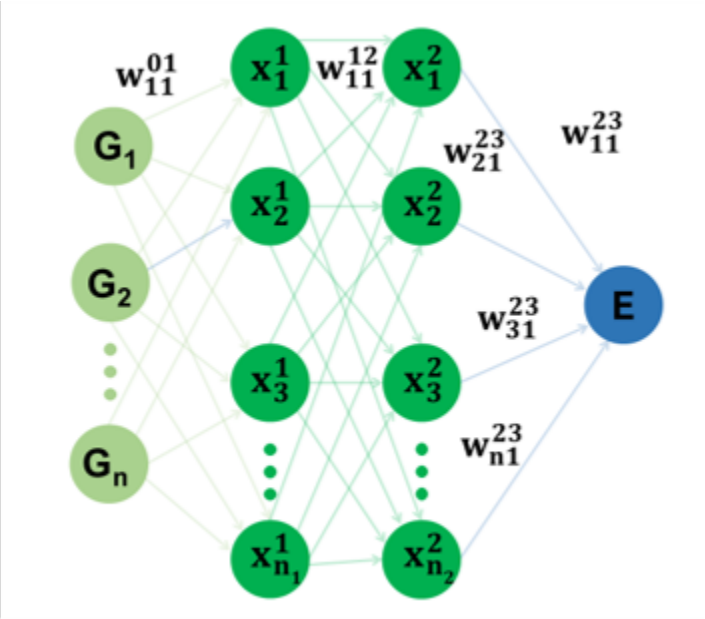


Figure 1: Schematic representation of a small feedforward NN. Nodes are interconnected in order to establish a functional relation between a set of coordinates G (input layer) and the potential energy of the structure E (output layer).

The equation is a linear combination of the values of all nodes in the previous layer scaled by the bias weight. The term f_i^j represents the **activation function** of the NN. The activation function ensures the ability to fit complex non-linear functions. The examples of activation functions are:

- hyperbolic tangent,
- sigmoid function,
- Gaussian function,
- Softplus
- trigonometric functions
- exponential function.

2.2 High–dimensional NNP

Feedforward neural networks discussed in the previous section suffer from several drawbacks preventing their application in the simulation of complex chemical systems. The limitations are, for instance:

1. absence of symmetry between equivalent terms (for instance, the OH bonds of a water molecule)
2. system size dependency: no atoms could be deleted or added as it causes problems in the connecting weights.

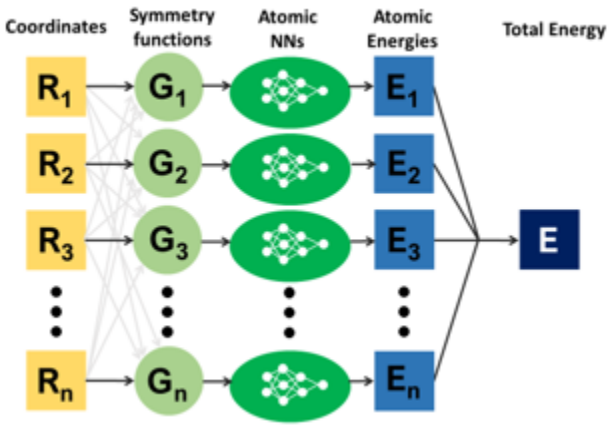


Figure 2: High dimensional NN scheme: Atomic positions are represented by a vector of symmetry functions G_i.

In 2007, Behler and Parrinello proposed a solution to this problem by expressing the total energy E_s as a sum of the atomic energy contributions E_i.[1]

$$E_s = \sum_{i=1}^{N_{atom}} E_i \tag{2}$$

Starting from the Cartesian coordinates R, every atom is represented by a set of atom-centered symmetry functions G_i, which describe the atoms' chemical environment within a given cutoff. Each atom is then assigned a feedforward NN yielding the atomic contribution to the total energy. The weights of the NN are identical for atoms of the same element to ensure that the trained NNPs do not depend on the number and order of the atoms.

2.3 Atom-centered symmetry functions

Together with the NN topology, the seminal work of Behler and Parrinello introduced also new types of symmetry functions to characterize the local environment of each atom. The so-called Atom-centered symmetry functions (ACSFs)[2] are constructed from the positions of the atoms and their closest neighbors. This representation guarantees that the atoms with the same environment yield the same atomic energy contribution which is also translationally and rotationally invariant.

The typically used ACSFs are two-body radial and three-body angular functions in a form:

$$G_i^2 = \sum_i e^{-\eta(R_{ij}-R_s)^2} \cdot f_c(R_{ij}) \tag{3}$$

$$G_i^3 = 2^{(1-\xi)} \sum_{j,k\neq i}^{\text{all}} (1+\lambda \cos\theta_{ijk})^{\zeta} \cdot e^{-\eta(R_{ij}^2+R_{ik}^2+R_{jk}^2)} \cdot f_c(R_{ij}) \cdot f_c(R_{ik}) \cdot f_c(R_{jk}), \tag{4}$$

where f_c is a cutoff function ensuring the smooth decay of the symmetry function to zero.

In this tutorial, we use the original Behler-Parrinello ACSFs. However, during the last years, modifications to the original ACSFs were proposed together with new functional forms

to describe the atomic environments. The different structural representations are summarized in Ref. 3.

2.4 Constructing and training NN

The general protocol to construct NNP is as follows:

1. **Define an initial set of structures and compute the reference energies and forces.** The electronic structure method used as a reference should be accurate enough to describe the studied problem, but also suitable to perform thousands of computations of large systems.

2. T**rain the first version of NNPs.** The fraction of the data set is used in the NNP training (so-called training set) while the remaining part serves as a test set to validate the error on unseen data. Ideally, several different NNP with different training sets should be trained to identify potential problems in the training set.

3. **Perform preliminary simulations using the NNPs.** The stability of the NNP should be verified and underrepresented areas of the PES should be added to the training set, *e.g.* structures triggering extrapolation warnings or unphysical geometries.

4. Repeat the validation and re-training of the NNP until no instabilities are present.

For a detailed general tutorial review on Behler-Parrinello NN, see Ref. 4.

3 Requirements

This tutorial presents a workflow that requires an extensive number of different software packages, scripts, and libraries. The versions of the codes used here are listed below.

3.1 Software

All the codes are available at GitHub (see the last section of tutorial):

- n2p2 (version 1.0.0)
- LAMMPS (version 2)
- i–PI (version 2.0)
- Tinker (version 8.8)
- cp2k (version 6.1)

3.2 Dependencies

- python (version 2.7 and 3.6)
- libraries intel, intel-mpi, intel-mkl, gsl, eigen, gc,c mvapich2, openblas, n2p2/lib

4 Part 1: How to generate neural network potentials

Instructions

Currently, there is no universal NNP that is applicable to model all classes of systems. Despite the ongoing efforts in the training of general neural network-based force fields, many systems require building the NNPs from scratch. In the following section, we provide general instructions on how to train NNP using the Behler-Parrinello approach. The required steps include:

1. Sampling of the phase space to produce initial training structure;
2. Generating a large pool of symmetry functions;
3. Identification of a representative set of fingerprints;
4. Careful selection of a small set of structures for the training of NN;
5. Computation of reference forces and energies for the selected structures;
6. Training of the NN.

The training protocol is summarized in the Fig. 3.

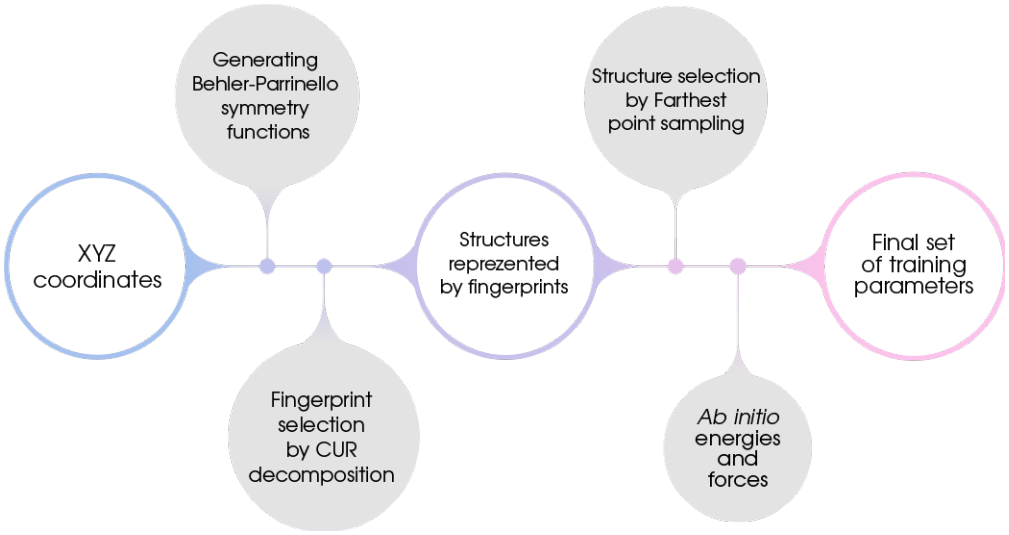


Figure 3: Scheme of the NNP preparation.

4.1 Step 0: The system

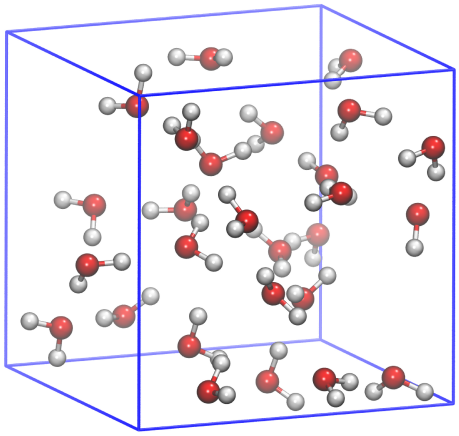


Figure 4: System containing 27 molecules in a cubic box with edge of 9.32 Å.

As a relatively simple test case, we train the NNPs for a small water box containing 27 water molecules with a cell size of 9.32 Å. These systems are composed of only two elements, *i.e.* H and O, which significantly simplifies the training.

4.2 Step 1: Generating the data

Files needed

- system.xyz
- system.key
- system.dyn
- water03.prm
- *dynamic* module

The first step in the tutorial is the preparation of the training set. The training set can contain systems of different sizes and different compositions. To keep things simple, we will train the NNPs using only one size of the water box. To generate an extensive set of the structures, it is beneficial to use molecular dynamics starting either from equilibrated structure or from experimental data (*e.g.*, X-ray). Random displacement of the atoms could be also applied.

As the first training set should be large enough and contain representative geometries, the potential for the exploratory dynamics should provide a reasonable cost/accuracy ratio. For example, classical force fields might not be the best choice, as they can sample structures that are too different from the ones obtained at the reference level. The suitable compromise is, therefore, the application of semiempirical methods or polarizable force fields.

Herein we generate the system using AMOEBA polarizable force-field as available in Tinker. The files required for the simulation are *.xyz, *.key, and *.prm. For more information on the preparation of the files for Tinker, see tutorials listed in section 7.

The files contain:

- The *.xyz file - the Cartesian coordinates of the initial water box in the Tinker xyz format (Caution! Tinker xyz differs significantly from standard xyz or extended xyz formats.)
- The *.key file - the settings to run the dynamics.
- The *.dyn - positions, velocities, and accelerations of the last frame of the dynamics.
- The *.prm file - AMOEBA parameters (all the terms to compute the potential energy of a specific structure)

MD using Tinker can be launched as:

```
dynamic system 1000000 2.0 1 2 300 > system.out
```

This command means that the simulation will run using the *dynamic* module of Tinker for 1000000 steps with a time step of 2.0 fs (*i.e.*, the total length of the simulation is 2 ns). The next term specifies the frequency (in ps) of saving the trajectory to the *.arc file (here we save it every 1 ps). The final *system.arc* file will thus contain 2000 frames. The following term specifies the mode of the dynamics, 2 corresponds to NVT canonical ensemble. The last term corresponds to the simulation temperature (300 K).

Enhanced sampling

⚠As the system is quite simple, the standard molecular dynamics are sufficient to provide an extensive sampling. In more complex cases, however, the simulations starting from different initial structures or using enhanced sampling techniques might be needed to ensure that all the relevant structures are present in the training set. Suitable simulations techniques include, for instance:

1. Replica exchange molecular dynamics
2. Sampling techniques using bias, such as metadynamics, steered molecular dynamics, umbrella sampling, ...
3. Transition path sampling

Extending the database with more distorted structures

⚠ The NNPs have in general very limiting extrapolation capacity. As they are trained only on the information included in the training set, it is beneficial to consider more distorted structures covering the repulsive and dissociative parts of the PES. Distorted structures can be generated by simulations using, for example:

1. High temperature
2. High pressure
3. Path integral molecular dynamics
4. Specific constraints and restraints
5. Random displacement

Output Files

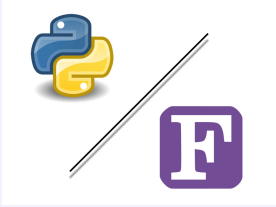
- system.out
- **system.arc** (file containing the structures)
- system.dyn

4.3 Step 2: Descriptor selection

To assign the structures with a suitable set of descriptors, we generate a large pool of the ACSFs and select a small representative set using, for example, CUR decomposition.

4.3.1 Generating symmetry functions

Input Files, Executables and Scripts



- system.arc
- from_txyz_to_n2p2.f90
- symmetry_functions.py

As a first step, place the structures generated in the previous part to the *system.arc* file. Structures in this file can be converted to input files for n2p2 as follows:

1. compile the fortran code from_txyz_to_n2p2.f90: *gfortran from_txyz_to_gen.f90 -o from_txyz_to_gen*
2. place the system.arc file in the same directory of the from_txyz_to_gen executable and launch it as *./from_txyz_to_gen*

The script generates input.data file for the n2p2. The definition of each frame starts with a short header:

```
begin
comment
lattice 17.612 0.0 0.0
lattice 0.0 17.612 0.0
lattice 0.0 0.0 17.612
```

"lattice" defines the dimension of the box in Bohr. Each atom of the structure is defined as:

```
atom     -4.367    5.267     4.603   O   0 0   0.0  0.0  0.0
```

The keyword "atom" specifies that the line corresponds to one atom, -4.367, 5.267, 4.603 are the xyz coordinates of the atom in Bohr, O is the element label, the following two columns 0 0 are not used. Finally, the last three columns 0.0 0.0 0.0 are the xyz force components acting on the atom. In this example, forces are set to 0.0 since they are not used.

The frame ends with:

```
energy 0.000
charge 0.0
end
```

where "energy" is the reference total energy of the system, "charge" is the total charge of the system, and "end" closes the definition of the structure.

Units in n2p2

⚠ As indicated on the n2p2 website, the units correspond to the physical units defined in the input files. As we combine several codes with possibly different unit specifications, keep track of the units and stay consistent within the workflow.
In the inputs for n2p2, we use:

1. lattice: **Bohr**
2. x/y/z positions: **Bohr**
3. x/y/z forces: **Hartree Bohr^{-1}**
4. energy: **Hartree**

The conversion from Angström to Bohr is done by a multiplication by a factor **1.88973**

All settings and parameters for the training of NNP are specified in **input.nn** file. It contains three main parts:

1. GENERAL NNP SETTINGS

2. ADDITIONAL SETTINGS FOR TRAINING

3. SYMMETRY FUNCTIONS

GENERAL NNP SETTINGS specify the general set-up of the neural network, for instance, a number of the elements, layers and nodes, cut-offs, scaling normalization, etc.

ADDITIONAL SETTINGS FOR TRAINING define explicitly the parameters of the training as the number of epochs, fraction of energies used in the training, and type of the optimizing algorithm.

The last part defines the symmetry functions sets used in training. While the parameters for the training can be found in the literature and possible modification directly depend on the simulated system, the symmetry functions are often built completely from scratch and require careful selection.

The n2p2 currently supports Behler-Parrinello types of symmetry functions (radial, angular, wide angular, for definition see Ref. 2), their weighted variants (Ref. 5) and polynomial symmetry functions (Ref. 6). In this tutorial, we use standard Behler-Parrinello functions.

The initial set of the symmetry function can be generated, for example, by a *sym-func_paramgen.py* code by Florian Buchner available at https://github.com/flobuch/n2p2/tree/symfunc_paramgen. Jupyter notebook with an example of the use of the code is provided at https://github.com/flobuch/n2p2/blob/symfunc_paramgen/tools/python/symfunc_paramgen/examples/example.ipynb.

For the NNP of the water box, we generate a set of radial symmetry functions with cutoff $r_c = 4, 8, 12$ Bohr and $N = 8$ and two sets of angular symmetry functions following the protocol described in Ref. 7. Store the symmetry functions in one file *e.g.*, asfs.txt and append it at the end of the input.nn.

cat asfs.txt >> input.nn

The initial input files needed for the training are therefore:

- **input.data**
- **input.nn**.

Output Files

- **input.data**
- **input.nn**

4.3.2 Selection of a subset of descriptors

Input Files, Executables and Scripts

- input.data
- input.nn
- nnp–scaling

Using the symfunc_paramgen tool, we generated 192 radial and 768 angular symmetry functions, *i.e.*, 480 SF per element. The goal of this section is to select a small subset of the SF that will be used for the training of the NNPs. Following the benchmark and recommendations discussed in Ref. 7, we use the CUR decomposition scheme to select 64 unique symmetry functions per element. The details of CUR and more advanced CovCUR techniques can be found in Ref. 7–9.

To evaluate the symmetry functions over the geometries, we use *nnp-scaling* tool in n2p2. However, the number of the symmetry function is huge and the storage of the values over the whole set of structures would be memory demanding. We therefore randomly select a subset of structures from the original input.data file using *nnp-select* tool.

```
nnp-select random 0.05 123
```

Keyword *random* indicates the random selection of the structures, 0.05 corresponds to the percentage of selection, and 123 is the seed for the random generator. Modify the percentage so it corresponds to roughly 1000 structures.

Finally, evaluate the symmetry functions by *nnp-scaling*. Regarding the computational cost, it is advisable to submit the scaling on cluster in parallel, for example:

```
mpirun -n 8 nnp-scaling 500 > logfile
```

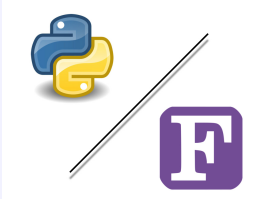
nnp-scaling generates three output files:

1. **logfile**: contains a list of the symmetry functions and their parameters. **Needed for the CUR selection.**
2. **function.data**: the evaluation of all the symmetry functions over the data in *input.data*. **Needed for CUR selection.**
3. **scaling.data**: statistic of the symmetry functions, *i.e.*, the minimum and maximum values, mean and sigma.

TO BY ADDED!!

4.4 Step 3: Hybrid DFT force–energy computations

Input Files, Executables and Scripts



- landmarks.dat
- input.data
- selection.f90
- from_data_to_xyz.f90
- from_xyz_to_cp2k.f90

From the landmarks.dat file generated in the previous step, we need to extract structures for the single point reference computations, for example using selection.f90:

gfortran selection.f90 -o selection ; ./selection

It will generate a new file **new–input.data** containing the 1000 structures selected by the FPS. In this tutorial, we use cp2k code to generate energy and forces of the water box. However, the reference computations for the NNP in general do not need to be periodic. The geometry input files for cp2k can be generated by:

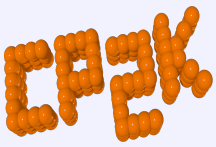
1. *gfortran from_data_to_xyz.f90 -o from_data_to_xyz ; ./from_data_to_xyz*
2. *gfortran from_xyz_to_cp2k.f90 -o from_xyz_to_cp2k ; ./from_xyz_to_cp2k*

Output Files

- **$i-cp2k.xyz**

4.4.1 CP2K computations

Input Files, Executables and Scripts



- $i-cp2k.xyz
- input.cp2k
- prep-file.sh

The cp2k computations can be automatized by prepare.sh script which automatically creates directories for the computations (one per structure). The directories contain geometry files and inputs to cp2k. Here, we use the CP2K 6.1 version to perform reference energies and forces computations at the PBE–D3BJ level of theory. All elements are described with the TZV2P–MOLOPT basis set with cores represented by the dual–space Goedecker–Teter–Hutter pseudopotentials (GTH PBE). The plane–wave cutoff is set to 700 Ry with a relative cutoff of 70 Ry.

Launch a cp2k computations following the installation of your machine. For example:

cp2k.popt -i input.cp2k -o output.cp2k

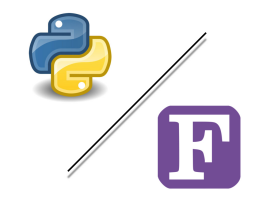
The output of the computation including energy and forces is in the output.cp2k file.

Output Files

- **output.cp2k**

4.4.2 Extraction of energy and forces

Input Files, Executables and Scripts



- output.cp2k
- process.sh

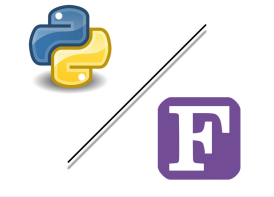
The database of the energy and forces can be created using the process.sh scrip. The resulting information is saved in the database_forces-pbe0.xyz file.

Output Files

- **database_forces-pbe0.xyz**

4.4.3 Final input.data file

Input Files, Executables and Scripts



- input.data
- database_forces-pbe0.xyz

Finally, with the database of energies and forces for the training structures, we create the final input.data file which is used for the training. This can be done, for instance, by the XXX code.

gfortran update.f90 ; ./a.out

Output Files

- **input.data**

4.5 Step 4: Neural Network Potentials

Input Files, Executables and Scripts

- input.data
- input.nn
- nnp–scaling
- nnp–train

The last file we need to prepare for the training of the neural network is the preparation of the new scaling.data file. As in the previous case, we generate it using nnp-scaling in n2p2.

Finally, the training procedures can be launched. Depending on the system, the process can be parallelized and submitted, for example, as:

```
mpirun -n 32 nnp–train > output.txt
```

The neural network potentials are trained to achieve as low RMSE on the test set as possible. The data to monitor the training behavior of the NNP are provided in the *learning-curve.out* file. The file contains:

1. Column 1: The index of the current epoch
2. Column 2: RMSE of training energies per atom (physical units)
3. Column 3: RMSE of test energies per atom (physical units)
4. Column 4: RMSE of training forces (physical units)
5. Column 5: RMSE of test forces (physical units)

The physical units correspond to the units used in the training data. In our case, energy is specified in Hartree (a.u.) and forces in Hartree per Bohr. The training errors discussed in the literature are commonly specified in the meV and meV/$\mathring{A}$ or kcal/mol and kcal/mol/$\mathring{A}$. The conversion factors are: 1 a.u = 627.5 kcal/mol = 27.211 eV, 1 Bohr = 0.529177 $\mathring{A}$. The learning curves can be plotted in Gnuplot to visualize the training progress. For example, the plotting of the RMSE in error in energy for the training set and test set:

```
p 'learning-curve.out' u 1:($2*1000*27.11) w l, 'learning-curve.out' u 1:($3*1000*27.11) w l
```

The Behler-Parrinello NNP can often achieve RMSE around 1 meV/atom in energy and less than 100 meV/$\mathring{A}$ in forces. The RMSE obtained for the water box show a similar trend and can be tested in molecular dynamics.

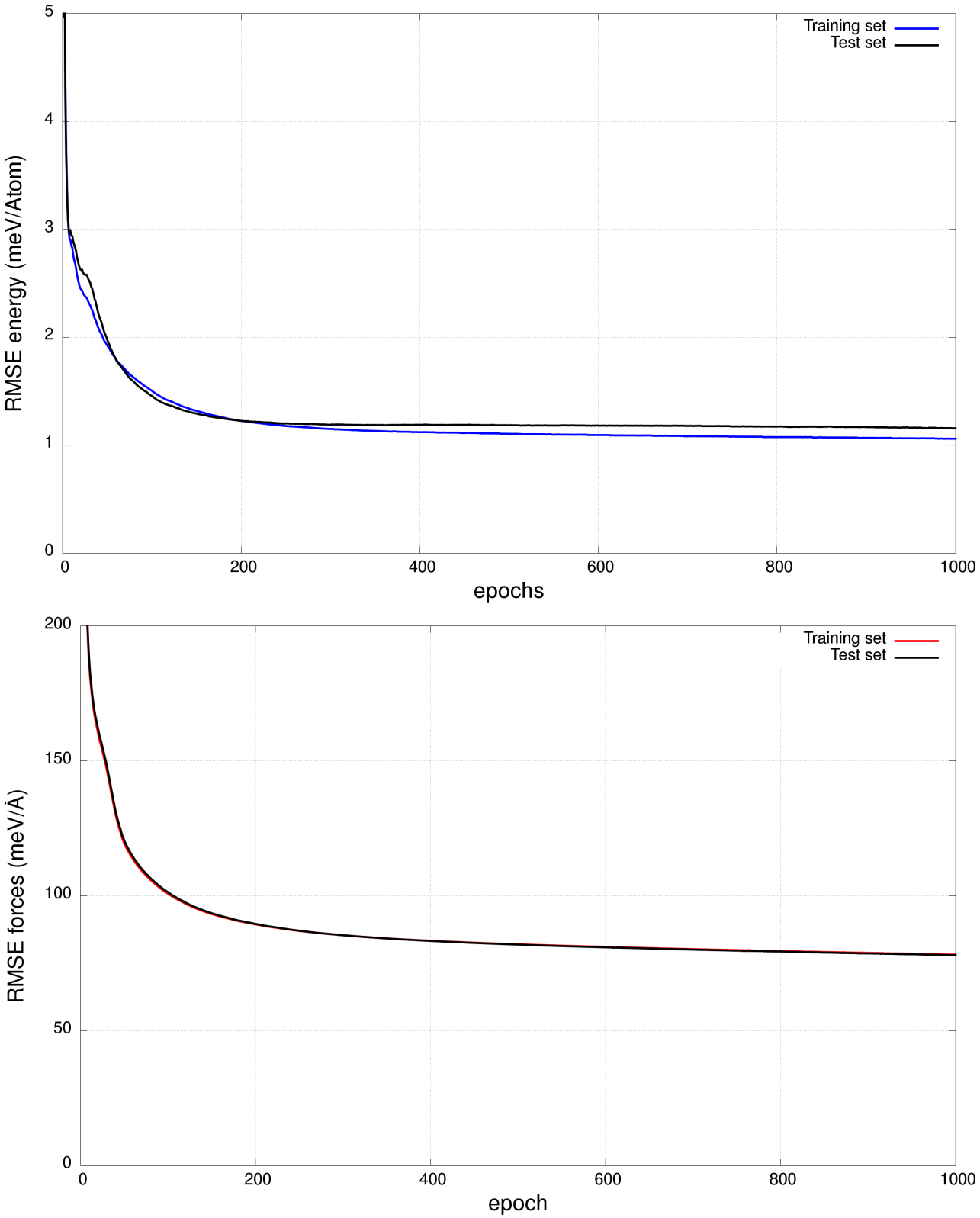


Figure 5: Energy and forces learning curves. The training has been performed on 800 structures and testing on 200 structures.

5 Part 2: Application of NNP in Molecular Dynamics

The stability of the NNP trained in the previous section should be verified in the molecular dynamics. To achieve this, we use a simulation workflow depicted in Fig. 6.

The workflow combines several existing approaches and codes representing a state-of-the-art method in molecular dynamics. The central part is the i-PI molecular dynamics driver, which propagates the nuclei using external potentials and guides the communication between all codes.

The evaluation of the energy and forces for a given geometry is performed by the n2p2 library implemented in LAMMPS using the NNP we trained in the previous sections. The workflow allows to run MD with two types of NNPs: (i) direct NNPs (DNNP) reproducing the DFT data, and (ii) baselined NNPs (BNNP) (*vide infra*) which serve as a correction applied on the top of a cheaper potential (*e.g.*, a semiempirical method).

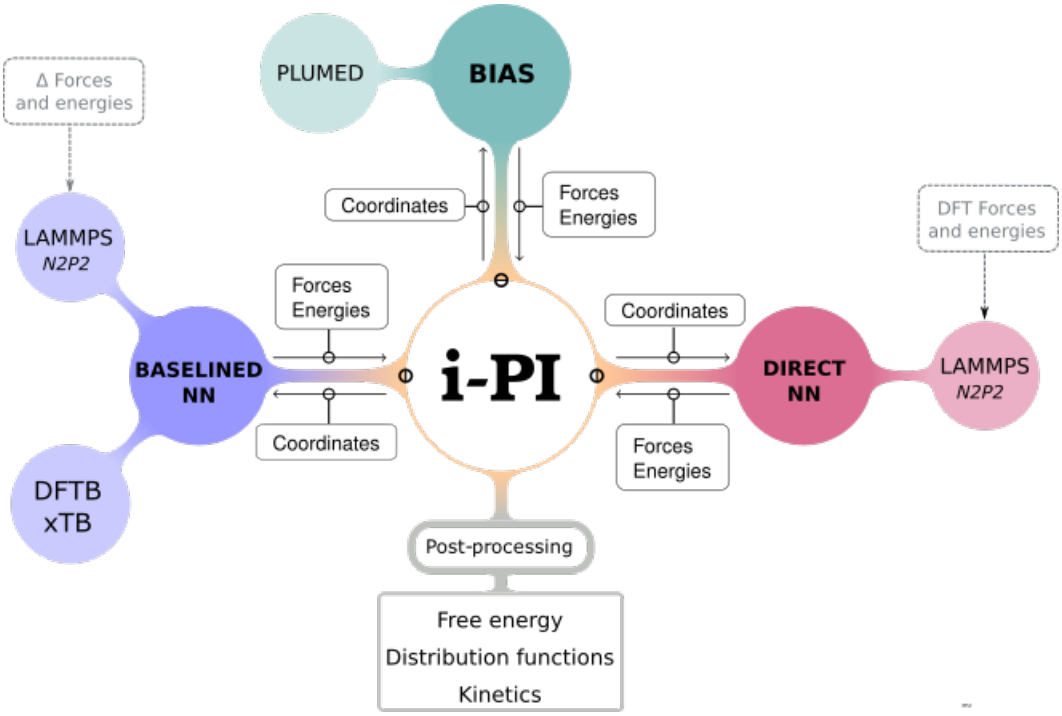
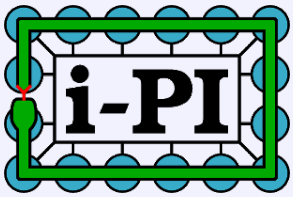


Figure 6: Scheme of the simulation workflow.

5.1 Launching LAMMPS with the i–PI interface

Input Files, Executables and Scripts



- system.xyz
- input.xml
- lmp1.in
- initial.data
- nnp-parameters directory
- lmp

The detailed documentation and tutorials for the codes used here are listed in section 7. We recommend users get familiar with the inputs and submission commands. Here, we guide to the setting relevant to the application of NNPs.

In this section, we use the i-PI driver in combination with the n2p2 library, which is part of the LAMMPS code.

To run MD with the NNP, create a new directory and copy them the following files:

- input.xml - the input file for i_PI
- system.xyz - the xyz coordinates of the initial structure for MD
- lmp1.in - the input file for LAMMPS
- initial.data - the same structure as in the system.xyz, but in the LAMMPS format
- NNP parameters

To extract the parameters of the NNP, create the nnp_parameters directory and place it in the files obtained during the training, namely: input.data, input.nn, scaling.data and weights.XXX.001000.out. The name of the weight files needs to be changed to format weights.XXX.data.

Now, let's have look at the content of the different input files. The system.xyz is the starting frame of the MD simulation, which is read by i-PI. The same structure needs to be provided also for the LAMMPS and can be found in the initial.data. The initial data file can be generated by the script get_lammps.sh.

The input.xml contains the parameters of the molecular dynamics, for example, simulation ensemble, thermostat, temperature, and time step. The energy and forces are computed by an external code, in our case LAMMPS. The connection between the i-PI and LAMMPS is provide via socket interface, specified in the part **ffsocket**:

```
<ffsocket mode="inet" name="driver-lammps1">
<address> 192.168.100.1</address>
<port>8762</port>
<slots>1</slots>
<timeout>600</timeout>
</ffsocket>
```

In this case, the i-PI and LAMMPs communicate via an inet interface, which is defined by a unique combination of IP address and port. Details of the setting can be found in the i-PI documentation. In general, the i-PI establishes the communication channel using the given address. The external code then connects to this channel to receive and send information. The socket interface prevets the initialization of the external code in every step, which significantly saves computational time.

The input parameters for the LAMMPS are placed in the lmp1.in file. The NNP specification is described in the pair_style:

```
pair_style nnp dir ${runnerDir} showew no showewsum 1 resetew yes maxew 200000
cflength 1.889726 cfenergy 0.036749

pair_coeff * * ${runnerCutoff}
```

The pair style informs LAMMPS to use the interaction computed by nnp potentials trained with n2p2. $runnerDir variable contains the path to the NNP parameters. The additional parameters control the printing of the extrapolation warning by n2p2. Detailed explanation of possible option is given in the n2p1 tutorial (see section 7).

The pair_coeff command contains information on the runner_cutoff, which is the larger cutoff of the symmetry functions used in the NNP training. It needs to be specified in the LAMMPS units, *i.e.* in Angstrom.

The last part of the input then specifies the connection to i-PI and the number of steps.

```
fix 1 all ipi 192.168.100.1 8762
```

To launch the molecular dynamics, initiat the the i-PI by:

python i-pi input.xml >> ipi.out &

and then launch LAMMPS:

1. export LD_LIBRARY_PATH=/path/to/n2p2/lib:${LD_LIBRARY_PATH}
2. lmp -i lmp1.in >> log1.lammps &

The simulation trajectory will be printed in the simulation.pos_0.xyz and can be visualized for example by vmd (https://www.ks.uiuc.edu/Research/vmd/).

While the NNPs have sufficiently small training errors, the MD will be stable for only a few steps. After that, it will start to produce unphysical geometries and eventually explode. **The monitoring RMSE itself is therefore not sufficient to estimate the quality of the resulting potentials!**

To obtain stable and reliable molecular dynamics, the NNP quality needs to be improved significantly. One of the possibilities is to identify the problematic geometries from the molecular dynamics, compute the reference energy and forces and retrain the NNP with the new set of structures. This needs to be often repeated several times until the NNP provides stable and reliable results. The number of needed structures may, however, often reach more than a dozen thousand. Another strategy is to start with the structures generated by a sufficiently good level of theory, for example, the semiempirical method or DFT (*e.g.*, PBE).

6 Part 3: From direct to baselined NNP

6.1 What is baselined NNP?

In the previous sections, we learned how to generate the NNPs and use them with the i-PI interface. While the NNP trained directly on DFT data provide predictions at impressive speed, they are often difficult to stabilize, especially for complex systems using more than three elements.

An alternative for large and complex systems is so-called baselined NNP.[10] Instead of reproducing directly the DFT data, the baselined NNPs are trained to correct the energy and forces computed at a lower level of theory. The baselined NNPs thus require a lower number of structures and descriptors and are more robust than the direct NNP. In the following parts, we demonstrate how to train the baselined NNPs and apply them in the MD.

6.2 Baselined energy and forces

Input Files, Executables and Scripts

- input.data
- xtb_ase.py xtb_io.py
- xtb

In the beginning, we need to choose a suitable baseline method for the NNP. The baseline can be any computational method that provides reasonable results at a computational cost significantly lower than the reference. This is important as the baseline is used also to propagate the dynamics of the system. Here, we use the GFN0-xTB Hamiltonian from the family of xTB methods.[11]

The construction of the baselined NNPs is practically identical to the direct NNP. However, the target properties are not the DFT data but the difference between the DFT and GFN0-xTB results. As a first step, we, therefore, need to compute the energy and forces for the existing set of structures. For this, we use the periodic implementation of GFN0-xTB in xtb v6.2. To automatize the preparation of the files and computations, we perform the computations by the xtb-ase calculator available at xtb_io.py.

In the beginning, copy the xyz structures used for PBE computations into a new folder. Then, use script prepare_xtb.sh to move every structure to a separated dictionary and submit the computations. The example of the submission script is at submit.sh. You will need to modify the path to the xtb and cluster-specific settings.

Once the computations are finished, use process_xtb_forces.sh to create the forces database and translate_xtb.f90 to generate new input.data file.

Output Files

- **input.data**

6.3 The training

Input Files, Executables and Scripts

- new–input.data
- input.nn
- nnp–scaling
- nnp–train

In the next step, create a new folder for the training of the NNP, move there the input.data and input.nn files and generate the scaling.data file for the new database. Here, we use the same input.nn file as for the direct NNP. However, the best parameters for the baselined NNP might be different. Feel free to experiment with the number of the epoch, size of the training set and test set, a ratio of the forces, etc.

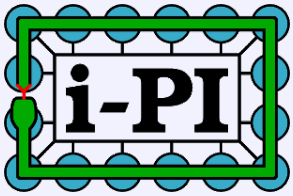
In general, the RMSE errors in the training of the baselined NNP will be lower than in the direct case. For the comparison of the baselined and direct NNP, see for example Ref. 12,13.

Output Files

- **learning-curve.out**
- **weights.001.001000.out**
- **weights.008.001000.out**

6.4 Baselined–NNP with i–PI

Input Files, Executables and Scripts



- input.data input.nn scaling.data
- weights.001.001000.out
  weights.008.001000.out
- input.xml
- initial.data system.xyz
- lmp1.in
- xtb_ase.py xtb_io.py

Once the training finishes, you can launch the baselined–NNP MD. The main difference concerning the direct DFT is that apart from the NNP itself, you also need to compute the baseline energy and forces at every step. The input.xml for the i-PI, therefore, includes computations of forces by additional external code, in our case xtb via xtb-ASE calculator. The new lines in the input.xml include two communication channels:

```
<ffsocket mode="inet" name="driver-lammps1">
   <address>192.168.100.117</address>
   <port>8762</port>
   <slots>1</slots>
   <timeout>600</timeout>
 </ffsocket>
 <ffsocket mode='inet' name='driver-xtb'>
  <address>192.168.100.117</address>
  <port>8763</port>
  <slots>1</slots>
  <timeout>600</timeout>
  </ffsocket>
```

and two force computations:

```
 <forces>
   <force forcefield ="driver-lammps1" weight='-1.000'></force>
   <force forcefield ="driver-xtb"></force>
 </forces>
```

The weight in the first line depends on the definition of your correction in the input.data. Here, the baselined NNP reproduces xTB - DFT energy and forces and has a negative sign.

While the MD with baselined NNP is more stable than the direct NNP one and does not explode, it still produces unphysical geometries and has large extrapolation warnings (see log1.lammps). The iterative improvement of the NNP is therefore needed, similarly as in the direct NNP case. However, a lower number of structures will be needed to improve the quality of the NNP.

6.5 Multiple-time-step approach

In the previous sections, we introduced direct and baselined NNP. While the baseline NNPs are more robust even for simple systems, their application in molecular dynamics is significantly more expensive due to the baseline computations. These limitations become more important for complex systems and systems with a large number of atoms. To decrease the computational cost of the baselined NNP while keeping its accuracy, we use the so-called multiple-time-step approach (MTS).[14–17]

In this approach, the molecular dynamics are propagated by the forces corresponding to the cheaper and less accurate potential computed with a shorter integration time step (*i.e.* inner time step) and are corrected by more accurate and expensive potential evaluated with a longer time step (*i.e.*, outer time step). To estimate the largest allowed difference between the outer and inner time step, we need to monitor the conservation of the energy during the simulation. An example of the MTS benchmark study is, for example, in the Supporting information of Ref. 12.

Here, we use an MTS with a 1-6 scheme, where the energy and forces are computed by a direct NNP using an inner time step of 0.5 fs and corrected by GFN0-xtb and baselined NNP every 3 fs. The input.xml file therefore needs to contain computation forces with direct NNP, baselined NNP and GFN0-xTB as:

```
<forces>
  <force forcefield ="driver-lammps1" weight='-1.000'>
  <mts_weights>[1.0,0]</mts_weights> </force>
  <force forcefield ="driver-xtb">
  <mts_weights>[1.0,0]</mts_weights> </force>
  <force forcefield ="driver-lammpsd1">
  <mts_weights>[-1.0,1.0]</mts_weights></force>
  </forces>
```

The mts_weights indicate the weights of the outer and inner time step, respectively. The baselined NNP and xtb potentials weigh 1.00 in the outer step and 0 in the inner step, while direct NNP has the weights 1.0 in the inner step and -1.0 in the outer step to avoid double counting of the NNP.

The MTS scheme is then defined as

```
<nmts> [1,6] </nmts>
<timestep units='femtosecond'> 3.0 </timestep>
```

where the time step corresponds to the outer step. The inner step is determined by the MTS scheme. You can change the MTS scheme and observe its impact on the simulation cost and stability of the dynamics.

7 Supplementary tutorials

7.1 Tinker(–HP)

- Frédéric CELERSE personnal tutorials (on request): cMD / preparing Tinker xyz files / Umbrella Sampling / Steered Molecular Dynamics / Gaussian accelerated Molecular Dynamics
- AMOEBA workshop: https://sites.google.com/site/amoebaworkshop/
- Tinker website: https://dasher.wustl.edu/tinker/
- GitHub (open source code): https://github.com/TinkerTools/tinker

7.2 n2p2

- GitHub (open source code): https://github.com/CompPhysVienna/n2p2
- Manual: https://compphysvienna.github.io/n2p2/index.html

7.3 cp2k

- GitHub (open source code): https://github.com/cp2k/cp2k
- cp2k website: https://www.cp2k.org/

7.4 i–PI

- GitHub (open source code): https://github.com/i-pi/i-pi
- i–PI website: http://ipi-code.org/
- i–PI manual: http://ipi-code.org/assets/pdf/manual.pdf

7.5 LAMMPS

- GitHub (open source code): https://github.com/lammps/lammps
- LAMMPS website: https://lammps.sandia.gov/
- LAMMPS tutorials: https://icme.hpc.msstate.edu/mediawiki/index.php/LAMMPS_tutorials.html

References

1. Behler, J.; Parrinello, M. Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces. *Phys. Rev. Lett.* **2007**, *98*, 146401.

2. Behler, J. Atom-Centered Symmetry Functions for Constructing High-Dimensional Neural Network Potentials. *J. Chem. Phys.* **2011**, *134*, 074106.

3. Musil, F.; Grisafi, A.; Bartók, A. P.; Ortner, C.; Csányi, G.; Ceriotti, M. Physics-Inspired Structural Representations for Molecules and Materials. *Chem. Rev.* **2021**, *121*, 9759–9815.

4. Behler, J. Constructing high-dimensional neural network potentials: A tutorial review. *Int. J. Quantum Chem.* **2015**, *115*, 1032–1050.

5. Gastegger, M.; Schwiedrzik, L.; Bittermann, M.; Berzsenyi, F.; Marquetand, P. WACSF - Weighted atom-centered symmetry functions as descriptors in machine learning potentials. *J. Chem. Phys.* **2018**, *148*, 241709.

6. Bircher, M. P.; Singraber, A.; Dellago, C. Improved description of atomic environments using low-cost polynomial functions with compact support. *Mach. Learn. Sci. Technol.* **2021**, *2*, 035026.

7. Imbalzano, G.; Anelli, A.; Giofré, D.; Klees, S.; Behler, J.; Ceriotti, M. Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials. *J. Chem. Phys.* **2018**, *148*, 241730.

8. Mahoney, M. W.; Drineas, P. CUR Matrix Decompositions for Improved Data Analysis. *Proc. Natl. Acad. Sci. USA* **2009**, *106*, 697–702.

9. Cersonsky, R. K.; Helfrecht, B. A.; Engel, E. A.; Kliavinek, S.; Ceriotti, M. Improving sample and feature selection with principal covariates regression. *Mach. Learn.: Sci. Technol.* **2021**, *2*, 035038.

10. Ramakrishnan, R.; Dral, P. O.; Rupp, M.; von Lilienfeld, O. A. Big data meets quantum chemistry approximations: the Δ-machine learning approach. *J. Chem. Theory Comput.* **2015**, *11*, 2087–2096.

11. Bannwarth, C.; Caldeweyher, E.; Ehlert, S.; Hansen, A.; Pracht, P.; Seibert, J.; Spicher, S.; Grimme, S. Extended tight-binding quantum chemistry methods. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2021**, *11*, e1493.

12. Rossi, K.; Jurásková, V.; Wischert, R.; Garel, L.; Corminbœuf, C.; Ceriotti, M. Simulating Solvation and Acidity in Complex Mixtures with First-Principles Accuracy: The Case of CH_3SO_3H and H_2O_2 in Phenol. *J. Chem. Theory Comput.* **2020**, *16*, 5139–5149.

13. Jurásková, V.; Célerse, F.; Laplaza, R.; Corminboeuf, C. Assessing the persistence of chalcogen bonds in solution with neural network potentials. *Submitted.*

14. Guidon, M.; Schiffmann, F.; Hutter, J.; Vandevondele, J. Ab Initio Molecular Dynamics Using Hybrid Density Functionals. *J. Chem. Phys.* **2008**, *128*, 214104.

15. Luehr, N.; Markland, T. E.; Martínez, T. J. Multiple time step integrators in ab initio molecular dynamics. *J. Chem. Phys.* **2014**, *140*, 084116.

16. Liberatore, E.; Meli, R.; Rothlisberger, U. A Versatile Multiple Time Step Scheme for Efficient Ab Initio Molecular Dynamics Simulations. *J. Chem. Theory Comput.* **2018**, *14*, 2834–2842.

17. Kapil, V.; VandeVondele, J.; Ceriotti, M. Accurate molecular dynamics and nuclear quantum effects at low cost by multiple steps in real and imaginary time: Using density functional theory to accelerate wavefunction methods. *J. Chem. Phys.* **2016**, *144*, 054111.