universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Machine learning potentials for Silicon using Bayesian regression“

verfasst von / submitted by

Bernhard Schmiedmayer, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 876
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Master Physics
Betreut von / Supervisor:	Univ.-Prof. Dipl.Ing. Dr. Georg Kresse
Mitbetreut von / Co-Supervisor:	

Acknowledgements

I would like to thank the members of the institute for computational material physics for supervising this master's thesis despite the difficulties caused by the ongoing Covid pandemic. Special thanks go to Georg Kresse, Carla Verdi, and to the employees of the VASP software GmbH Peitao Liu, Andreas Singraber, and Ferenc Karsai for giving extensive support. Additionally I would like to thank Christoph Dellago, Michael Pöltl, and Atsushi Togo.

Abstract

In this master's thesis a multi purpose machine learned force field for silicon is generated using an on-the-fly method. The data for training are selected during molecular dynamics simulations. This selection process uses Bayesian error estimation to decide when to perform first principles calculations and to update the existing machine learned force field. On top of the on-the-fly generated data we update the machine learned force field with some additional data from high-pressure phases. For training, various silicon bulk phases, defects and nano crystals are used. A number of tests are performed to validate the generated machine learned force field. Finally, a machine learned force field alongside a neural network potential is used to calculate the thermal conductivity of silicon via non equilibrium molecular dynamics simulations and the two different machine learning methods are compared.

Kurzfassung

In dieser Masterarbeit wird ein maschinell erlerntes Mehrzweck-Kraftfeld für Silizium erzeugt. Dafür wird eine „on-the-fly“ Methode verwendet. Die Daten für das Training werden während molekulardynamischer Simulationen ausgewählt. Dieser Auswahlprozess verwendet die Bayes'sche Fehlerschätzung, um zu entscheiden, wann direkte Methoden durchgeführt werden sollen, um das vorhandene maschinell erlernte Kraftfeld zu aktualisieren. Zusätzlich zu den on-the-fly-generierten Daten wird das maschinell erlernte Kraftfeld mit einigen zusätzlichen Daten von Hochdruckphasen ergänzt. Für das Training werden verschiedene Silizium-Bulk-Phasen, Defekte und Nanokristalle verwendet. Um die Genauigkeit des maschinell erlernten Kraftfeldes zu validieren, werden einige Tests durchgeführt. Schließlich wird ein maschinell erlerntes Kraftfeld zusammen mit einem Neuronalen Netzwerk Potenzial verwendet, um die Wärmeleitfähigkeit für Silizium unter Verwendung von molekulardynamischen Nichtgleichgewichts-Simulationen zu berechnen und die beiden Ergebnisse werden miteinander verglichen.

Contents

1. Introduction

The main goal of this master thesis is to generate a multi-purpose potential for silicon using an on-the-fly machine learning strategy supported by the Vienna Ab *initio* Simulation Package (VASP).

Machine Learned Force Fields (MLFF) are a promising tool to reduce the computational cost of calculations based on density functional theory or other first First Principles (FP) methods. Since with MLFF the computing time increases only linearly with the number of atoms in the simulation cell, they are particularly useful for Molecular Dynamics (MD) simulations where large system sizes and long time scales are needed to accurately determine many materials properties.

Silicon is of great importance in materials science because of its multifaceted areas of application. For example this semiconductor is used in photovoltaics, thin-film transistors, microprocessors and batteries[1]. For this study we focus on silicon for a number of different reasons. One aspect is the far-reaching experience and the understanding in the simulation community about this material and its potential. In particular, we are mainly interested in the fact that silicon can present many stable and metastable phases which result in a rich phase diagram, as well as point and line defects[2].

There are two reasons behind the choice to generate a general purpose potential for Silicon using Machine Learning (ML) techniques. First computing complex properties of materials such as phase transitions or in general temperature dependent properties is extremely hard using density functional theory or other FP methods. The main reason behind this is the high computational cost and the poor scaling properties of FP methods. This means that only small systems with a maximum of a few hundred atoms can be simulated over small timescales[1, 3, 4]. Empirical Force Fields (EFF) are a common way to keep computing times low, but they have a disadvantage which leads to our second motivation. EFF can well reproduce individual properties of a material, such as melting points, if using the right parameters, however the simple functional forms derived by EFF can not reproduce multiple observables from experiments simultaneously[2]. ML methods are a promising technique to overcome these hurdles.

A MLFF model will be used to train a potential for Silicon. The data needed to fit the MLFF consist of the Bravais lattice, the atomic positions, the total energy, the forces, the stress tensor[3]. The Gaussian Approximation Potential (GAP) and Smooth Overlap of Atomic Positions (SOAP) are used for the potential energy fitting to train the MLFF. The advantage of such potentials is that they have no fixed functional form, and can therefore model complex potential energy surfaces. Furthermore, GAP can be systematically improved with more data[5].

In general, ML potentials are only accurate for structures close to the training set. An convenient on-the-fly strategy is implemented by using Bayesian regression to train the MLFF. With Bayesian

regression it is possible to estimate the error of thr MLFF predictions[6]. This error is used to judge whether the MLFF is accurate enough or FP calculations are needed. In the latter case new FP calculations are preformed and used to refine the MLFF.

2. Machine Learned Force Fields

This chapter deals with the basic method and theory for creating a MLFF. The underlying idea is to use ML methods to predict the most vital observables from structural information. These observables are the total potential energy, the forces and the stress tensor. The goal is to minimize human intervention during training while attaining near FP accuracy. For each atom in a structure, a local configuration around the atom can be determined, which is then mapped onto a set of descriptors by using the radial and angular distribution functions. In the following we will call Local Reference Configurations (LRC) a set of chosen local reference structures. Each structure dataset, contains the Bravais lattice, the atomic positions, the resulting total energy, the forces acting on each atom, and the stress tensor. We call 'Reference Structure DataSet (RSDS) ' a chosen structure dataset. Both LRC and RSDS are used to generate the MLFF. The observables are obtained by FP calculations. As ML model a kernel based method is used. The detailed methodology is originally described in [3] and in the VASP Manual [7].

2.1. On-the-fly learning algorithm: an overview

In order to generate a FF using ML methods the amount of training data has to be chosen carefully. Too few will result in bad accuracy. Too many can lead to over fitting of the MLFF and would be a waste of compute time, since data are generated by expensive FP calculations. One can get around this problem by adapting an on-the-fly learning strategy. During each step of an MD calculation the algorithm judges whether to make a FP calculation or use the MLFF to compute the next time step. This decision is made on the basis of an error estimation of the MLFF prediction. If the error is too large a FP calculation is performed and the resulting structure dataset is added to the training set to update the MLFF. With this method the MLFF is built up during an MD run and refines itself. The more precise the MLFF, the fewer expensive FP calculations are needed and eventually the training can be stopped.

To generate or update a MLFF using our on-the-fly learning method the following steps are used:

1. Read an existing MLFF if available and start an MD simulation.

2. The existing MLFF is used to predict the energy, the forces and the stress tensor and their uncertainties for a given structure $\mathbf{R}_I$.

3. If the maximal uncertainty of any force is lager than a threshold a FP calculation is performed (continue with step 4) else the MLFF is used to compute the next MD step (skip to step 6).

4. A FP calculation is performed and a new structure dataset is obtained which could be used to update the MLFF.

5. If a certain number of new structure datasets is acquired or the predicted error is too large, the MLFF is retrained after adding new LRC and RSDS computed by previous FP calculations.

6. The equations of motion are solved by using the MLFF, if accurate enough, or by a FP calculation to obtain the new atomic positions and velocities of structure $\mathbf{R}_{I+1}$.

7. If the desired number of MD steps is reached, the final MLFF parameters are written to output files together with the LRC and RSDS database. Otherwise we loop back to step 2.

This method relies heavily on an accurate description of the potential energy surface and on the evaluation of the uncertainty. It also requires careful optimization of the threshold parameters for the uncertainty. Additional crucial steps are the sparsification and data selection, which will be explained in the following sections. Algorithm 1 shows an outline of the on-the-fly learning method for generating a MLFF.

Algorithm 1: The on-the-fly MLFF generation scheme is outlined here. $\mathbf{R}$ contains all atomic positions and N_{MD} denotes the total number of desired MD steps

```
Input: Atomic positions at start of MD simulation R_0 (Optional: an existing MLFF for
       further training)
for I = 0 until I = N_MD do
    Use MLFF to get energy, forces and stress tensor for R_I;
    Update threshold parameter on estimated errors;
    if MLFF accurate enaugh then
        Use MLFF to solve equation of motions to get R_{I+1};
    else
        Perform FP calculation;
        Select Data (adding new LRC and RSDS);
        Update MLFF;
        Use FP calculation to solve equations of motion to get R_{I+1};
    end
end
Output: LRC and RSDS database and new/updated MLFF
```

2.2. Descriptors and potential energy surface

A similar method to GAP [5] with SOAP [8] with some modifications is used to describe the potential energy surface. The modifications are added to explore a more efficient and stable on-the-fly MLFF generation. In order to describe the methodology qualitatively, simplified equations for single element systems are used. These could be extended to multi-element systems easily.

2.2.1. Potential Energy

Suppose we have a system with N_a atoms. Then the potential energy U can be written as summation of local energies U_i:

$$U = \sum_{i=1}^{N_a} U_i, \tag{2.1}$$

where U_i is determined by the local environment around the atom i. This is achieved by writing the local energies as functions of a probability density

$$U_i = F[\rho_i(\mathbf{r})]. \tag{2.2}$$

The density ρ_i describes how likely another atom j is to be found at the position $\mathbf{r}$. It is defined as

$$\begin{aligned} \rho_i(\mathbf{r}) &= \sum_{j=1}^{N_a} \tilde{\rho}_{ij}(\mathbf{r}), \\ \tilde{\rho}_{ij}(\mathbf{r}) &= f_{\text{cut}}(r_{ij}) g(\mathbf{r} - \mathbf{r}_{ij}). \end{aligned} \tag{2.3}$$

Where $\tilde{\rho}_{ij}$ describes the likelihood to find an atom j at position $\mathbf{r}$ relative to an atom i, and $\mathbf{r}_i$ denotes the position of the atom i [9]. The relative distance between two atoms i and j is defined by $r_{ij} = |\mathbf{r}_{ij}| = |\mathbf{r}_j - \mathbf{r}_i|$. This distribution function is truncated by a certain cut off radius R_{cut}. The function f_{cut} is used to smoothly remove information beyond R_{cut}. The following cutoff function described by Behler and Parrinello [10] is used

$$f_{\text{cut}}(r_{ij}) = \begin{cases} \frac{1}{2}\left[\cos\left(\pi \frac{r_{ij}}{R_{\text{cut}}}\right) + 1\right] & \text{if } r_{ij} \leq R_{\text{cut}} \\ 0 & \text{else} \end{cases}. \tag{2.4}$$

When paired with descriptors, this function efficiently describes the structural differences that strongly influence the potential energy of the central atom i. This is done by weighing atoms close to the center more strongly. Normally $g(\mathbf{r})$ is the Dirac delta function $\delta(\mathbf{r})$ but as in SOAP it is replaced by a normalized Gauss distribution written as

$$g(\mathbf{r}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{atom}}} \exp\left(-\frac{|\mathbf{r}|^2}{2\sigma_{\text{atom}}^2}\right), \tag{2.5}$$

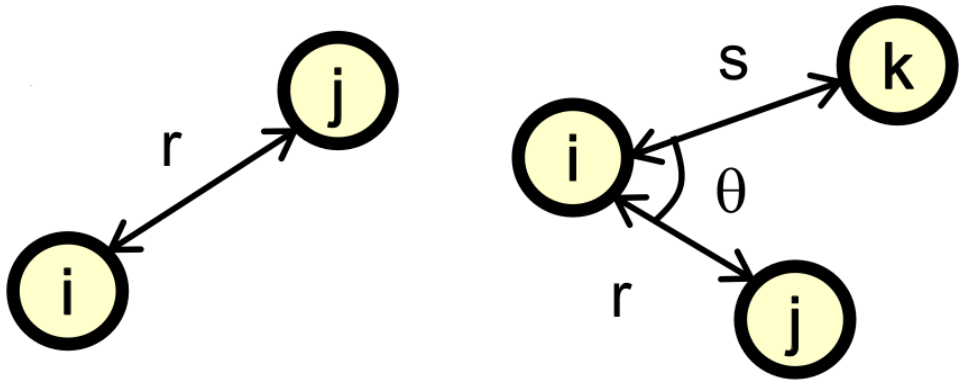
where σ_{atom} is the width of the Gaussian function. At this point one could simply express F through $\rho_i(\mathbf{r})$ via a numerical approach. Unfortunately, $\rho_i(\mathbf{r})$ is not rotationally invariant. Consequently F is neither rotationally nor translationally invariant. This problem is solved by using so-called descriptors as intermediate functions. Although they depend on $\rho_i(\mathbf{r})$, they possess rotational as well translational invariance.

2.2.2. Radial descriptors

The radial distribution function is used as the simplest descriptor. It is both rotationally and translationally invariant. The radial/two-body descriptor is defined as

$$\rho_i^{(2)}(r) = \frac{1}{4\pi} \int \rho_i(r\hat{\mathbf{r}}) \mathrm{d}\hat{\mathbf{r}}. \tag{2.6}$$

It contains information on pairwise distances around atom i within R_{cut}. The unit vector of $\mathbf{r}$ is denoted as $\hat{\mathbf{r}}$. Figure 2.1a shows the vector $\mathbf{r}$ between two atoms i and j schematically.



(a) Radial descriptors (b) Angular descriptors

Figure 2.1.: The descriptors are visualized [11]

2.2.3. Angular descriptors

The radial descriptor lacks sufficient information about the system to predict the potential energy surface accurately. It is possible to obtain the same $\rho_i^{(2)}$ from different probability densities ρ_i. Therefore an additional descriptor containing angular information is introduced. It describes the probability to find an atom j at a distance r and another atom k at a distance s around the atom i along the angle $\theta = \angle kij$ between the two vectors $\mathbf{r}$ and $\mathbf{s}$. A schematic is shown in Figure 2.1b. The so called angular descriptor, defined as:

$$\rho_i^{(3)}(r,s,\theta) = \iint \delta(\hat{\mathbf{r}}\cdot\hat{\mathbf{s}} - \cos\theta)\left[\rho_i(r\hat{\mathbf{r}})\rho_i^*(s\hat{\mathbf{s}}) - \sum_{j\neq i}^{N_a}\tilde{\rho}_{ij}(r\hat{\mathbf{r}})\tilde{\rho}_{ij}(s\hat{\mathbf{s}})\right]\mathrm{d}\hat{\mathbf{r}}\mathrm{d}\hat{\mathbf{s}} \tag{2.7}$$

is crucial for computing the potential energy surface. The descriptor $\rho_i^{(3)}$ is similar to the power spectrum used in other applications of the GAP.

2.2.4. LRC expansion

One can expand the atomic probability density in terms of radial basis functions χ_{nl} and spherical harmonics Y_{lm}

$$\rho_i(\mathbf{r}) = \sum_{l=1}^{L_{\max}}\sum_{m=-l}^{l}\sum_{n=l}^{N_R^l} c_{nlm}^i \chi_{nl}(r)Y_{lm}(\hat{\mathbf{r}}). \tag{2.8}$$

Here c_{nlm}^i denote the expansion coefficients which are defined in Eq. (2.15). The indices n, l and m are the main, angular and magnetic quantum numbers. As radial basis functions, normalized

spherical Bessel functions are used $\chi_{nl}(r) = j_l(q_{nl}r)$. The parameters q_{nl} are chosen such that

$$\chi_{nl}(r) = \begin{cases} j_l(q_{nl}r) & \text{if } r \leq R_{\text{cut}} \\ 0 & \text{else} \end{cases}. \tag{2.9}$$

Since j_l is orthogonal, χ_{nl} satisfies the relation

$$q_{nl}q_{n'l}\frac{2}{\pi}\int_0^\infty \chi_{nl}(r)\chi_{n'l}(r)r^2\mathrm{d}r = \delta(n-n'). \tag{2.10}$$

Due to the orthogonal behavior the accuracy can be systematically improved by increasing the number of radial basis functions. By using the above Eq. (2.8) for the probability density the radial (Eq. (2.6)) and angular (Eq. (2.7)) descriptors can be rewritten as

$$\rho_i^{(2)}(r) = \frac{1}{\sqrt{4\pi}}\sum_{n=1}^{N_R^0} c_n^i \chi_{nl}(r), \tag{2.11}$$

$$\rho_i^{(3)}(r,s,\theta) = \sum_{l=1}^{L_{\max}}\sum_{n=1}^{N_R^l}\sum_{v=1}^{N_R^l}\sqrt{\frac{2l+1}{2}}p_{nvl}^i\chi_{nl}(r)\chi_{vl}(s)P_l(\cos\theta), \tag{2.12}$$

with P_l defined as a Legendre polynomial of l-th order, N_R^l the number of radial basis functions,

$$c_n^i = c_{n00}^i, \tag{2.13}$$

and

$$p_{nvl}^i = \sqrt{\frac{8\pi^2}{2l+1}}\sum_{m=-l}^{l}\left[c_{nlm}^i c_{vlm}^{i*} - \sum_{j\neq i}^{N_a}\tilde{c}_{nlm}^{ij}\tilde{c}_{vlm}^{ij*}\right]. \tag{2.14}$$

The expansion coefficients c_{nlm}^i and $\tilde{c}_{nlm}^{ij}$ are defined as

$$\begin{aligned} c_{nlm}^i &= \sum_{j=1}^{N_a}\tilde{c}_{nlm}^{ij}, \\ \tilde{c}_{nlm}^{ij} &= h_{nl}(r_{ij})Y_{lm}^*(\hat{\mathbf{r}}_{ij}), \end{aligned} \tag{2.15}$$

where $\tilde{c}_{nlm}^{ij}$ denotes the expansion coefficient of the distribution

$$\tilde{\rho}_{ij}(\mathbf{r}) = \sum_{l=0}^{L_{\max}}\sum_{m=-l}^{l}\sum_{n=1}^{N_R^l}\tilde{c}_{nlm}^{ij}\chi_{nl}(r)Y_{lm}(\hat{\mathbf{r}}). \tag{2.16}$$

Finally, we define $h_{nl}(r)$ used for $\tilde{c}_{nlm}^{ij}$ as

$$h_{nl}(r) = \frac{4\pi}{(2\pi\sigma_{\text{atom}}^2)^{\frac{3}{2}}}f_{\text{cut}}(r)\int_0^\infty \chi_{nl}(r')\exp\left(-\frac{r'^2+r^2}{2\sigma_{\text{atom}}^2}\right)\imath_l\left(\frac{rr'}{\sigma_{\text{atom}}^2}\right)r'^2\mathrm{d}r', \tag{2.17}$$

where $\imath_l$ is the modified spherical Bessel function of the first kind.

During the on-the-fly learning all expansion coefficients c_{nlm}^i and their derivatives $\mathrm{d}c_{nlm}^i/\mathrm{d}r$ need to be calculated at every MD step, however h_{nl} and $\mathrm{d}h_{nl}/\mathrm{d}h_{nl}$ are rather expensive to compute. To speed things up, h_{nl} is computed on a radial mesh over r at the beginning of the training. During training a spline interpolation is adopted to calculate the coefficients c_{nlm}^i and derivatives from the mesh. This allows to compute the coefficients faster by one order of magnitude. Moreover, one can noticeably reduce the necessary number of LRC by multiplying χ_{nl} with an angular filtering function η (see Eq. (3.1)) without loss of accuracy during calculations [12].

2.2.5. Kernel

We can now approximate the local energies U_i as functionals of the radial $\left(\rho_i^{(2)}\right)$ and angular $\left(\rho_i^{(3)}\right)$ descriptors instead of the density distribution ρ_i, since this has the benefit of being rotationally and translationally invariant:

$$U_i = F\left[\rho_i^{(2)}, \rho_i^{(3)}\right] = \sum_{i_B=1}^{N_B} \omega_{i_B} K(\mathbf{X}_i, \mathbf{X}_{i_B}). \tag{2.18}$$

Here the functional is written as summation over the number of LRC N_B where $K(\mathbf{X}_i, \mathbf{X}_{i_B})$ is the kernel which measures the similarity between a local configuration of interest ρ_i and a reference LRC ρ_{i_B}. The linear coefficients ω_{i_B} are fitted as explained in section 2.3. All coefficients c_n^i and p_{nvl}^i used in $\rho_i^{(2)}$ and $\rho_i^{(3)}$ are collected in the vectors $\mathbf{X}_i$, $\mathbf{X}_{i_B}$ like:

$$\mathbf{X}_i^{(2)} = \begin{pmatrix} c_1^i \\ c_2^i \\ \vdots \end{pmatrix}, \tag{2.19}$$

$$\mathbf{X}_i^{(3)} = \begin{pmatrix} p_{110}^i \\ p_{111}^i \\ \vdots \\ p_{120}^i \\ p_{121}^i \\ \vdots \end{pmatrix}, \tag{2.20}$$

where the superscript T indicates the transpose of the vector. The two-body coefficients in $\mathbf{X}_i^{(2)}$ can describe long-range radial interactions like Coulomb and Lennard-Jones and are referred to as radial coefficients. The three-body coefficients in $\mathbf{X}_i^{(3)}$ describe nonlinear many-body interactions and are referred to as radial coefficients.

In order to compute the kernel, first the two vectors $\mathbf{X}_i^{(2)}$ and $\mathbf{X}_i^{(2)}$ are written in a super-vector

$$\mathbf{X}_i = \begin{pmatrix} \sqrt{\beta^{(2)}}\mathbf{X}_i^{(2)} \\ \sqrt{\beta^{(3)}}\mathbf{X}_i^{(3)} \end{pmatrix}. \tag{2.21}$$

The parameters $\beta^{(2)}$ and $\beta^{(3)}$ control the weighting of the radial coefficients respectively. Finally the kernel is computed by [9]

$$K(\mathbf{X}_i, \mathbf{X}_{i_B}) = \left(\hat{\mathbf{X}}_i \cdot \hat{\mathbf{X}}_{i_B}\right)^{\zeta}. \tag{2.22}$$

The hat indicates that $\hat{\mathbf{X}}_i$ is the normalized vector of $\mathbf{X}_i$, while ζ define the polynomial power of the kernel.

2.3. Fitting the potential energy, forces and stress tensor

Each RSDS contains the FP data (energies, forces and stress tensors) to which the parameters ω_{i_B} are fitted. In the following the superscript $\alpha = 1, \ldots, N_{st}$ is introduced, where N_{st} denotes the total number of RSDS in the database. Using Eq. (2.18) the following equation must be fulfilled for ω_{i_B} in a least square sense for all RSDS

$$\frac{U^\alpha}{N_a^\alpha} = \sum_{i_B=1}^{N_B} \omega_{i_B} \sum_{i=1}^{N_a^\alpha} \frac{K(\mathbf{X}_i^\alpha, \mathbf{X}_{i_B})}{N_a^\alpha}, \tag{2.23}$$

with N_a^α the number of atoms and U^α being the FP potential energy of RSDS α. The forces and the stress tensor are also described as linear functions of ω_{i_B}. To calculate the forces, derivatives of the kernel with respect to the atomic coordinates are used. Derivatives of the kernel with respect to the unit cell coordinates are used to calculate the stress tensor.

2.3.1. Matrix-vector form

The energies per atom, forces and the stress tensors are fitted simultaneously. A matrix-vector form to collect the linear equations can be written as

$$\mathbf{Y} = \Phi \mathbf{w}, \tag{2.24}$$

where $\mathbf{w}$ is a vector of size N_B containing the coefficients ω_{i_B}. The super vector $\mathbf{Y}$ consists of sub vectors y^α which store the FP energy per atom, forces and stress tensor of RSDS α. The super vector $\mathbf{Y}$ has a total of

$$M = \sum_{\alpha=1}^{N_{st}} (1 + 3N_a^\alpha + 6) \tag{2.25}$$

entries. For each RSDS α there is one for the energy, three (one in every spatial direction) times the number of atoms N_a^α for the forces, and six for the components of the stress tensor. The entries of $\mathbf{Y}$ are divided by the respective standard deviations and are thus dimensionless (see section 3.2). The the so-called design matrix Φ [6] consists of N_{st} matrices ϕ^α stacked on top of each other. The first row of ϕ^α is made up of the kernel used to calculate the energy. The subsequent $3N_a^\alpha$ rows are the derivative of the kernel with respect to the various atomic coordinates to calculate the forces, and the final six rows are used to compute the stress tensor and thus consists of the derivatives of the kernel with respect to the unit cell coordinates. The design matrix thus has the

form

$$\Phi = \begin{bmatrix}
\sum_i \frac{1}{N_i^{\alpha=1}} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \frac{1}{N_i^{\alpha=1}} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{x_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{x_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{y_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{y_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{z_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{z_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{x_2} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{x_2} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{y_2} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{y_2} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{z_2} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{z_2} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\vdots & \vdots & \ddots \\
\sum_i \nabla_{a_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{a_1} K(\mathbf{X}_i^{\alpha=1}, \mathbf{X}_{i_B=2}) & \cdots \\
\vdots & \vdots & \ddots \\
\sum_i \frac{1}{N_i^{\alpha=2}} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \frac{1}{N_i^{\alpha=2}} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{x_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{x_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{y_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{y_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{z_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{z_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{x_2} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{x_2} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{y_2} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{y_2} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\sum_i \nabla_{z_2} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{z_2} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\vdots & \vdots & \ddots \\
\sum_i \nabla_{a_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=1}) & \sum_i \nabla_{a_1} K(\mathbf{X}_i^{\alpha=2}, \mathbf{X}_{i_B=2}) & \cdots \\
\vdots & \vdots & \ddots
\end{bmatrix}. \tag{2.26}$$

2.3.2. Fitting and error estimation

A key component of this on-the-fly learning scheme is the ability to evaluate uncertainties of predictions. This is used to make decisions whether FP calculations are necessary or not. Bayesian linear regression is the method of choice to both optimize $\mathbf{w}$ and estimate the uncertainties. Bayesian linear regression works by determining the probability $p(\mathbf{w}|\mathbf{Y})$ to find coefficients $\mathbf{w}$ that reproduce the energy, forces and stress tensor exactly for a finite number of FP data $\mathbf{Y}$. Besides the numerical noise in $\mathbf{Y}$ also the cutoff radius R_{cut} as well as the restriction to radial and angular descriptors makes an exact reproduction of the FP data impossible [13]. It is assumed that all the errors can be modeled using Gaussian distributed noise in the FP data thus the probability $p(\mathbf{w}|\mathbf{Y})$ is determined as a multivariate Gaussian distributions $\mathcal{N}$ centered at $\bar{\mathbf{w}}$:

$$p(\mathbf{w}|\mathbf{Y}) = \mathcal{N}(\bar{\mathbf{w}}, \Sigma). \tag{2.27}$$

This is called the posterior distribution, where the center is determined to be

$$\bar{\mathbf{w}} = \frac{1}{\sigma_v^2} \Sigma \Phi^T \mathbf{Y} \tag{2.28}$$

with

$$\Sigma^{-1} = \frac{1}{\sigma_w^2} \mathbf{I} + \frac{1}{\sigma_v^2} \Phi^T \Phi. \tag{2.29}$$

Here $\mathbf{I}$ denotes the unit matrix. The parameters σ_v^2 and σ_w^2 are used to balance the accuracy and robustness of the MLFF during training. $\mathscr{N}$ is defined as

$$\mathscr{N}(\bar{\mathbf{w}},\Sigma) = \frac{1}{\sqrt{(2\pi)^{N_B}||\Sigma||}} \exp\left[-\frac{(\mathbf{w}-\bar{\mathbf{w}})^T\Sigma^{-1}(\mathbf{w}-\bar{\mathbf{w}})}{2}\right], \tag{2.30}$$

with $||\Sigma||$ being the determinant of Σ. The posterior probability $p(\mathbf{w}|\mathbf{Y})$ is maximized with the coefficients found at the center of the Gaussian distribution $\bar{\mathbf{w}}$, thus the optimal coefficients to solve Eq. (2.24) are $\mathbf{w} = \bar{\mathbf{w}}$ from Eq. (2.28).

2.3.3. Bayesian error estimation

Assume a structure with the vector $\mathbf{y}$ containing its exact FP energy per atom, forces and stress tensor, and ϕ its associated kernel and kernel derivatives. Then the corresponding predicted quantities are computed by the vector $\phi\bar{\mathbf{w}}$. The uncertainty of these predictions is determinated by the probability to observe $\mathbf{y}$ on the basis of the training set $\mathbf{Y}$. This posterior distribution, $p(\mathbf{y}|\mathbf{Y})$, is also a Gaussian distribution

$$p(\mathbf{y}|\mathbf{Y}) = \mathscr{N}(\phi\bar{\mathbf{w}},\sigma) \tag{2.31}$$

and is derived from p($\mathbf{w}$|$\mathbf{Y}$). Here the diagonal elements of σ contain the dimensionless variances of the predicted values $\phi\bar{\mathbf{w}}$ and have the form [3]:

$$\sigma = \sigma_v^2\mathbf{I} + \phi^T\Sigma\phi. \tag{2.32}$$

The variances are also multiplied by the respective weights and used as Bayesian errors in order to determine the accuracy of the MLFF predictions, and thus whether to perform FP calculations or not.

In order to define the parameters σ_v^2 and σ_w^2 we use the evidence approximation described in Refs [14, 15]. This works by maximizing the evidence function

$$p(\mathbf{Y}|\sigma_v^2,\sigma_w^2) = \left(\frac{1}{\sqrt{2\pi\sigma_v^2}}\right)^M \left(\frac{1}{\sqrt{2\pi\sigma_w^2}}\right)^{N_B} \int \exp[-E(\mathbf{w})]\mathrm{d}\mathbf{w}, \tag{2.33}$$

with

$$E(\mathbf{w}) = \frac{1}{2\sigma_v^2}||\Phi\mathbf{w}-\mathbf{W}||^2 + \frac{1}{2\sigma_w^2}||w||^2. \tag{2.34}$$

The evidence function describes the probability to obtain the training data $\mathbf{Y}$ from the regression model. The best fit is obtained when the probability $p(\mathbf{Y}|\sigma_v^2,\sigma_w^2)$ is maximized, therefore the optimal parameters σ_v^2 and σ_w^2 are derived by solving the two equations $\partial p/\partial\sigma_v^2 = 0$ and $\partial p/\partial\sigma_v^2 = 0$. The solution is given by

$$\sigma_w^2 = \frac{|\bar{\mathbf{w}}|^2}{\gamma}, \tag{2.35}$$

$$\sigma_v^2 = \frac{|\mathbf{Y}-\phi\bar{\mathbf{w}}|^2}{M-\gamma}, \tag{2.36}$$

where

$$\gamma = \sum_{k=1}^{N_B} \frac{\lambda_k}{\lambda_k + 1/\sigma_w^2}. \tag{2.37}$$

with λ_k denoting the eigenvalues of the matrix $\Phi^T\Phi/\sigma_v^2$. Not all eigenvalues are used to compute σ_v^2 and σ_w^2. The matrix $\Phi^T\Phi/\sigma_v^2$ can become nonpositive definite due to its nonregularized nature. This can lead to numerical instabilities [14]. To overcome this potential problem, eigenvalues smaller than a certain threshold are excluded from the calculations. Choosing the appropriate parameters σ_v^2 and σ_w^2 can also prevent over fitting.

2.4. Decision to perform FP calculation and data selection

2.4.1. Decision to perform FP calculations

In order to decide whether to perform FP calculations or not, the steps described in algorithm 2 are followed. This decision is the backbone of the on-the-fly learning technique and is based on the estimated errors of the forces for each atom $\vec{\sigma} = \mathrm{Diag}(\sigma)$, the number of steps passed since previous sampling, and certain criteria/thresholds. In particular, the maximum estimated errors in the forces need to be all smaller than its criteria in order to skip FP calculations. The criteria for the Bayesian error threshold $\varepsilon_{\mathrm{BE}}$ is updated automatically throughout the on-the-fly training run. Finally there is a parameter for the variance of the previous estimated errors $\varepsilon_{\mathrm{Var}}$, which is used to decide whether to update $\varepsilon_{\mathrm{BE}}$ or not. This is also described in algorithm 2.

The Bayesian error criterion update works as follows. First the uncertainties using the MLFF and the spilling factors are calculated on a structure R_I. The criterion update routine is only entered when the MLFF was renewed at the previous MD step. If this is the case, the supremum norm of the Bayesian errors $||\vec{\sigma}||_\infty$ of the current MD step is stored in a vector σ_{max}. This vector is of size M_{HIS}, only holding the most recent M_{HIS} Bayesian errors (older ones are discarded). If the standard deviation divided by the average of σ_{max} is smaller than $\varepsilon_{\mathrm{Var}}$, then the average of σ_{max} is set to be the new criterion for the Bayesian error, $\varepsilon_{\mathrm{BE}}$.

The decision whether to perform FP calculations or not is based on the following steps:

- If $||\vec{\sigma}||_\infty$ is larger than two times $\varepsilon_{\mathrm{BE}}$ a FP calculation is performed.
- Otherwise the machine checks how many MD steps have passed since the last sampling step.
- If this number is smaller than the minimum interval to obtain training samples N_{INT}, the FP calculation is skipped. This is done in order to avoid sampling too similar structures.
- If the number is equal to or greater than N_{INT}, the threshold for the Bayesian errors, $\varepsilon_{\mathrm{BE}}$, is used for the final decision.

- If $||\vec{\sigma}||_\infty$ is larger than $\varepsilon_{\mathrm{BE}}$, a FP calculation is performed.

Algorithm 2: The scheme whether to perform a FP calculation or not is outlined here. $\mathrm{Var}(x)$ and $\mathrm{E}(x)$ refer to the variance and the average respectively. $||x||_\infty$ is defined as the supremum norm. $\varepsilon_{\mathrm{BE}}$, ε_s and $\varepsilon_{\mathrm{Var}}$ denote the various thresholds. The vector σ_{max} stores a certain number M_{HIS} of previously calculated maximum Bayesian errors after the MLFF was renewed $\sigma_{\mathrm{max},I}$. The vector $\vec{\sigma}$ stores the error for each atom.

Input: The uncertanties $\vec{\sigma} = \mathrm{Diag}(\sigma)$ and the spilling factors $\vec{\mathbf{s}}$ on structure $\mathbf{R}_I$.
if *MLFF was renewed at previous MD step* **then**
  $\sigma_{\mathrm{max},I} = ||\vec{\sigma}||_\infty$;
  if $\sqrt{\mathrm{Var}(\sigma_{max,I})}/\mathrm{E}(\sigma_{max}) < \varepsilon_{Var}$ **then**
    $\varepsilon_{\mathrm{BE}} = \mathrm{E}(\sigma_{\mathrm{max}})$;
  end
end
if $||\vec{\sigma}||_\infty > 2\varepsilon_{BE}$ **then**
  Do FP;
else
  if *Number of passed MD steps since previous sampling* $\geq N_{INT}$ **then**
    if $||\vec{\sigma}||_\infty > \varepsilon_{BE}$ **then**
      Do FP;
    else
      Skip FP;
    end
  else
    Skip FP;
  end
end
Output: Whether to perform FP or not.

2.4.2. Data selection

When the uncertainties or the spilling factors are too large, more information is needed for that specific structure. Therefore after each FP calculation the new structure dataset is stored up to $M_{\mathrm{CONF_{NEW}}}$ temporarily. After $M_{\mathrm{CONF_{NEW}}}$ structure datasets are stored the LRC are updated and new RSDS to update the MLFF are chosen. When this selection is completed the MLFF is refitted. As the fitting process is computationally expensive, by doing this in batches of $M_{\mathrm{CONF_{NEW}}}$ resources are saved. In algorithm 3 the data selection process is outlined. It consists of two parts: first, the selection of LRC, and second, the selection of RSDS to further refine the MLFF.

The selection of LRC works as follows. For each sampled structure dataset the local configurations for each atom are computed. Then those whose uncertainties are below the threshold are discarded. Often this is not enough 'sparsification' af the data and numerical instabilities may occur. The source of those instabilities is the potential overcompleteness among the remaining local configurations. Therefore the CUR algorithm [16] is used to further thin out the local configurations

Algorithm 3: The methodology for sparsification and data selection is shown here. The matrix **K** contains the kernels of various local configurations and LRC. l_ξ denote the eigenvalues and N_{low} the number of l_ξ that are smaller than a certain threshold ε_{CUR}.

Input: Sampled structure datasets computed by FP steps.
Structure datasets $\rightarrow$ local configuration;
for *every local configuration i* **do**
  Compute the uncertainty σ_i and the spilling factor s_i;
  if $\sigma_i \leq \varepsilon_{BE}$ *or* $s_i \leq \varepsilon_s$ **then**
    Discard local configuration;
  end
end
Calculate eigenvalues l_ξ of **K**;
Determine the number N_{low} of l_ξ that are smaller than ε_{CUR};
Calculate the leverage scoring ω_i;
for *remaining local configuration i* **do**
  if ω_i *in largest* N_{low} *scorings* **then**
    Discard local configuration;
  end
end
for *sampled structure datasets i* **do**
  if *structure dataset i contain no LRC* **then**
    Discard structure dataset;
  end
end
Use all remaining structure datasets and LRC for updating the MLFF;
Output: RSDS and LRC for the MLFF.

before finally using the remaining ones as LRC.

The CUR algorithm evaluates correlations between the remaining local configurations and scores them using the parameter ω_i. Let us write the kernel from Eq. (2.22) of two local configurations i and j as $K_{i,j} = K(i,j)$. The kernel is computed for all combinations of remaining local configurations and LRC and is written as a matrix **K**. First the matrix **K** is diagonalized using its eigenvector matrix **U**

$$\mathbf{U}^T\mathbf{K}\mathbf{U} = \mathbf{L} = \text{Diag}(l_1, \ldots, l_{N_B}), \tag{2.38}$$

where l are the eigenvalues and **U** consists of the various eigenvectors $\mathbf{u}_j$ of **K**

$$\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_{N_B}), \tag{2.39}$$

$$\mathbf{u}_j = \begin{pmatrix} u_{1j} \\ \vdots \\ u_{N_B j} \end{pmatrix}, \tag{2.40}$$

where u_{ij} denotes the ith entry of the jth eigenvector or the individual entries in the matrix U.

Using this notation we can rewrite Eq. (2.38) as

$$\mathbf{K}_{ij} = \sum_{\xi=1}^{N_B} (u_{j\xi} l_\xi) u_{i\xi}, \tag{2.41}$$

We want to maintain LRC which correlate as little as possible with others. The CUR algorithm is modified to dispose columns in $\mathbf{K}$ which strongly correlate with eigenvectors $\mathbf{u}_\xi$ that have small eigenvalues l_ξ (that is, smaller then a threshold ε_{CUR}). The correlation is measured using a statistical leverage scoring function and the scoring is determined for each column j of $\mathbf{K}$

$$\omega_j = \frac{1}{N_{\text{low}}} \sum_{\xi=1}^{N_B} \gamma_{\xi j} \tag{2.42}$$

where N_{low} denotes the total number of eigenvectors $\mathbf{u}_\xi$ with $l_\xi < \varepsilon_{\text{CUR}}$ and

$$\gamma_{\xi j} = \begin{cases} u_{j\xi}^2 & \text{if } l_\xi < \varepsilon_{\text{CUR}} \\ 0 & \text{else} \end{cases}. \tag{2.43}$$

The LRC are chosen by discarding N_{low} of the largest scores ω_j and keeping the rest. During this sparsification, old LRC can be discarded. Finally, all structure datasets which contain a LRC are added to the training data. This is equivalent to keeping the smallest scores, when using a slight modification to the CUR algorithm, as shown in Appendix A.

2.5. Fitting using singular value decomposition

When computing the parameters $\bar{\mathbf{w}}$ using Bayesian regression one has to calculate the matrix $\Phi^T\Phi$. In fact, after inserting Eq. (2.29), Eq. (2.28) can be rewritten as

$$\bar{\mathbf{w}} = \left(\frac{\sigma_v^2}{\sigma_w^2} \mathbf{I} + \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{Y} \tag{2.44}$$

Inverting $\Phi^T\Phi$ is easily prone to numerical instabilities [17]. In order to explain this, imagine the matrix Φ has the form

$$\Phi = \begin{bmatrix} 1 & 1 \\ b & 0 \\ 0 & b \end{bmatrix}, \tag{2.45}$$

then

$$\Phi^T\Phi = \begin{bmatrix} 1+b^2 & 1 \\ 1 & 1+b^2 \end{bmatrix}. \tag{2.46}$$

If b is smaller than machine accuracy then this results in numerical inaccuracy and the computed parameters $\bar{\mathbf{w}}$ deviate form the optimal (accurate least squares) result. This is why a nonzero regularization term σ_v^2/σ_w^2 is introduced.

Another solution is to perform a Singular Value Decomposition (SVD) of Φ when solving the Eq. (2.24) in a least squares sense. In SVD we write the design matrix as

$$\Phi = \mathbf{U}\hat{\mathbf{S}}\mathbf{V}^T, \tag{2.47}$$

where $\Phi \in \mathbb{R}^{M \times N_B}$. $\mathbf{V} \in \mathbb{R}^{N_B \times N_B}$ and $\mathbf{U} \in \mathbb{R}^{M \times M}$ denotes the domain matrix and codomain matrix respectively. Both are orthogonal

$$\mathbf{U}^T\mathbf{U} = \mathbf{U}\mathbf{U}^T = \mathbf{I}_M, \quad \mathbf{V}^T\mathbf{V} = \mathbf{V}\mathbf{V}^T = \mathbf{I}_{N_B}. \tag{2.48}$$

The matrix $\hat{\mathbf{S}}$ of size $M \times N_B$ contain the singular values κ of Φ. There are in total ρ values and they are ordered from largest κ_1 to smallest κ_ρ. ρ corresponds to the effective rank of Φ, and is determined by counting the number of singular values that are larger than a certain threshold. The matrix has the form

$$\hat{\mathbf{S}} = \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \kappa_1 & & & \\ & \ddots & & \mathbf{0} \\ & & \kappa_\rho & \\ & \mathbf{0} & & \mathbf{0} \end{bmatrix}, \tag{2.49}$$

where $\mathbf{S}$ is a diagonal matrix of size $\rho \times \rho$ with the singular values as its diagonal entries

$$\mathbf{S} = \mathrm{Diag}(\kappa_1, \ldots, \kappa_\rho). \tag{2.50}$$

SVD solves the least squares problem by separating it in two terms, a range space part $\mathscr{R}$ and a null space part $\mathscr{N}$ [18]. $\mathscr{R}$ can be minimized while $\mathscr{N}$ remains as a residual error. $\mathbf{U}$ and $\mathbf{V}$ can be split in their range space and null space terms as shown here by rewriting Eq. (2.47)

$$\mathbf{U}\hat{\mathbf{S}}\mathbf{V}^T = \begin{bmatrix} \mathbf{U}_\mathscr{R} & \mathbf{U}_\mathscr{N} \end{bmatrix} \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{V}_\mathscr{R}^T \\ \mathbf{V}_\mathscr{N}^T \end{bmatrix}, \tag{2.51}$$

where $\mathbf{U}_\mathscr{R}$ and $\mathbf{V}_\mathscr{R}$ of size $M \times \rho$ and $N_B \times \rho$ respectively. Their null space counterparts are of size $M \times (M - \rho)$ for $\mathbf{U}_\mathscr{N}$ and of size $N_B \times (N_B - \rho)$ for $\mathbf{V}_\mathscr{N}$. The least squares problem for Eq. (2.24) is solved by $\mathbf{w}$ which minimizes the residues $\mathbf{r}$ written as

$$\min(\mathbf{r}^2) = \min\left(||\Phi\mathbf{w} - \mathbf{Y}||_2^2\right), \tag{2.52}$$

where $||\cdot||_2$ defines the 2-norm. By using the invariance to orthogonal transformations of $||\cdot||_2$ and the SVD of Φ the residues can be rewritten as

$$\mathbf{r}^2 = ||\mathbf{U}^T(\Phi\mathbf{w} - \mathbf{Y})||_2^2 = ||\hat{\mathbf{S}}\mathbf{V}^T\mathbf{w} - \mathbf{U}^T\mathbf{Y}||_2^2. \tag{2.53}$$

The residues can be separated into range and null space terms by inserting the relations of Eq. (2.51) which finally leads to

$$\mathbf{r}^2 = ||\mathbf{S}\mathbf{V}_\mathscr{R}^T\mathbf{w} - \mathbf{U}_\mathscr{R}^T\mathbf{Y}||_2^2 + ||\mathbf{U}_\mathscr{N}^T\mathbf{Y}||_2^2. \tag{2.54}$$

The first term, only containing range space components can be minimized by setting it equal to zero, whereas the second one containing only a null space component will remain as a residual error. The least squares solution of $\mathbf{w}$ when using SVD is thus determined to be

$$\mathbf{w} = \mathbf{V}_\mathscr{R}\mathbf{S}^{-1}\mathbf{U}_\mathscr{R}^T\mathbf{Y}. \tag{2.55}$$

This solution is computationally more expensive than the Bayesian least squares solution (Eq. (2.28)), but numerically more stable. In practice, it is generally found that Φ has full rank N_B, in contrast to $\Phi^T\Phi$ which requires regularization. Because of its greater cost SVD is not used in the

on-the-fly learning routine but, could be used to get a more accurate MLFF when the training is complete.

It is useful to make a connection between SVD and the Bayesian regression method. The least-squares solution using SVD can also be written as:

$$\mathbf{w} = \sum_{i=1}^{N_B} \frac{\mathbf{u}_i^T \mathbf{Y}}{\kappa_i} \mathbf{v}_i, \tag{2.56}$$

where u_i and v_i are the ith columns of the matrices U and V respectively. Eq. (2.55) is restored if we include in the sum only the ρ singular values that are larger than a certain threshold. This is usually termed truncated SVD. Another method to treat an ill-conditioned matrix Φ, and thus small singular values, is regularization [19]. To accomplish this, we introduce filtering factors θ_i and rewrite Eq. (2.56) as

$$\mathbf{w}_{\text{regular}} = \sum_{i=1}^{N_B} \theta_i \frac{\mathbf{u}_i^T \mathbf{Y}}{\kappa_i} \mathbf{v}_i. \tag{2.57}$$

The filtering factors are chosen such that $\theta_i \approx 0$ for small singular values, and $\theta_i \approx 1$ for larger ones. In the Tikhonov method, the filtering factors are

$$\theta_i = \frac{\kappa_i^2}{\kappa_i^2 + \alpha^2}, \tag{2.58}$$

where α is the regularization parameter. The Tikhonov method is a smoother approach compared to the truncated SVD discussed before. Additionally it prevents oscillations to the solution. It should be added that Eq. (2.56) is formally equivalent to Eqs. (2.28)-(2.29) with $\alpha^2 = \sigma_v^2/\sigma_w^2$.

3. Training

In this section we will focus on how the training data for silicon are generated and discuss the parameters for fitting the FF. There are two main drawbacks of MLFF which need to be considered when generating the training data. First, MLFF are much better at interpolating than extrapolating. In order to generate a reliable multipurpose potential, therefore one has to train over a wide range of the configuration space. Second, generating new FP data is computationally expensive. Performing too much training on a specific set of configurations can worsen the predictive power on other configurations, as shown in chapter 4. A careful balance between too many and too little training data needs to be kept. To accomplish this, on-the-fly training with Bayesian error estimation is used, as detailed in chapter 2.

In order to describe multiple phases of silicon a wide range of known structures is used for training. We start with the ground-state cubic diamond (cd) and the closely related hexagonal diamond (hd) as well as stable high-pressure phases bc8[20], β-Sn and simple hexagonal (sh). Additionally the even higher pressure non-stable phases face-centered cubic (fcc), body-centered cubic (bcc), hexagonal close packed (hcp), and hcp' are added to the training. The difference between hcp and hcp' is the ratio of lattice parameters. The hcp phase has a ratio of $c/a \approx \sqrt{3/2}$ while hcp' has a much lower ratio of $c/a < 1$. All bulk structures are well relaxed initially. To include also defect configurations the MLFF various interstitial sites based on cubic diamond are added to the database. Listed in descending energetic stability: dumbbell configuration (X), hexagonal hollow (H), C_{3v} (lower symmetry hexagonal interstitial) and tetragonal site (T)[21]. Furthermore a Jahn-Teller distorted vacancy configuration (JTV) is added to the training set. All structures mentioned so far are computed using a 64 atom supercell. In case of the vacancy/interstitial site, one atom is added or removed from the structure. Finally, to include surfaces to the database, three cubic diamond nano-crystals are added with combinations of the surfaces $(1\,1\,1)$ and $(1\,0\,0)$: one with 136 atoms and both $(1\,1\,1)$ and $(1\,0\,0)$ ($\text{cdc}_{(1\,1\,1)\&(1\,0\,0)}$), another with only $(1\,1\,1)$ and 148 atoms ($\text{cdc}_{(1\,1\,1)}$), and finally one with 139 atoms and only $(1\,0\,0)$ surfaces ($\text{cdc}_{(1\,0\,0)}$).

3.1. First Principal Parameters

The FP code VASP is used for the data generation and the machine learning. For the Density Functional Theory (DFT) calculations a planewave cutoff energy of $E_{\text{cut}}350\,\text{eV}$ and $0.05\,\text{eV}$ Gaussian smearing is chosen. The electronic degrees of freedom are relaxed until the total and band structure energy change is smaller than $10^{-5}\,\text{eV}$. The calculations are performed using the Perdew-Burke-Ernzerhof (PBE) functional. A k-point grid centered at the Γ-point is generated to sample the Brillouin zone. The smallest allowed spacing between k-points is set to $0.15\,\text{Å}^{-1}$,

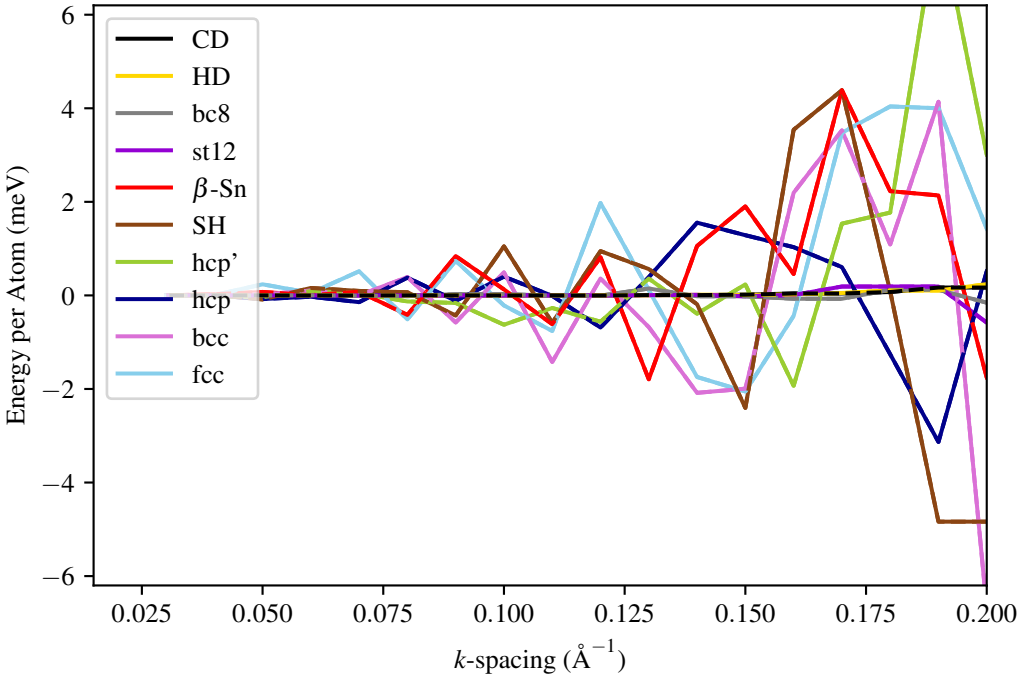


Figure 3.1.: The energy difference per atom vs smallest allowed spacing between k-points for various bulk crystal lattice structures. The k-scan for the cd and hd structures can be viewed in more detail in Figure B.1 in Appendix B.

except for the diamond structures for which we use a smallest spacing of 0.2 Å^{-1}. For the nano crystals, we use only a single Γ-point. Figure 3.1 shows the energy difference per atom versus the k-point spacing. The diamond structures converge faster, therefore a larger k-spacing is used in order to save computational time. In Figure 3.1 one can see that the high-pressure structures do not fully converge with a smallest allowed spacing between k-points of 0.15 Å^{-1}. However, our aim is to examine how well on-the-fly learning is suitable for creating a multi purpose potential, hence low computing time is preferred over high precision. The structures are also trained under pressure, therefore a k-point scan with 10 % smaller structures is performed. It returns similar results as with relaxed cells hence the plot is not shown here.

3.2. MLFF Fitting Parameters

For fitting the MLFF a cut-off radius for the radial and angular descriptors of R_{cut} =5 Å is used. To broaden the atomic distributions for both the radial and angular descriptors a width of $\sigma_{\text{atom}} = 0.4$ Å for the Gaussian functions is chosen (see Eq. (2.3), Eq. (2.4), and Eq. (2.5)). The angular kernel is a fourth order polynomial $\zeta = 4$ (see Eq. (2.22)). The maximum angular momentum quantum number of spherical harmonics is used to expand the atomic distributions is $L_{\text{max}} = 4$. The angular

filtering function is used

$$\eta = \frac{1}{1 + a[l(l+1)]^2}, \tag{3.1}$$

with $a = 0.002$. Furthermore the radial descriptors are weighted four times stronger than the angular descriptors by setting $\beta^{(2)} = 0.8$ and $\beta^{(3)} = 0.2$ (see Eq (2.21)).

The initial criteria for the Bayesian error estimation is set to $\varepsilon_{\mathrm{BE}} = 10^{-16}$ at the start of each on-the-fly training run. The threshold for the variance in the stored estimated errors is set to $\varepsilon_{\mathrm{Var}} = 0.4$ respectively. This threshold was exceeded multiple times during the training. The number of estimated errors stored in memory to determine the criteria is chosen to be $M_{\mathrm{HIS}} = 5$. Also the number of stored structure datasets and the minimum number of MD steps that have to pass before changing the Bayesian error threshold are both set to $N_{\mathrm{INT}} = M_{\mathrm{CONF_{NEW}}} = 5$. The threshold used in the CUR algorithm is chosen to be $\varepsilon_{\mathrm{CUR}} = 10^{-12}$. For details see section 2.4.

Before fitting, the unnormalized training data are divided by their respective weights, which are chosen to be the standard deviation of the data during the first training run. In particular, the weight used for the stress is 10 kbar and for the forces we used 1 eV Å^{-1}. When weighting the total energies we should note that using the standard deviation does not show the best results. Therefore the weight is 50 times larger, specifically 1.5 meV/atom. We determine this value by computing the mean squared errors in total energy for training and test structures for different energy weights. This is shown in Table 3.1. We choose the weight which gives the best predictive power for the test data. When using a weight larger than 1.5 meV/atom both the error in the training set and the testing set increases, while using a smaller weight leads to overfitting of the energy equation. This manifests in larger error in the testing set, while the error for the training set further decreases. To obtain the test data sets, we performed an MD heating run with 10000 time steps, using the MLFF. By performing an independent MD run we can guarantee that different configurations are generated than during the on-the-fly training. For each phase 50 configurations are chosen, which are selected every 200th MD time step. For each configuration DFT energies are computed for comparison.

Table 3.1.: The mean squared errors in total energies for the training and test data set in dependence of the scaling weight.

Weight Total Energy [meV/atom]	12.000	6.000	3.000	1.500	0.750	0.375	0.188
Error in Training set [meV/atom]	1.551	1.209	0.936	0.704	0.521	0.398	0.314
Error in Test set [meV/atom]	1.677	1.543	1.497	1.476	1.500	1.605	1.795

At this point we should emphasize that the MLFF can be optimized, depending on the desired application, by weighting the energies, forces and stress tensors before fitting accordingly. After all training is complete the MLFF is refitted using SVD (explained in section 2.5).

3.3. On-the-fly generated Database

In order to have a wide range of training data that is as close to the desired applications as possible, MD simulations are performed in combination with on-the-fly machine learning. Bayesian error estimation is used to determine whether to skip ab-initio steps and instead use the MLFF obtained up to this point, or to perform a DFT calculation and further refine the FF.

Table 3.2.: The number of RSDS per structure type, the total number of LRC and the detailed training order are listed starting from the beginning.

Structure type	Temperature range	Number of RSDS	Total number of LRC
CD at 0 GPa	100K to 1500K	333	398
CD at 10 GPa	100K to 1500K	362	733
CD at 5 GPa	100K to 1500K	487	1044
HD at 0 GPa	100K to 1500K	239	1251
β-Sn at 0 GPa	100K to 1000K	261	1556
β-Sn at 10 GPa	100K to 1000K	307	1759
β-Sn at 5 GPa	100K to 1000K	453	2024
SH at 0 GPa	100K to 1000K	342	2254
SH at 10 GPa	100K to 1000K	377	2421
SH at 5 GPa	100K to 1000K	387	2570
HD at 10 GPa	100K to 1500K	395	2831
bc8 at 0 GPa	100K to 1000K	445	3190
bc8 at 10 GPa	100K to 1000K	455	3495
hcp' at 0 GPa	50K to 200K	102	3525
hcp' at 10 GPa	50K to 200K	102	3534
hcp at 0 GPa	50K to 200K	33	3564
hcp at 10 GPa	50K to 200K	122	3463
bcc at 0 GPa	50K to 200K	29	3581
bcc at 10 GPa	50K to 200K	31	3609
fcc at 0 GPa	50K to 200K	148	3717
fcc at 10 GPa	50K to 200K	23	3747
V at 0 GPa	100K to 1500K	70	3766
H at 0 GPa	100K to 1500K	140	3843
T at 0 GPa	100K to 1500K	155	3908
C3V at 0 GPa	100K to 1500K	125	3958
X at 0 GPa	100K to 1500K	185	4067
$\mathrm{cdc}_{(111)\&(100)}$ at 0 GPa	100K to 1500K	91	4255
JTV at 0 GPa	100K to 1500K	120	4253
$\mathrm{cdc}_{(100)}$ at 0 GPa	100K to 1000K	91	4489
$\mathrm{cdc}_{(111)}$ at 0 GPa	100K to 1000K	230	4746
	Total:	6640	4746

To obtain a wide range of configurations an NpT ensemble controlled by a Langevin thermostat[22] is used to heat different bulk systems, defective supercells and nano-crystals. For the atomic

degrees-of-freedom a friction coefficient of $10\,\mathrm{ps}^{-1}$ is used. The lattice degrees-of-freedom are controlled by the Parrinello-Rahman method with a fictitious mass of 100 u and friction coefficient of $3\,\mathrm{ps}^{-1}$. The MD simulations are performed using 10 000 time steps of 3 fs. For the unstable high pressure phases hcp, hcp', fcc and bcc the number of steps is reduced to 1000, since these structures are not stable during MD heating runs and a fast 'undesired' phase transition occurs. In order to guarantee that no large cell distortions occur, different heating rates are chosen, as shown in Table 3.2. This is important, because the plane-wave basis set for the ab-initio calculations is computed at the beginning of the MD simulations, according to the following equation:

$$\frac{\hbar^2}{2m_e}|\mathbf{G}+\mathbf{k}|^2 < E_{\mathrm{cut}}, \tag{3.2}$$

where $\hbar$ is the reduced Plank constant, m_e the mass of an electron, $\mathbf{k}$ denotes the Bloch wave vector, and $\mathbf{G}$ is a point on the reciprocal $\mathbf{G}$-grid. Therefore, if the cell basis vectors change significantly, one would need to recalculate the plane-wave basis set, in order to avoid a large error.

The training is started with the bulk structures in order of stability with the exception of HD (for details see Table 3.2). Each bulk structure is trained at zero pressure and at 10 GPa. To expand the training data for CD and the stable high pressure phases SH and β-Sn additional MD simulations at 5 GPa are performed. Then the vacancy site followed by the interstitial sites is added to the database, this time starting with the energetically most unstable. After that the various nanocrystals are trained. In Table 3.2 the number of RSDS per structure type, the total number of LRC and the detailed training order are listed. One should keep in mind that during the MD heating runs, phase transitions occurred for the unstable high pressure phases and β-Sn which converted to SH at temperatures $>$600 K. Also the T interstitial site changes to an energetically more favorable one.

For the JTV the automatic criteria determination is disabled because when used too few RSDS are added. A fixed criterion is set a posteriori in order to guarantee continued training. The criterion is set rather tight to $\varepsilon_{\mathrm{BE}} = 0.005$.

3.4. Supplemental RSDS

For phases that are unstable during MD simulations the MLFF contain insufficient information to make meaningful predictions. In our case, such phases are the high pressure ones hcp, hcp', fcc, and bcc. Additional RSDS are created in order to supplement the MLFF.

The new RSDS are generated by 'mirroring' the atomic displacements of the CD phase during the MD heating simulation onto the high pressure phases. In practice, the relative displacement of a reference atom in CD is mapped onto an atom in each of the hcp, fcc, and bcc structures. A small code to perform this mirroring is shown in Appendix C. A total of 600 RSDS are generated in this way, 200 for each high pressure phase, which are equality split between 0 GPa and 10 GPa. Figure 3.2 shows the resulting total potential energy per atom distribution for all the structures generated by this method.

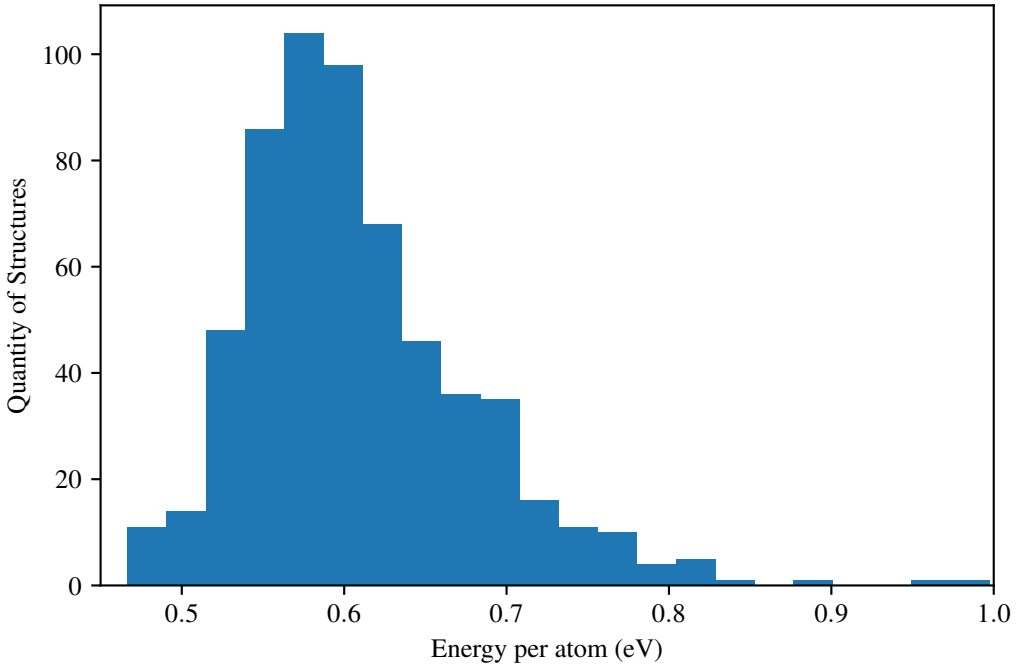


Figure 3.2.: The potential energy per atom distribution of the additional RSDS generated by 'mirroring' on MD simulation, referenced to the lowest total potential energy per atom of CD.

DFT calculations are performed in the same manner as during the on-the-fly training, while the Bayesian error criteria are set much tighter to guarantee that nearly all structures are included in the MLFF generation. The threshold for the spilling factors is not changed. The training is performed in six batches of 100 structures for each phase and their different pressures. For every phase all 100 structures are added to the existing MLFF except for fcc and hcp at 0 GPa. For those phases only 97 and 99 structures are added respectively. No LRC are transferred to the final MLFF. The final MLFF is refitted using SVD and contains a total of 7236 RSDS and 4745 LRC. One LRC is discarded during the final fit.

4. Testing the MLFF

We conduct a series of basic tests using the MLFF which are described in this section. We start by exploring the predictive capability of the MLFF on the various bulk crystal phases of silicon. Next we use the MLFF to calculate formation energies of different interstitial and vacancy sites. In addition to the defects, surfaces are also tested. Finally the phonon bands of the CD phase are computed. To compare the results FP calculations are performed using DFT.

4.1. Bulk crystals

For the various bulk crystal phases we perform three tests. First the energies for distorted structures are computed. Second we perform a volume scan for the different bulk crystal phases and observe the energy-volume dependency. Third and last we calculate the elastic properties of the CD, HD, and β-Sn phases.

4.1.1. Bulk energies

As an initial test, we calculate the bulk energies for many, according to the temperature, distorted structures of CD, HD, bc8, β-Sn, SH, hcp', hcp, bcc, and fcc phases. The structures are generated by heating up a relaxed structure for each phase using the MLFF. The temperature range is chosen to be the same for each phase as during the training. The heating run produces 10000 distorted structures per phase, from which 50 are chosen for testing. These structures are selected evenly distributed along the total simulation time. For comparison the energies of these structures are computed from FP. In order to be comparable we use the same DFT settings as used during the training of the MLFF. We calculate the error in energy per atom for each structure. This is done by subtracting the energy derived using the MLFF from the energy computed using FP and dividing it by the total number of atoms in the associated structure. For each bulk phase we calculate the mean ($\langle E_{err} \rangle$), the absolute mean ($\langle |E_{err}| \rangle$), the maximum ($\max(E_{err})$), the minimum ($\min(E_{err})$), and the standard deviation of the error ($\sigma(E_{err})$). The errors are computed for each MLFF trained, that is, after each new phase is added to the on-the-fly training. In Table 4.1 the errors are displayed.

One can see that the accuracy worsens with each new phase added. The exceptions are HD when the β-Sn phase is added and SH when bc8 is added. The worsening of the MLFF is less severe when the phases are similar to each other. For this test we did not use the final MLFF, therefore

Table 4.1.: The mean ($\langle E_{err} \rangle$), absolute mean ($\langle |E_{err}| \rangle$), maximum ($\max(E_{err})$), minimum ($\min(E_{err})$), and standard deviation ($\sigma(E_{err})$) of the errors for different phases are shown here. The errors are computed after each new phase is added to the MLFF. This is represented in the columns starting on the right with CD. The rows contain the errors of the different phases. The errors are in units of meV per atom.

Trained Phase:		fcc	bcc	hcp	hcp'	bc8	SH	β-Sn	HD	CD
	$\langle E_{err} \rangle$	0.69	0.74	0.70	0.70	0.69	0.53	0.49	0.42	0.24
	$\langle \lvert E_{err} \rvert \rangle$	1.04	1.06	1.02	1.01	0.98	0.89	0.86	0.81	0.70
CD	$\min(E_{err})$	−3.11	−3.03	−2.96	−2.96	−3.08	−2.81	−2.86	−2.89	−3.01
	$\max(E_{err})$	5.36	5.27	5.22	5.15	5.06	3.99	3.72	3.00	2.40
	$\sigma(E_{err})$	1.30	1.30	1.28	1.27	1.24	1.10	1.09	1.01	0.97
	$\langle E_{err} \rangle$	0.69	0.61	0.61	0.63	0.66	0.62	0.43	0.61	
	$\langle \lvert E_{err} \rvert \rangle$	1.26	1.18	1.17	1.18	1.13	1.15	1.09	1.14	
HD	$\min(E_{err})$	−3.15	−2.89	−2.85	−3.09	−2.67	−2.07	−2.54	−2.18	
	$\max(E_{err})$	5.70	4.80	4.69	4.69	3.76	3.34	2.95	3.73	
	$\sigma(E_{err})$	1.63	1.44	1.42	1.41	1.29	1.22	1.24	1.27	
	$\langle E_{err} \rangle$	−0.28	−0.22	−0.27	−0.22	−0.23	−0.39	0.25		
	$\langle \lvert E_{err} \rvert \rangle$	1.40	1.28	1.27	1.26	1.13	1.12	1.09		
β-Sn	$\min(E_{err})$	−3.39	−3.19	−3.34	−3.23	−3.13	−3.05	−3.20		
	$\max(E_{err})$	4.19	3.52	3.45	3.45	2.70	3.14	4.84		
	$\sigma(E_{err})$	1.67	1.56	1.54	1.54	1.37	1.33	1.45		
	$\langle E_{err} \rangle$	−0.00	−0.16	−0.21	−0.19	−0.20	−0.31			
	$\langle \lvert E_{err} \rvert \rangle$	1.68	1.52	1.51	1.47	1.44	1.46			
SH	$\min(E_{err})$	−3.04	−2.71	−2.83	−2.77	−2.61	−2.77			
	$\max(E_{err})$	8.22	8.27	8.14	8.00	8.63	8.90			
	$\sigma(E_{err})$	2.18	2.02	1.99	1.96	1.96	2.00			
	$\langle E_{err} \rangle$	0.29	0.46	0.41	0.38	0.44				
	$\langle \lvert E_{err} \rvert \rangle$	2.65	2.28	2.30	2.24	2.28				
bc8	$\min(E_{err})$	−5.69	−5.06	−5.12	−5.00	−5.13				
	$\max(E_{err})$	19.59	15.05	15.03	15.10	16.58				
	$\sigma(E_{err})$	4.40	3.66	3.66	3.61	3.79				
	$\langle E_{err} \rangle$	0.19	0.09	−0.22	−0.11					
	$\langle \lvert E_{err} \rvert \rangle$	0.29	0.27	0.36	0.29					
hcp'	$\min(E_{err})$	−0.33	−0.57	−1.19	−0.95					
	$\max(E_{err})$	1.21	0.93	0.57	0.55					
	$\sigma(E_{err})$	0.32	0.32	0.38	0.36					
	$\langle E_{err} \rangle$	1.84	1.23	0.98						
	$\langle \lvert E_{err} \rvert \rangle$	1.89	1.47	1.16						
hcp	$\min(E_{err})$	−0.54	−1.99	−1.71						
	$\max(E_{err})$	22.06	13.65	6.99						
	$\sigma(E_{err})$	3.50	2.21	1.25						
	$\langle E_{err} \rangle$	−5.07	−12.35							
	$\langle \lvert E_{err} \rvert \rangle$	5.65	12.35							
bcc	$\min(E_{err})$	−15.37	−19.40							
	$\max(E_{err})$	4.81	−1.31							
	$\sigma(E_{err})$	4.76	4.55							
	$\langle E_{err} \rangle$	−0.95								
	$\langle \lvert E_{err} \rvert \rangle$	1.17								
fcc	$\min(E_{err})$	−7.95								
	$\max(E_{err})$	2.32								
	$\sigma(E_{err})$	1.89								

for the high pressure phases the MLFF contains very little information and the predictive power is not as good as for the other phases.

In Table 4.2 the mean ($\langle E_{err} \rangle$), absolute mean ($\langle |E_{err}| \rangle$), maximum ($\max(E_{err})$), minimum ($\min(E_{err})$), and standard deviation ($\sigma(E_{err})$) of the errors in the bulk energies using the final MLFF are shown. We use the same test dataset of CD, HD, bc8, β-Sn, SH, hcp', hcp, bcc, and fcc structures as for the previous error calculation. The energies here need to be compared with the first column in Table 4.1.

Table 4.2.: The mean ($\langle E_{err} \rangle$), absolute mean ($\langle |E_{err}| \rangle$), maximum ($\max(E_{err})$), minimum ($\min(E_{err})$), and standard deviation ($\sigma(E_{err})$) of the errors for different phases are shown here. The errors are computed using the final MLFF. The errors are in units of meV per atom.

Phase:	CD	HD	β-Sn	SH	bc8	hcp'	hcp	bcc	fcc
$\langle E_{err} \rangle$	0.50	0.27	-0.12	0.12	0.09	-0.01	1.63	-2.09	0.77
$\langle \lvert E_{err} \rvert \rangle$	0.73	0.75	1.00	1.15	1.21	0.14	1.82	3.94	1.21
$\min(E_{err})$	-1.89	-3.00	-2.60	-1.78	-4.02	-0.49	-2.06	-12.04	-2.61
$\max(E_{err})$	3.03	3.15	3.03	3.88	6.58	0.52	2.89	8.24	4.88
$\sigma(E_{err})$	0.85	0.98	1.27	1.48	1.86	0.19	1.19	4.65	1.21

Overall the final MLFF is more accurate than the intermediate MLFF. This seems contrary to what we observed previously, but we used SVD to fit the final MLFF which gives overall more accurate results. Hence this clearly demonstrates how beneficial SVD is. For instance, $\max(E_{err})$ is consistently reduced, except for the unstable bcc and fcc phases.

4.1.2. Volume scan

As the second test we compute the energy dependence with respect to the volume for different bulk crystal phases, including st12 which is similar to bc8 but slightly more complex [20]. The st12 phase is not included in the training data. It is used to explore the predictive capabilities of the MLFF when extrapolating. The volume scan is performed by changing the volume of a unit cell of a certain phase. Then the ions are relaxed using a conjugate-gradient method until the forces are smaller than 1 meV Å^{-1}. The cell shape is also relaxed while keeping the volume fixed. The volume is scanned between $\pm 10\%$ of the relaxed volume of the associated phase. For comparison the same is done using DFT. Except for a tighter k-spacing the same DFT settings as for the training are used: the k-spacing was changed for the β-Sn and SH phases because instabilities occurred during the volume scan when using the same k-spacing as during training. Figure 4.1 shows the resulting energies versus volume.

Even though the st12 structure is very similar to the bc8 structure, the MLFF shows a clear discrepancy to the FP calculation. One can also observe a small discrepancy of the MLFF and the DFT results when expanding the volume. This is due to the MLFF not being trained under expansion, but only under positive pressure. Compared to the error in the curve for the st12 phase the discrepancy for expansion is however small.

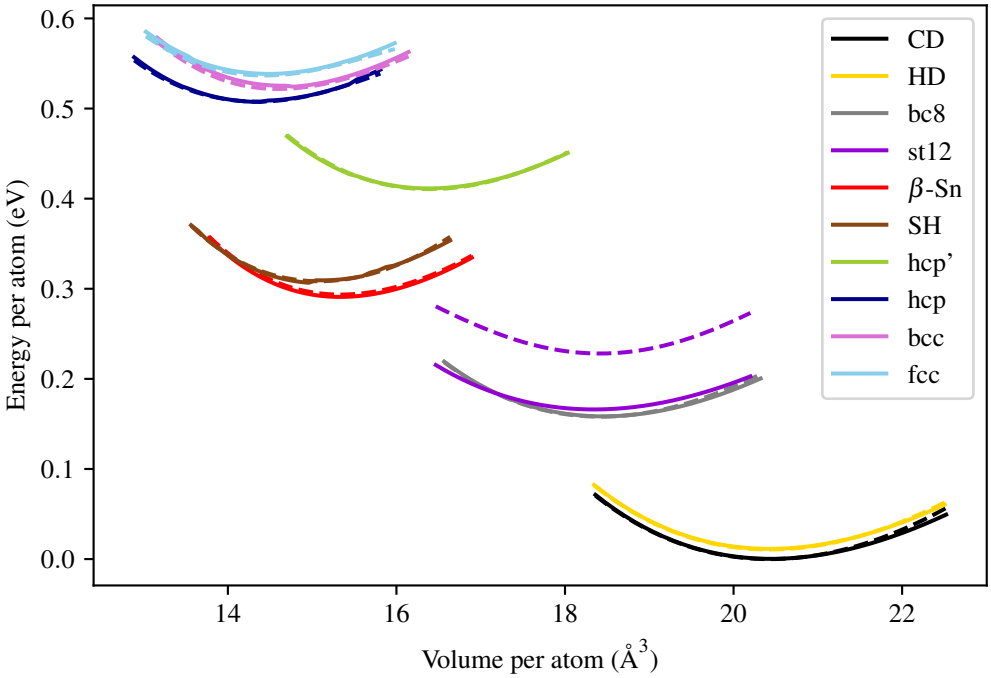


Figure 4.1.: Energy per atom versus volume for different bulk crystal phases. The dashed lines represent the energies computed using the MLFF while the solid lines are computed using DFT. The st12 structure represented by the purple lines is not included in the training and thus extrapolated. Note that this yields unsatisfactorial results for st12.

4.1.3. Elastic modulus

As final test for the bulk crystals, we compute the elastic moduli for the CD, SH, and β-Sn phases. The elastic moduli are computed by a multiplication of the inverted ionic Hessian matrix and the internal strain tensor [23]. To compute the Hessian matrix four displacements in each cartesian direction are used. The displacements are of size ± 0.01 Å and ± 0.02 Å. For comparison the elastic moduli are computed using DFT with the same settings as for the training. Due to the crystal symmetries of CD the non-zero components of the elastic moduli are $C_{11} = C_{22} = C_{33}$, $C_{12} = C_{13} = C_{23}$, and $C_{44} = C_{55} = C_{66}$. For SH and β-Sn the symmetries and non-zero components are $C_{11} = C_{22}$, C_{12}, $C_{13} = C_{23}$, C_{33}, C_{44} and, $C_{55} = C_{66}$.The different elastic moduli are shown in Table 4.3.

There is a larfe discrepancy between the elastic moduli computed by DFT and the MLFF for the β-Sn phase. Most likely this is due to an insufficient k-point density during training. Figure 3.1 shows the difference in energy vs smallest allowed spacing between k-points. For β-Sn we used a k-spacing of 0.15 Å^{-1} (corresponding to a k-point grid of $9 \times 9 \times 16$) for which the energy difference is greater than 2 meV per atom. Table 4.4 shows the elastic moduli of β-Sn computed by DFT using different k-point grids. This further indicates that a denser k-point sampling is

Table 4.3.: The different elastic moduli for the phases CD, SH, and β-Sn are shown here. Only non-zero and non symmetric entries are displayed. $\mathrm{DFT}_{\mathrm{acc}}$ uses a k-point grid of $33 \times 33 \times 60$. The elastic moduli are in kbar.

Phase	Method	C_{11}	C_{12}	C_{13}	C_{33}	C_{44}	C_{55}
CD	DFT	1533	567			741	
	MLFF	1550	632			714	
HD	DFT	1822	489	336	2060	666	491
	MLFF	1782	495	467	1800	644	450
β-Sn	DFT	2437	406	571	1933	779	796
	$\mathrm{DFT}_{\mathrm{acc}}$	1706	936	740	1562	815	630
	MLFF	1205	1469	772	1405	753	568

needed during training in order to predict the elastic moduli of β-Sn.

Table 4.4.: Convergence of the elastic moduli in kbar for the β-Sn phase with respect to k-point grids.

k-points	C_{11}	C_{12}	C_{13}	C_{33}	C_{44}	C_{55}
$9 \times 9 \times 16$	2437	406	571	1933	779	796
$10 \times 10 \times 18$	2337	874	441	1829	933	872
$11 \times 11 \times 20$	1961	418	990	1043	558	589
$12 \times 12 \times 20$	1934	649	773	1571	557	592
$13 \times 13 \times 24$	1647	1234	618	1627	769	549
$23 \times 23 \times 42$	1622	1004	766	1485	828	584
$33 \times 33 \times 60$	1706	936	740	1562	815	630

Astoundingly the MLFF for β-Sn is closer to the final k-point converged DFT result ($\mathrm{DFT}_{\mathrm{acc}}$) than the DFT calculation using the same k-spacing as for the training. Because the MLFF was trained at finite temperature, the single electron energies are smeared, therefore the results are less sensitive to the k-point sampling.

4.2. Defects

For testing defects we computed the formation energies for the one vacancy and the different interstitial sites which are included in the training. Additionally we explored the predictive capabilities of the MLFF in calculating surface energies.

4.2.1. Point defects: Formation energies

The formation energies of the JTV, H, T, C_{3V}, and X interstitial sites are computed in order to test the MLFF further. This is done by calculating the energy of a CD bulk structure, E_{bulk}, with $N_{\text{bulk}} = 64$ atoms, and the energy of the structure containing the vacancy/interstitial site of interest E_{def}. To finally compute the formation energy E_{form} the following equation is used

$$E_{\text{form}} = E_{\text{def}} - \frac{N_{\text{def}}}{N_{\text{bulk}}} E_{\text{bulk}}. \tag{4.1}$$

Both energies E_{def} and E_{bulk} are computed using the MLFF. The ionic degrees of freedom are relaxed until the change in total energy is smaller than 1×10^{-4} eV. For relaxation the RMM-DIIS algorithm is used [24]. This is a quasi-Newton (variable metric) method. A maximum of ten ionic steps are stored in the iteration history for approximating the Hessian matrix. As reference, the formation energies are computed in the same manner using DFT. For DFT the same settings as for the MLFF training are used. The results are displayed in Table 4.5. For comparison the formations energies are computed using different versions of the MLFF.

Table 4.5.: The formation energies for the one vacancy and the different interstitial sites are shown here. They are in units of eV. The two additional MLFF used are $\text{MLFF}_{\text{BayReg}}$ and MLFF_{-}. $\text{MLFF}_{\text{BayReg}}$ is the original MLFF but fitted using Bayesian regression, while MLFF_{-} does not include the supplemental RSDS described in Section 3.4. It is fitted like the original one using SVD.

Method	X	T	H	C_{3V}	JTV
DFT	3.614	3.790	3.658	3.651	3.642
MLFF	3.628	3.670	3.627	3.595	3.627
$\text{MLFF}_{\text{BayReg}}$	3.626	3.677	3.590	3.574	3.375
MLFF_{-}	3.597	3.663	3.620	3.617	3.629

The differences between DFT and MLFF are mostly very small. The error is largest for the T interstitial site. This is probably because during the on-the-fly training this interstitial site was not stable and changed to an energetically more favorable configuration, therefore the MLFF contain less information on the T interstitial site. For JTV one can see a large discrepancy between the MLFF fitted using SVD and Bayesian regression, with SVD yielding a better result.

4.2.2. Surfaces

The predictive capabilities of the MLFF on surfaces are tested by computing the total energies while a gap is opened in a supercell elongated in one direction, exposing either the $(1\,0\,0)$ or $(1\,1\,1)$ surfaces. For both surfaces the gap is opened in 30 continuous steps by 3 Å. The energy calculations are performed with no relaxation. As reference the same calculations are recreated using DFT with the same settings used during training. The results are shown in Figure 4.2.

According to the Wulff construction, surfaces with lower surface energy are more exposed on crystals [25]. Therefore, it is unlikely that during the nanocrystal on-the-fly training the initial

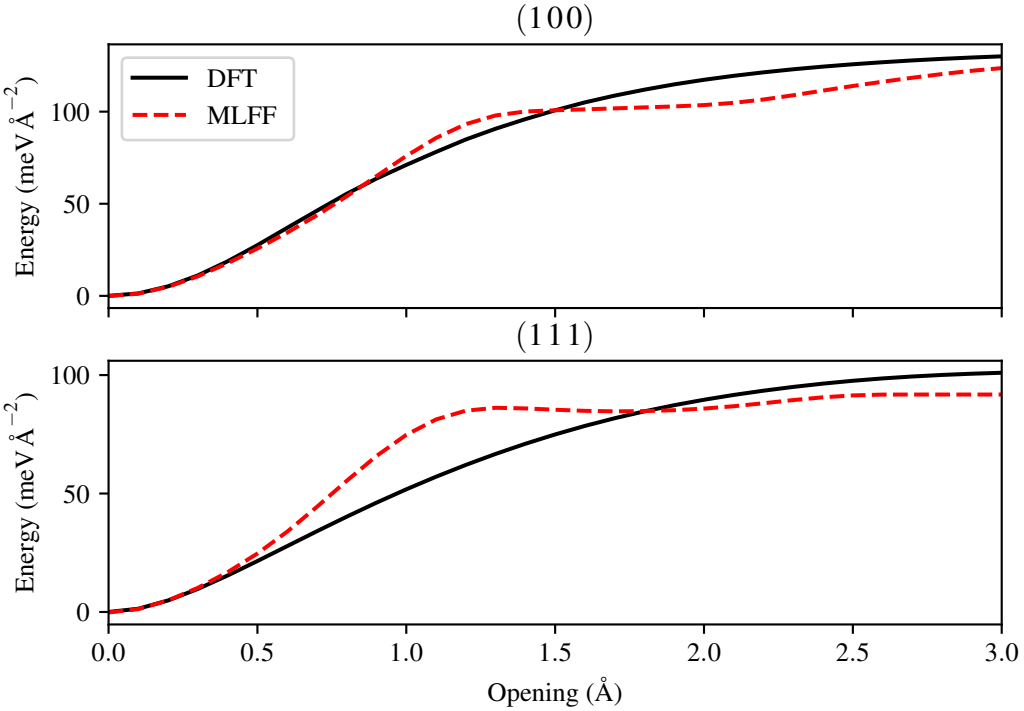


Figure 4.2.: The surface energies computed while opening a gap in the $(1\,0\,0)$ and $(1\,1\,1)$ directions are shown here. The solid black lines are the DFT results while the dashed red lines are obtained using the MLFF.

surfaces have remained intact. When checking with a 3D atomic visualization software, exactly this behavior can be observed. The $(1\,1\,1)$ surface is energetically more favorable than the $(1\,0\,0)$ surface. As a result, the MLFF should contain more information on the $(1\,1\,1)$ surface and should therefore have lower accuracy on the $(1\,0\,0)$ surface. When looking at Figure 4.2 it seems that the MLFF is more accurate for the $(1\,0\,0)$ surface, but this is misleading. There are no training data containing gaps openings, therefore the MLFF is extrapolating during this test. When comparing different surface structures computed using the MLFF and DFT, the average error in the surface energy for the $(1\,0\,0)$ and $(1\,1\,1)$ surface is very similar, despite the MLFF containing more information on the $(1\,1\,1)$ surface. The errors are summerized in Table 4.6. The error is computed for 50 different surface structures obtained during an MD run at 500 K. A total of 10000 steps are performed and 50 structures, evenly spaced along the time steps, are selected. The structures for the $(1\,0\,0)$ surface have 72 atoms and for the $(1\,1\,1)$ surface 48 atoms. Both have a vacuum width of 6 Å to separate the periodic cell.

Clearly, by training on the nanocrystals, our MLFF are not highly accurate for surfaces. In hindsight, the strategy to train on nanocrystals instead of surface unit cells was not particularly great.

Table 4.6.: The mean ($\langle E_{err} \rangle$), absolute mean ($\langle |E_{err}| \rangle$), maximum ($\max(E_{err})$), minimum ($\min(E_{err})$), and standard deviation ($\sigma(E_{err})$) of the errors in surface energy between DFT and MLFF are shown here.

meV Å^{-2}	(111)	(100)
$\langle E_{err} \rangle$	-5.77	5.89
$\langle \lvert E_{err} \rvert \rangle$	5.77	5.89
$\min(E_{err})$	-8.69	3.41
$\max(E_{err})$	-0.92	8.00
$\sigma(E_{err})$	1.82	1.14

4.3. Phonons

As final test, we have computed the phonon band structure of cubic diamond silicon at zero pressure using a finite displacement method in a $4 \times 5 \times 5$ supercell. For the calculation we used phonopy [26]. The phonon band structure is displayed in Figure 4.3.

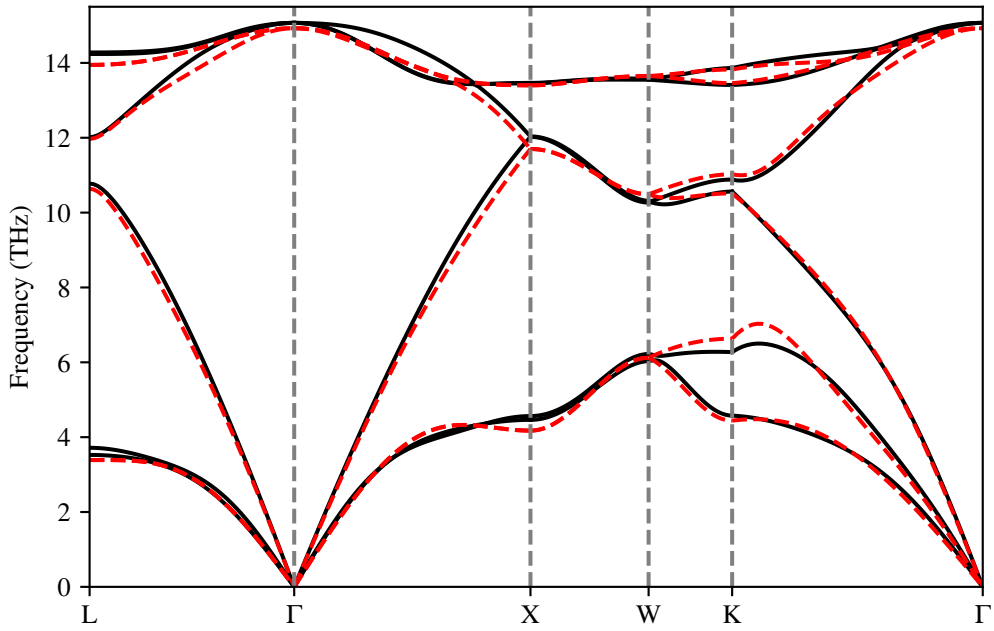


Figure 4.3.: The phonon band structure of cubic diamond silicon at zero pressure. The solid black line is computed using DFT while the red dashed line is obtained using the MLFF.

The reference DFT data are computed using tighter settings. As plane wave cutoff 500 eV are chosen. The orbitals are optimized using the Kosugi algorithm which is a special blocked-Davidson iteration scheme. Fermi smearing is used with a smearing width of 0.01 eV. The electronic degrees

of freedom are relaxed until the change in energy is smaller than 1×10^{-8} eV.

The phonon band structure computed by the MLFF is very close to the reference one computed by DFT. The largest discrepancies are for the optical mode between the Γ and X point where two distinct bands merge, and the acoustic mode near the K point.

5. Computing the thermal conductivity using a MLFF and a NNP

In this chapter, we want to further explore the capabilities of MLFF in general. For this purpose we determine the thermal conductivity of silicon using a Non Equilibrium Molecular Dynamics (NEMD) method in conjunction with a MLFF. Such simulations are virtually impossible to perform using FP methods, because long MD simulations with a large number of atoms are needed. This would require immense computational resources which are not available as of the writing of this theses. Using MLFF we are able to perform multiple MD simulations up to 1 ns long and with up to 32 768 atoms. The NEMD method used to calculate the thermal conductivity is a modified version of the Müller-Plathe algorithm [27, 28], which we implemented into VASP. Additionally, we train a Neural Network Potential (NNP) and use it to calculate the thermal conductivity for comparison. The Neural Network Potential (NNP) package used here is *n2p2* [29, 30, 31, 32] which is integrated with the MD software LAMMPS [33, 34].

5.1. NEMD method

On the macroscopic level the thermal conductivity κ is defined from the heat flux vector $\mathbf{J}$ and the temperature gradient ∇T as Fourier's law:

$$\mathbf{J} = -\kappa \nabla T. \tag{5.1}$$

The heat flux vector describes the energy transfer over time through an area normal to the direction of the flux. In Eq (5.1) the thermal conductivity is a 3×3 tensor. When dealing with an isotropic system the thermal conductivity can be written as scalar when $\mathbf{J}$ and the temperature gradient point in the same direction. In this case we chose the z direction. Now the thermal conductivity can be defined on the microscopic scale as

$$\kappa = \lim_{\substack{\partial T/\partial z \to 0 \\ t \to \infty}} -\frac{\langle J_z(t) \rangle}{\langle \partial T/\partial z \rangle}, \tag{5.2}$$

where $\langle \cdot \rangle$ indicates time averages. In the approach of Müller-Plathe, a heat flux is imposed in the system and the resulting temperature gradient is computed. For this the simulation box is split into N_{slab} slabs along the z direction. These slabs are equal in size (thickness and volume). The kinetic temperature of slab k is computed as

$$T_k = \frac{1}{3 n_k k_B} \sum_{i \in k}^{n_k} m_i v_i^2, \tag{5.3}$$

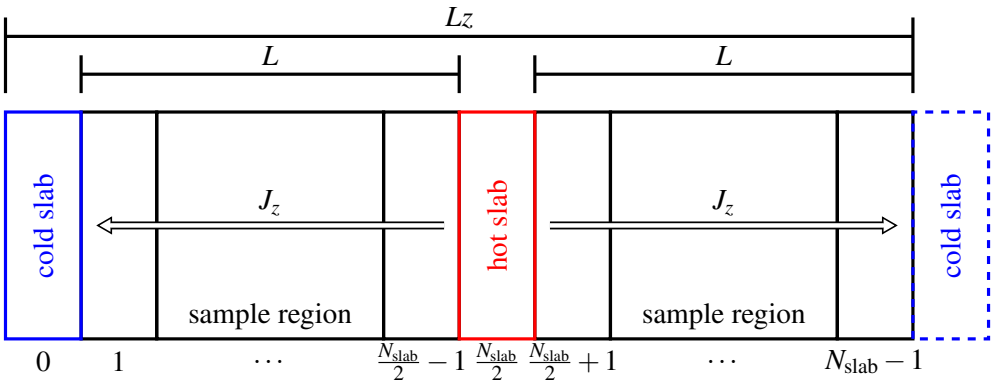


Figure 5.1.: The periodic simulation cell is shown in this sketch. It comprises in total of N_{slab} slabs and has a length Lz along the z direction. The first slab is the 'cold' one and slab $N_{\text{slab}}/2$ is the 'hot' slab. A heat flux J_z is imposed between those two slabs by momentum and energy exchange between the 'hot' and the 'cold' regions. The regions bordered by the 'hot' and the 'cold' slab are the 'two sample regions' and are of length L.

with k_B denoting Boltzmann's constant and the sum extending over all n_k atoms in slab k with their individual masses m_i and velocities v_i. These individual temperatures are used to compute the time averaged heat gradient $\langle \partial T/\partial z \rangle$. We define a 'cold' slab at position $k = 0$ and a 'hot' one at $k = N_{\text{slab}}/2$ (see Figure 5.1).

The heat flux is generated by a momentum and energy exchange between the 'hot'and 'cold' slab; after an interval of W MD steps a virtual elastic collision between the atom possessing the largest kinetic energy in the 'cold' slab ant the atom with the smallest kinetic energy in the 'hot' slab takes place. For the atom in the 'cold' slab the new velocity is expressed as

$$\mathbf{v}_c^{\text{new}} = 2\frac{m_c\mathbf{v}_c^{\text{old}} + m_h\mathbf{v}_h^{\text{old}}}{m_c + m_h} - \mathbf{v}_c^{\text{old}}, \tag{5.4}$$

and for the atom in the 'hot' slab

$$\mathbf{v}_h^{\text{new}} = 2\frac{m_c\mathbf{v}_c^{\text{old}} + m_h\mathbf{v}_h^{\text{old}}}{m_c + m_h} - \mathbf{v}_h^{\text{old}}, \tag{5.5}$$

where the subscripts c and h denotes the 'cold' and the 'hot' slab respectively. The energy transferred between the hot and the cold region is used to compute the heat flux

$$\langle J_z(t) \rangle = -\frac{1}{2At} \sum_{\text{transfers}(t)} \frac{1}{2} m_h \left((\mathbf{v}_h^{\text{new}})^2 - (\mathbf{v}_h^{\text{old}})^2 \right), \tag{5.6}$$

where A is the area of the simulation box perpendicular to the z direction and t is the simulation time. The factor $1/2$ is due to the periodic boundary conditions where energy is transferred on both the left and right side of the 'hot' region to the 'cold' one. Once a steady regime is reached the thermal conductivity can be computed by combining Eq (5.2) and (5.6)

$$\kappa = \lim_{\substack{\partial T/\partial z \to 0 \\ t \to \infty}} \frac{1}{2At \langle \partial T/\partial z \rangle} \sum_{\text{transfers}(t)} \frac{1}{2} m_h \left((\mathbf{v}_h^{\text{new}})^2 - (\mathbf{v}_h^{\text{old}})^2 \right), \tag{5.7}$$

where the heat gradient $\langle \partial T/\partial z \rangle$ is computed from the resulting temperature profile of the different slabs T_k. The limit $\partial T/\partial z \to 0$ implies that a linear response regime needs to be achieved in the temperature profile in order to apply classical thermodynamics. The limit $t \to \infty$ needs to be truncated and the time averages are computed when a steady regime is reached. Moreover, to converge the calculations with respect to the supercell size, multiple calculations with different length of the simulation box have to be performed. To obtain the final result, the thermal conductivity is extrapolated to $L \to \infty$

$$\kappa_{\text{macro}} = \lim_{L\to\infty} \kappa(L). \tag{5.8}$$

This method is simple and easy to implement. In appendix C a code for implementing the Müller-Plathe algorithm is shown. During this NEMD method the total energy is not conserved. Even though with this NEMD method the total energy should be conserved, higher-order truncation terms in the Verlet time integration are the cause of this energy drift [35]. The numerical stability of the Verlet integrator is due to its time reversibility. Since this system is not in equilibrium no time reversibility can relied upon. The drift in total energy per atom during a long NEMD is shown in Figure 5.2. When doubling Δt we were able to observe an increase of approximately a factor of eight in the total energy shift. This hints at scaling properties of $\mathcal{O}(\Delta t^3)$.

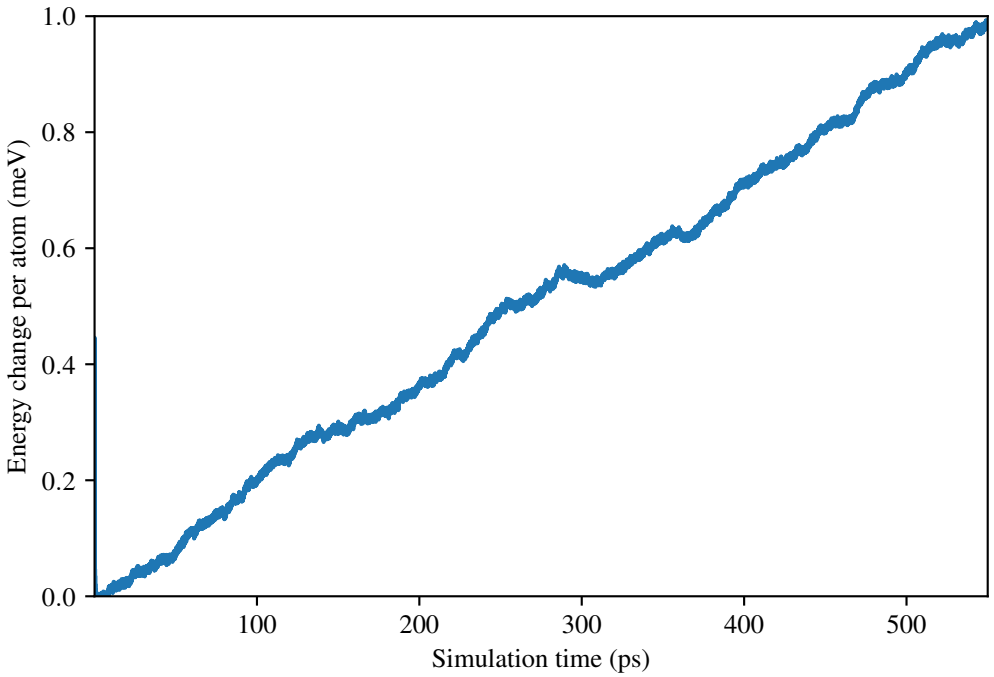


Figure 5.2.: The drift in total energy per atom during a NEMD simulation with $\Delta t = 2\,\text{fs}$ is shown. The corresponding stimulation box has 32 768 atoms.

5.2. Size dependency of the thermal conductivity: Regression model

There are two phenomena that contribute to the system-size dependency of the thermal conductivity when using a NEMD method. The first is phonon-phonon scattering. For a too small supercell too few phonon modes are available and thus the scattering can not be reproduced accurately. The second phenomenon is the likelihood of ballistic effects. Assume a phonon emitted from the 'hot' slab with a mean free-path Λ which is larger than L. For such a phonon it is possible to travel across the sample without scattering. This is called ballistic transport. Phonons traveling ballistically contribute less to the thermal conductivity. When L is increased the possibility of phonons with a larger mean free-path scattering before passing through the sample region is higher [36]. On a macroscopic scale it is very unlikely that phonons pass from the hot reservoir to the cold reservoir without scattering.

For modeling the L dependency of the thermal conductivity along the z direction, we start by solving the Boltzmann transport equation under the relaxation-time approximation. By using the Fourier law and converting the summation over all phonon modes to an integral over the first Brillouin zone one obtains[37]

$$\kappa_z(L) = \frac{V}{8\pi^3} \sum_{\nu} \int c_{\mathrm{ph}} v_{g,z}^2(\mathbf{k},\nu)\tau(\mathbf{k},\nu,L)\mathrm{d}\mathbf{k}, \tag{5.9}$$

where V is the sample volume. The sum is over the phonon branches ν, c_{ph} is the volumetric-specific heat of each mode and is defined as k_{B}/V in the classical approximation, and the group velocity in the z direction is written as $v_{g,z}(\mathbf{k},\nu)$. The relaxation time $\tau(\mathbf{k},\nu,L)$ is size dependent. It is defined by the average time between successive scattering events and can be written as

$$\tau(\mathbf{k},\nu,L) = \frac{\Lambda(\mathbf{k},\nu,L)}{|\mathbf{v}_g(\mathbf{k},\nu)|}, \tag{5.10}$$

with $\Lambda(\mathbf{k},\nu,L)$ the free-mean path of a phonon of mode $(\mathbf{k},\nu)$ in a simulation cell with a sample region length L. The group-velocity vector is denoted by $\mathbf{v}_g(\mathbf{k},\nu)$. We assume that the phonon-phonon and boundary scattering are independent from each other. Therefore according to Matthiesen's rule $\tau(\mathbf{k},\nu,L)$ can be written as a combination of the relaxation times for the different scattering mechanisms

$$\frac{1}{\tau(\mathbf{k},\nu,L)} = \frac{1}{\tau_{pp}(\mathbf{k},\nu)} + \frac{1}{\tau_b(\mathbf{k},\nu,L)}, \tag{5.11}$$

where $\tau_{pp}(\mathbf{k},\nu)$ and $\tau_b(\mathbf{k},\nu,L)$ are the phonon-phonon scattering and boundary scattering relaxation times respectively. This is under the assumption of an ideal crystal (e.g. without any defects). The relaxation time for boundary scattering can be written as the average time between boundary scattering events

$$\tau_b(\mathbf{k},\nu,L) = \frac{L}{2|v_{g,z}(\mathbf{k},\nu)|}. \tag{5.12}$$

Using this we can rewrite Eq (5.9) to explicitly show the length dependency of the thermal conductivity

$$\kappa_z(L) = \frac{k_{\mathrm{B}}}{8\pi^3} \sum_{\nu} \int v_{g,z}^2(\mathbf{k},\nu)\tau_{pp}(\mathbf{k},\nu) \left(1 + \frac{2|v_{g,z}(\mathbf{k},\nu)|\tau_{pp}(\mathbf{k},\nu)}{L}\right)^{-1} \mathrm{d}\mathbf{k}.$$

When modelling the length dependency of the thermal conductivity in Eq (5.2) one can therefore assume that the inverse of the thermal conductivity behaves according to a function of the inverse sample length

$$\frac{1}{\kappa_z} = \chi\left(\frac{1}{L}\right). \tag{5.13}$$

We are interested in the limit $1/L \to 0$ in order to obtain $1/\kappa_{\text{macro}}$. We therefore approximate the unknown function χ at 0 using a Taylor-series expansion around $1/L$

$$\frac{1}{\kappa_{\text{macro}}} = \chi(0) = \chi\left(\frac{1}{L}\right) - \chi'\left(\frac{1}{L}\right)\frac{1}{L} + \frac{1}{2}\chi''\left(\frac{1}{L}\right)\left[\frac{1}{L}\right]^2 + \mathcal{O}\left(\left[-\frac{1}{L}\right]^3\right). \tag{5.14}$$

Since χ is unknown, the results of the NEMD for different L can be used to obtain the derivatives via a polynomial fit. This is done by fitting $1/\kappa$ versus $1/L$. Normally, however, the series expansion is truncated after the first-order term because higher-order terms are difficult to obtain using a polynomial fit due to the uncertainty in NEMD predictions and the computational cost of the additional data points needed.

In Reference [36] it is argued that the maximal thermal conductivity that can be estimated within 10% of the true value using a linear approximation with a certain minimal sample length L_{min} is approximated by

$$\kappa_{\text{max}} = \frac{k_{\text{B}}(\sqrt{C_{11}} + 2\sqrt{C_{44}})}{18a^3\sqrt{\rho}} L_{\text{min}}, \tag{5.15}$$

where a is the lattice constant, C_{11} and C_{44} are the elastic constants in longitudinal and transverse direction respectively, and ρ is the density of the material.

5.3. NNP: an overview

For comparison to the MLFF using the regression method Bayesian and SVD we use a NNP. An advantage of NNP is that they are computationally less expensive than MLFF, therefore we are able to go to even larger simulation box sizes. The speed of NNP is near to that of EFF. In contrast to the MLFF, which is a kernel based method, the NNP is based on artificial neural networks [31]. The neural networks are trained using FP data with the goal of reproducing the potential energy surface as accurately as possible. In this section we will give a brief overview of how a basic NNP works. Additional information can be found in [31, 30, 38].

5.3.1. The Neural Network

In order to predict the potential energy U from the atomic configuration a functional relation between these two observables is constructed using artificial neurons. The neurons are ordered in one input layer, one or multiple 'hidden' layers and one output layer with a single neuron. This single neuron represents the potential energy. The input neurons $\mathbf{G} = \{G_j\}$ contain information on the atomic configurations. A sketch of a neural network is shown in Figure 5.3. It is described by

the short notation $3-5-5-1$, where the individual numbers, starting on the left with the input layer, indicate the number of neurons per layer. Typical neural networks contain more neurons than shown here.

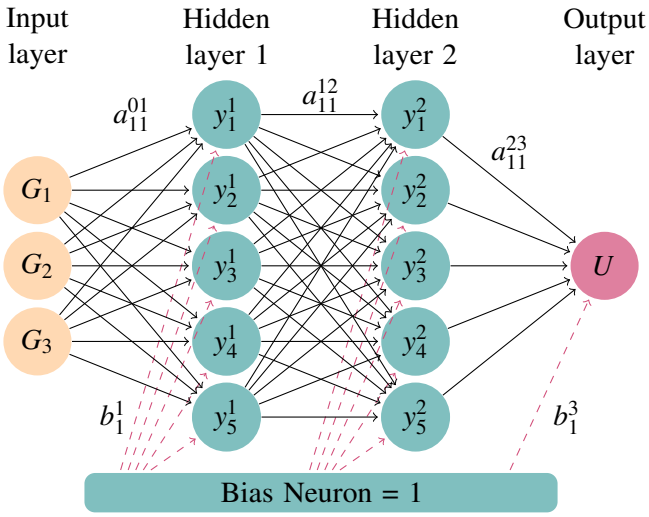


Figure 5.3.: Sketch of a small feed-forward neural network. It consists of two 'hidden' layers. This neural network has the goal to describe a functional relation between the atomic positions $\{G_i\}$ and the potential energy U [38].

Each neuron in each layer is connected to all the neurons of the adjacent layers via a weight parameter a_{ij}^{kl}, where the superscripts indicate the two connected layers and the subscripts indicate the two connected neurons. In detail, the neuron i located in layer k is connected to the neuron j located in layer l by the weight parameter a_{ij}^{kl}. The superscript 0 defines the input layer. In this neural network the information only flows in one direction, as indicated by the arrows. Additionally, there is a so-called bias neuron which provides a fixed value. This node is connected to all 'hidden' layers and to the output layer. The fixed value is scaled by a bias weight b_i^j and targeted at neuron i in layer j. Each value y_i^j of a neuron i in the layer j can be computed by a linear combination of the values of the previous layer shifted by the bias weight

$$y_i^j = f_i^j \left(b_i^j + \sum_{k=1}^{N_{j-1}} a_{k,i}^{j-1,j} y_k^{j-1} \right), \tag{5.16}$$

where N_{j-1} indicates the number of neurons in the previous layer i.e. N_j is the number of neurons in layer j. Since the topology of the potential energy surface that we want to model cannot be expressed as linear combinations, a nonlinear function f_i^j is used. This so-called 'activation function' needs to be differentiable in order to determine the analytic forces for each atom. Additionally the derivatives with respect to the weight parameters are used for the gradient based optimization of the weights, therefore when fitting forces also the second derivatives of the activation functions are needed. There are multiple options for such activation functions. Generally they converge to 1, 0, or -1 for very large and very small arguments. In this thesis we use a hyperbolic tangent function for the hidden layers and a linear function for the single output

neuron. For example the neural network shown in Figure 5.3 has the complete functional form

$$U = f_1^3 \left(b_1^3 + \sum_{k=1}^{5} a_{k1}^{23} \cdot f_k^2 \left(b_k^2 + \sum_{j=1}^{5} a_{jk}^{12} \cdot f_j^1 \left(b_j^1 + \sum_{i=1}^{3} a_{ij}^{01} \cdot G_i \right) \right) \right). \tag{5.17}$$

5.3.2. High-Dimensional NNP

The neural network need to be trained on a number of atomic structures and their corresponding potential energy surfaces in order to obtain accurate results. This can become very time-consuming for large training data sets and huge neural networks with many weight parameters. The total number of weight parameters N_w is calculated by

$$N_w = \sum_{k=1}^{M_{\mathrm{HL}}} (N_{k-1} + 1) N_k, \tag{5.18}$$

where M_{HL} indicates the number of hidden layers. Large neural networks with many weight parameters need a sufficiently diverse training set. This is challenging in two ways: selecting the right data and the required compute time. Additionally, too many weight parameters will make the fitting process very inefficient. Finally, a single neural network could only be used for the same system size for which it was trained. Every atom in a structure adds three degrees of freedom that need to be included in the form of additional input neurons. This requires even more weight parameters. Because of these limitations it is impractical to generate a High-Dimensional NNP using a single feed-forward neural network.

A clever workaround is to use multiple neural networks, one for each atom in the system. This is similar to the regression method discussed before in Chapter 2. The energy U of this system is calculated as a sum over all individual atomic energy contributions U_i which are calculated by neural networks

$$E = \sum_{i=1}^{N_{\mathrm{atom}}} E_i, \tag{5.19}$$

where N_{atom} is the total number of atoms in the system. The atomic energy contributions are computed via symmetry functions $\mathbf{G}$ which contain information about the local environment around the atom of interest up to a certain cutoff R_{cut}. The symmetry functions of an atom are fed into an atomic neural network to obtain its atomic energy contribution. If a system contains multiple atom types one would need as many individual atomic neural networks as atom types. Finally, the total energy of the system is computed by summing over all atomic energy contributions as written in Eq. (5.19). A schematic representation of such a high-dimensional NNP for silicon is shown in Figure 5.4.

5.3.3. Symmetry Functions, Cutoff Function, and Forces

The symmetry functions are similar to the descriptors used for the MLFF, i.e. they are rotationally and translationally invariant and capture the radial and angular distribution of other atoms around

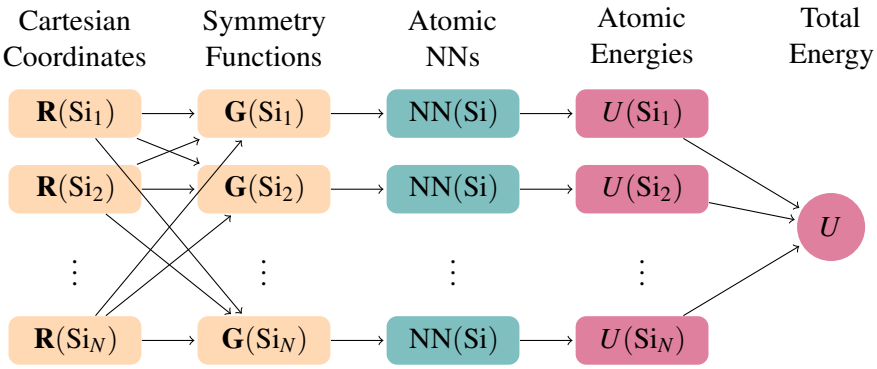


Figure 5.4.: Sketch of a high-dimensional NNP for a system with N Silicon molecules. The Cartesian coordinates of each atom are represented with the vector $\mathbf{R}$. The coordinates are transformed to symmetry functions stored in the vectors $\mathbf{G}$. These symmetry functions describe the local environment around a atom. The vectors $\mathbf{G}$ are used as input for the atomic neural networks which provides the individual atomic energies. The atomic energies are summed to obtain the total energy U [38].

each atom. Generally one could use a wide range of different symmetry functions up to a certain cutoff radius.

To keep things short we will only focus on those which we used for the present work. The first function type we use describes the radial arrangement of atoms within the cutoff radius around the atom i. This is the so-called radial symmetry function by Behler and Parrinello, written as [10]

$$G_i^2 = \sum_{j \neq i} \exp\left(-\eta \left(r_{ij} - r_s\right)^2\right) f_{\text{cut}}(r_{ij}). \tag{5.20}$$

The parameter η defines the width and r_s the shift of the Gaussian. One can use multiple radial symmetry function with different parameters to probe different radii. We will define the cutoff function f_{cut} later.

The second symmetry function we use is the angular one, written as [10]

$$G_i^3 = 2^{1-\zeta} \sum_{j \neq i} \sum_{k \neq i,j} \Big[\left(1 + \lambda \cos(\theta_{ijk})\right)^\zeta \exp\left(-\eta \left(\left(r_{ij} - r_s\right)^2 + \left(r_{ik} - r_s\right)^2 + \left(r_{jk} - r_s\right)^2 \right)\right) f_{\text{cut}}(r_{ij}) f_{\text{cut}}(r_{ik}) f_{\text{cut}}(r_{jk}) \Big], \tag{5.21}$$

where θ_{ijk} is the angle between the atomic distances r_{ij} and r_{ik}. It depends on two additional parameters. The exponent ζ is used to probe the angular distribution while the parameter λ is used to invert the shape of the cosine, therefore it can have the values 1 or -1. The third and final symmetry function we use is an additional angular symmetry function described by Behler [39]. It has the form

$$G_i^9 = 2^{1-\zeta} \sum_{j \neq i} \sum_{k \neq i,j} \Big[\left(1 + \lambda \cos(\theta_{ijk})\right)^\zeta \exp\left(-\eta \left(\left(r_{ij} - r_s\right)^2 + \left(r_{ik} - r_s\right)^2 \right)\right) f_{\text{cut}}(r_{ij}) f_{\text{cut}}(r_{ik}) \Big], \tag{5.22}$$

having the same parameters as the previous one.

As for the symmetry functions there is also a wide variety of cutoff functions. In this thesis we use a polynomial of fifth order [30]

$$f_{\text{cut}}(c) = \begin{cases} x^3\left(x\left(15-6x\right)-10\right)+1 & \text{if } r < r_c \\ 0 & \text{else} \end{cases}, \tag{5.23}$$

with $x = r/r_c$. Using polynomials has the advantage of being faster to calculate compared to the cutoff function defined in Eq. (2.4) because they do not contain any transcendental functions.

For performing MD simulations it is crucial to calculate forces. The neural network and the symmetry functions are well defined, therefore it is possible to optain the forces via the derivatives of the neural network with respect to the atomic coordinates. The force for a specific atomic coordinate α is defined as

$$F_\alpha = -\frac{\partial U}{\partial \alpha} = -\sum_{j=1}^{N_{\text{atom}}} \sum_{\mu=1}^{N_{\text{sym},j}} \frac{\partial U_j}{\partial G_{j\mu}} \frac{\partial G_{j\mu}}{\partial \alpha}, \tag{5.24}$$

where $N_{\text{sym},j}$ denotes the number of symmetry functions of atom j.

5.3.4. Training the NNP

The individually constructed symmetry functions can have a widely different range of values. This could unintentionally weight some symmetry functions more strongly. Therefore, before fitting a neural network the input data need to be preconditioned. In order to accomplish this one scales the symmetry functions according to the scaling function [31]

$$G_j^{\text{scaled}} = S_{\text{min}} + \left(S_{\text{max}} - S_{\text{min}}\right)\frac{G_j - \langle G_j \rangle}{G_{j,\text{max}} - G_{j,\text{min}}}. \tag{5.25}$$

The parameter S_{min} defines the desired minimum scaled symmetry function value and S_{max} defines the desired maximum scaled symmetry function value. In this work S_{min} was set to 0 and S_{max} to 1. The mean of the symmetry function j is written as $\langle G_j \rangle$, while $G_{j,\text{max}}$ and $G_{j,\text{min}}$ indicate the maximum and minimum respectively.

Not only the input of the neural network needs to be normalized, also the output. The energy of each configuration j is scaled according to

$$U_j^{\text{scaled}} = \frac{U_j - N_{j,\text{atom}} \langle u \rangle}{\sigma_u}, \tag{5.26}$$

where the mean energy per atom is defined as

$$\langle u \rangle = \frac{1}{N_{\text{data}}} \sum_{j=1}^{N_{\text{data}}} \frac{U_j}{N_{j,\text{atom}}}, \tag{5.27}$$

and the standard deviation of the energy per atom is calculated by

$$\sigma_u = \sqrt{\frac{1}{N_{\text{data}} - 1} \sum_{j=1}^{N_{\text{data}}} \left(\frac{U_j}{N_{j,\text{atom}}} - \langle u \rangle\right)^2}, \tag{5.28}$$

and N_{data} is the total number of training data sets. Since the NNP is also trained using forces, these need to be scaled as well. This is done by

$$F_{\alpha}^{\text{scaled}} = \frac{F_{\alpha}}{\sigma_F}. \tag{5.29}$$

The standard deviation of the forces is calculated by

$$\sigma_F = \sqrt{\frac{\sum_{j=1}^{N_{\text{data}}} \sum_{i=1}^{N_{j,\text{atom}}} \vec{F}_{ji}^{\,2}}{3\sum_{j=1}^{N_{\text{data}}} N_{j,\text{atom}} - 1}}, \tag{5.30}$$

where $\vec{F}_{ji}^{\,2}$ denotes the force vector of the ith atom in the jth data set.

The weight parameters a and the bias weights b are optimized by the extended Kalman filter method [40, 41, 42]. This method minimizes the cost function, i.e. the root-mean-square error, by using gradients with respect to the weights. It is superior in performance to the standard backpropagation algorithm, which is often used for fitting neural networks. Additionally, this method takes the noise in the training data into account. Noise in the training dataset can occur especially due to the truncated symmetry functions. The cost function which is minimized is written as

$$\Gamma = \sum_{j=1}^{N_{\text{data}}} \left(U_j^{\text{ref}} - U_j^{\text{NN}}\right)^2 + \beta^2 \sum_{j=1}^{N_{\text{data}}} \sum_{i=1}^{N_{\text{atom}}} \left(\vec{F}_{ji}^{\,\text{ref}} - \vec{F}_{ji}^{\,\text{NN}}\right)^2. \tag{5.31}$$

The superscript 'NN' indicates that the observable was calculated via the NNP, while the superscript 'ref' denotes energies or forces from the training data set. The parameter β is introduced to control the relative importance between forces and energies. A more detailed description of the fitting process can be found in [31].

5.4. Computational details

The thermal conductivity was calculated for a total of five simulation boxes with different lengths in the z directions. This allows to extrapolate the value to an infinite cell size as discussed in section 5.2. The different lengths are shown in Table 5.1. Because the NNP is faster in terms of compute time, we only calculate the largest simulation box using the NNP. We also perform significantly more MD steps when using NNP. Momentum and energy between one pair of atoms, one in the 'cold' slab and another in the 'hot' slab, are exchange every $W = 25$ MD steps for the MLFF. For the NNP we exchanged every $W = 50$ MD steps. We use for both ML methods time steps of $\Delta t = 2\,\text{fs}$, therefore, momenta are exchanged every 50 fs or 100 fs respectively. An NVE microcanonical ensemble is used. The initial velocities are set according to the Maxwell-Boltzmann distribution at 2000 K for the MLFF. This results in a temperature of about 975 K during the MD simulation (Half the kinetic energy is converted to potential energy). For the NNP we use a thermostat to equilibrate the initial atomic velocities at 1000 K using a NVT ensemble. The number of slabs N_{slabs} in each simulation box is chosen in a way that every slab contains four layers of atoms along the direction of the heat flux (one lattice parameter thick). The cross-section of each slab is a 4×4 conventional unit cell. In order to observe the convergence of the thermal

profile and the heat flux, their averages are collected every 5000 MD steps. Once a steady regime is reached, we compute the averages to obtain the final thermal profile and the heat flux. To obtain the temperature gradient, a least squares fit is performed on the linear parts of the thermal profile. Note that the different simulation box sizes need different length of time to reach a steady regime: the longer the box the more simulation time is needed. As an example, Figure 5.5 shows the convergence of the heat gradient in all simulation boxes computed using the MLFF. In this case, every system reaches its steady state after at least 200 ps.

Table 5.1.: The four different simulation boxes are summarized in this table. Their lengths Lz, number of atoms N_a, number of slabs N_{slab}, and total simulation time t for both the MLFF and NNP are shown.

Lz	N_a	N_{slab}	t_{MLFF}	t_{NNP}
48.9 nm	11520	90	1.00 ns	4.00 ns
74.0 nm	17408	136	0.55 ns	4.00 ns
97.9 nm	23040	180	0.45 ns	4.00 ns
139.2 nm	32768	256	0.55 ns	4.00 ns
278.3 nm	65536	512	/	2.44 ns

The MLFF trained in Chapter 3 contains a lot of LRC which increases the computational cost. Here, we use a MLFF which contains only information on the phase of interest and therefore needs fewer LRC. In our case, the desired phase is CD. The FP calculations are performed using the PBEsol functional which describes the ground-state of silicon with better accuracy than PBE. The MLFF was trained in an NpT ensemble for 30 000 time steps at 500 K followed up by additional 30 000 time steps at 1000 K. Time steps of 3 fs are used during training. FP calculations were performed with a plane wave cutoff of 325 eV. The atomic distribution of the angular descriptors is broadened by Gaussian functions with a width of $\sigma_{\text{atom}} = 0.3$ Å. The number of radial LRC used for expanding the atomic distribution of the angular descriptor is fixed to 24. A cutoff of 6 Å is used for the angular descriptors. We set $L_{\text{max}} = 4$ to be the maximum angular momentum quantum number. For fitting, the forces are weighed ten times stronger. The final MLFF consists of 1350 LRC and 1142 RSDS with 216 atoms each.

For training the NNP we use the same training data set as for the MLFF supplemented with additional configurations up to 1500 K resulting in a total of 3138 data points for energy and 2 058 384 data points for forces. Approximately 10 % are used as test data. We use a $62-25-25-1$ neural network with a total of 2200 weights and 51 biases resulting in 2251 connections. We use a hyperbolic tangent activation function for the hidden layers and a linear activation function for the output layer. A total of 62 symmetry functions are used of which 14 are of type 2, 24 are of type 3, and 24 are of type 9. A detailed list of symmetry functions and the corresponding parameters is shown in Appendix E. 30 training epochs are performed to obtain the final NNP resulting in a root-mean-square error of approximately 0.52 meV for both the test and training energy and of approximately 50 meV Å^{-1} for both the test and training forces.

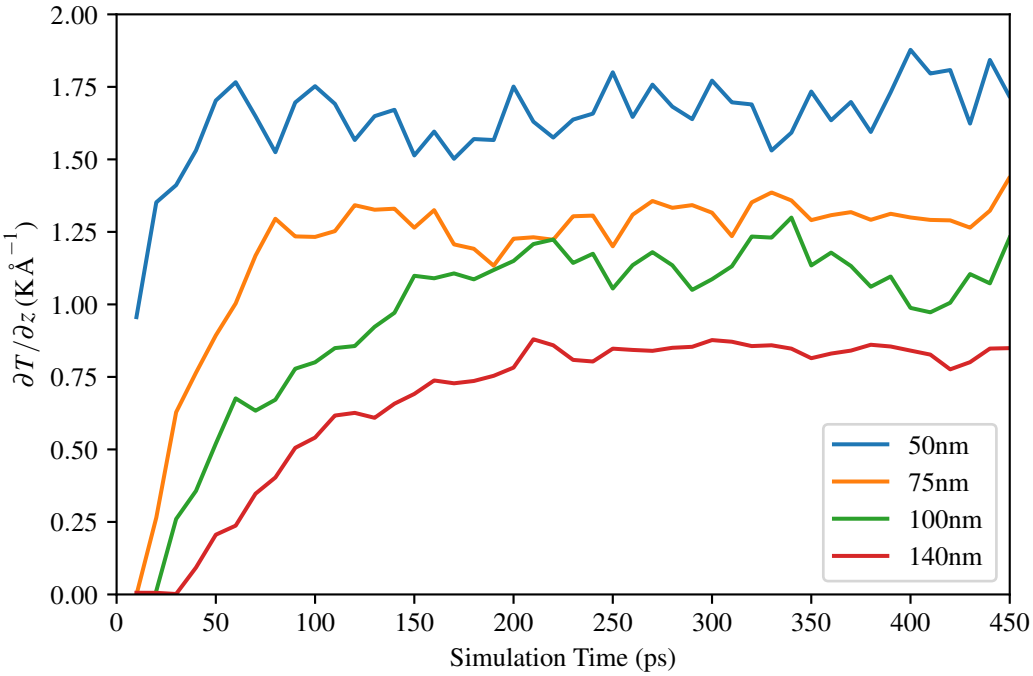


Figure 5.5.: The convergence of the heat gradient $\partial T/\partial z$ for the simulation boxes which are computed using the MLFF are shown here. Each point is an average over 5000 MD time steps.

5.5. Results

Table 5.2 summarizes the heat flux computed after a steady regime is reached for each individual cell size, as well as the mean temperature gradient left and right of the hot region. For each individual cell the thermal conductivities are computed after averaging the temperature gradient for the left and the right side. The inverse thermal conductivities as a function of the inverse sample length (see also Table 5.2) are shown in Figure 5.6. The macroscopic thermal conductivity is obtained by fitting the truncated function described in Eq (5.14) to κ^{-1} vs L^{-1}. Since we can clearly observe a non-linear behavior, a second order polynomial fit is performed in addition to the linear fit. The resulting thermal conductivities are shown alongside some reference data in Table 5.3. The thermal profiles of each simulation cell are shown in Appendix B in Figures B.2a, B.2b, B.2c, and B.2d for the MLFF. Thermal profiles computed using the NNP are shown in Figures B.3a, B.3b, B.3c, B.3d, and B.3e.

The discrepancy in the heat flux and temperature gradient between the MLFF and the NNP results is due to different momenta exchange frequencies. Since we exchange the momenta twice as often for MLFF than for the NNP, it is not surprising, that the heat fluxes and temperature gradients derived via the MLFF are twice the ones computed using the NNP. The resulting individual thermal conductivities are found to be similar with the two methods, but a small systematic shift can be

Table 5.2.: The sample length L, the heat flux J_z, the mean temperature gradient for the left and right $(\partial T/\partial z)$ side, the thermal conductivity κ and the mean temperature over the fitting area $\langle T \rangle$.

MLFF					
Lz(nm)	L(nm)	J_Z(W m^{-1} Å^{-1})	$(\partial T/\partial z)$(mK Å^{-1})	κ (W m^{-1} K^{-1})	$\langle T \rangle$(K)
48.9	23.9	13.0 ± 0.3	1703.4 ± 0.3	7.6 ± 0.2	986
74.0	36.4	12.0 ± 0.3	1308.5 ± 0.1	9.2 ± 0.2	954
97.9	48.4	11.4 ± 0.3	1130.5 ± 0.1	10.1 ± 0.3	987
139.2	69.0	10.6 ± 0.2	842.3 ± 0.1	12.6 ± 0.3	931
NNP					
Lz(nm)	L(nm)	J_Z(W m^{-1} Å^{-1})	$(\partial T/\partial z)$(mK Å^{-1})	κ (W m^{-1} K^{-1})	$\langle T \rangle$(K)
48.9	23.9	7.6 ± 0.1	1038.5 ± 0.2	7.3 ± 0.1	1002
74.0	36.4	7.3 ± 0.1	853.6 ± 0.1	8.5 ± 0.1	981
97.9	48.4	7.0 ± 0.1	736.8 ± 0.1	9.6 ± 0.1	982
139.2	69.0	6.7 ± 0.1	606.6 ± 0.1	11.0 ± 0.1	893
278.3	138.7	5.8 ± 0.1	413.3 ± 0.1	14.0 ± 0.1	903

observed. This could have two origins. First, there is a difference between the two ML methods used, and second, the momenta exchange frequencies were different. Indeed one can observe small differences in the thermal conductivity when changing the momenta exchange frequency [27]. Nevertheless the shift may also be due to the difference in the ML methods or the training data. In Table 5.2 one sees that the mean temperature over the fitting area $\langle T \rangle$ decreases with the cell length. This is probably due to the increasing maximal temperature of the temperature profile with increasing cell length, therefore the linear regime ends at lower temperatures (see Appendix B).

Table 5.3.: The final thermal conductivity at approximately 1000 K is shown below. The values from the linear extrapolation and for the second order polynomial are reported in addition to the thermal conductivities of this work. Two reference values are also shown. 'MLFF Shift' is determined by refitting the NNP polynomial fit to the MLFF data and not changing the curvature.

Method	κ(W m^{-1} K^{-1})
MLFF Linear	22 ± 4
MLFF Polynomial	34 ± 15
NNP Linear	18.5 ± 0.3
NNP Polynomial	20.4 ± 0.8
MLFF Shift	25 ± 1
Tersoff [43]	59 ± 4
Experimental[44]	32.5

When taking a closer look at Figure 5.6 one can observe the previously mentioned shift more clearly. The linear fits are nearly parallel. The polynomial fit is very accurate for the points calculated by the NNP. This is a strong hint for non-linear behavior. Also the points calculated via the MLFF exhibit non-linear behavior, but the data are more noisy, and hence potentially inaccurate

compared to the NNP. Since the MLFF is computationally more expensive we performed fewer time steps, resulting in higher noise.

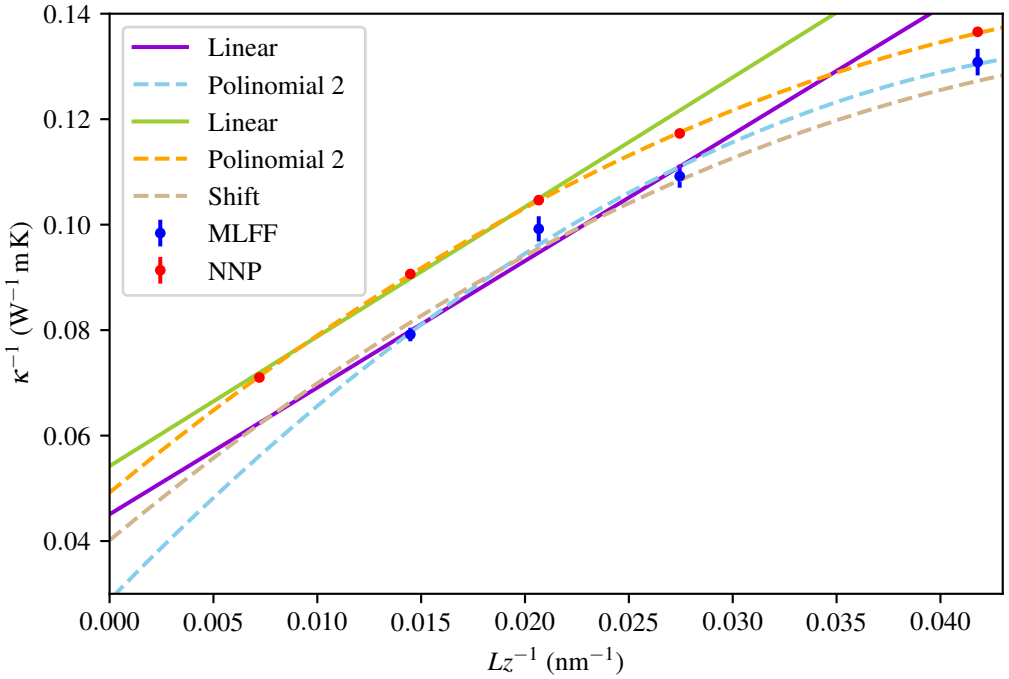


Figure 5.6.: The L^{-1} dependency of κ^{-1} is shown here. The red and blue dots are the inverse thermal conductivities obtained by the NEMD method with error bars for the MLFF and the NNP respectively. The purple and green lines are linear fits including only the largest three cells while the dashed blue and orange lines are polynomial fits of second order including all data points. The dashed tan line is the NNP polynomial fit shifted to the MLFF data.

With the exception of the polynomial fit performed over the points generated by the MLFF, we underestimate the thermal conductivity by about 10 W m^{-1} K^{-1} (30 %) compared to experimental results. Since the error of the polynomial fit for the MLFF data is large, this result should be considered with caution. Within the error bars, the MLFF and NNP clearly show the same behavior in dependence of cell size with an offset of about 5 mW^{-1} m K to 11 mW^{-1} m K. The best estimate from the MLFF using a polynomial fit is hence closer to 25 W m^{-1} K^{-1}, the NNP estimate minus 9 mW m K. We obtain this value by fitting a quadratic polynomial with fixed curvature to the MLFF data. This fit is shown in Figure 5.6 as 'Shift'. The curvature is the one determined by the fit to the NNP data. Using a EFF such as the Tersoff potential, instead, overestimates the thermal conductivity.

The strong non-linear behavior could indicate that our cell sizes are too small in order to predict the thermal conductivity accurately. Eq (5.15) can be used to approximate the largest thermal conductivity which can be extrapolated using a linear function. By using the properties in Table

5.4 calculated for this specific MLFF for silicon one obtains

$$\kappa_{\max} \approx 5\,\mathrm{W\,m^{-1}\,K^{-1}}.$$

This is four times lower than the thermal conductivity we obtain with the linear regression. When

Table 5.4.: The lattice constant, density, and elastic constants for silicon using the MLFF.

Property	MLFF silicon
a	5.438 Å
ρ	$2320\,\mathrm{kg\,m^{-3}}$
C_{11}	1497 kbar
C_{44}	998 kbar

inverting eq (5.15) one obtains a minimal sample length of

$$L_{\min} \approx 347\,\mathrm{nm}$$

for an assumed thermal conductivity of $35\,\mathrm{W\,m^{-1}\,K}$. This is five times lager than our largest cell confirming that our simulation boxes are too small in order to obtain more accurate results from linear fitting.

At this point we should emphasize that the non-linear regression model described in Section 5.2 only takes into account phonon-phonon scattering and ballistic effects. This is probably a too simplified model and more effects contribute to the non-linear behavior and in general to the discrepancy between our result and experiments.

We find that NNP are much faster than the present implementation of the kernel based MLFF in VASP, but require more training data to function properly. The MLFF on the other hand, is more stable: while the MLFF was stable during the NEMD runs up to 1500 K despite only including RSDS up to 1000 K, the NNP had some issues and performed nonphysically. This was fixed by expanding the training dataset.

6. Conclusion

In this thesis we have shown that generating a general purpose MLFF is possible, but extremely hard. The major limitations come from the training data. For example, the multi purpose MLFF generated in this work is very imprecise in predicting phases that are not included in the training data. More generally speaking, a MLFF is bad at extrapolating. In order to generate a true and reliable general purpose MLFF one would need enough training data and basis functions so that the MLFF is always interpolating. Since the compute time of the MLFF is linearly dependent on the number of LRC, it is more efficient to train a MLFF for a specific purpose like we did for the thermal conductivity calculations. Such specific MLFF needs fewer LRC and therefore is less computationally expensive. In addition to the reduction in compute time, such a MLFF should be more accurate, since adding vastly different configurations to the training generally worsens the predictive capabilities. In general, MLFF are a more accurate alternative to EFF, especially over a wide range of different structural phases. Also EFF give very bad estimates of the thermal conductivity.

MD simulations are a good method for generating training data, especially when coupled with the on-the-fly learning scheme. When heating, a broader variety of different configurations is added. The only drawback we encountered during this thesis is that unstable structures like the high-pressure phases undergo a phase transition during the MD simulation. If one desires to train a structure outside of a stable regime, however, this problem can be avoided by computing slightly different configurations interdependently and adding them to the MLFF training data. This works as long as sufficient LRC are already available. The slightly different configurations can be generated by 'mirroring' the atomic displacements extracted from an MD simulation of a different structure with the same number of atoms. The MLFF can be further refined at a later point for improved accuracy.

Using nanocrystals for surface training may seem tempting, but due to the fact that there is not an equal amount of information on each surface type the convergence is hard to control. The relative surface area per type is given by the respective surface energy according to the Wulff construction, therefore the MLFF has no high accuracy for surfaces. Training individual surface sites with periodic boundary conditions could potentiality lead to better predictive capabilities of the MLFF.

The elastic constants computed for β-Sn are way off compared to the ones obtained by DFT. This is most likely due to insufficient k-point sampling. If one would like to have a more precise MLFF the k-point sampling has to be increased especially for the high-pressure structures.

ML methods are a powerful tool for calculating thermal conductivity properties. Such calculations require long simulations with thousands of atoms which are, as of the writing of this thesis,

impossible to perform using DFT, because of the sheer compute power needed. Unfortunately, we found that for a highly harmonic material such as bulk silicon, the lengths of the supercells required to linearly extrapolate the thermal conductivity to $L \to \infty$ are even larger than the ones we accessed in this work. We estimate that we would need a five times larger sample region length than our largest supercell. An additional error source could be the fact that our MLFF for calculating the thermal conductivity was only trained up to 1000 K but the temperatures around the 'hot' slab are larger, therefore an uncertainty caused by extrapolation could arise. Nevertheless, when using a fitting function of second polynomial order, we are able to predict a thermal conductivity close to the experimental value.

Bibliography

[1] V. L. Deringer, N. Bernstein, A. P. Bartók, M. J. Cliffe, R. N. Kerber, L. E. Marbella, C. P. Grey, S. R. Elliott, and G. Csányi, "Realistic atomistic structure of amorphous silicon from machine-learning-driven molecular dynamics," *The journal of physical chemistry letters*, vol. 9, no. 11, pp. 2879–2885, 2018.

[2] A. P. Bartók, J. Kermode, N. Bernstein, and G. Csányi, "Machine learning a general-purpose interatomic potential for silicon," *Physical Review X*, vol. 8, no. 4, p. 041048, 2018.

[3] R. Jinnouchi, F. Karsai, and G. Kresse, "On-the-fly machine learning force field generation: Application to melting points," *Physical Review B*, vol. 100, no. 1, p. 014105, 2019.

[4] R. Jinnouchi, J. Lahnsteiner, F. Karsai, G. Kresse, and M. Bokdam, "Phase transitions of hybrid perovskites simulated by machine-learning force fields trained on the fly with bayesian inference," *Physical review letters*, vol. 122, no. 22, p. 225701, 2019.

[5] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, "Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons," *Physical review letters*, vol. 104, no. 13, p. 136403, 2010.

[6] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[7] "The vasp manual." `https://www.vasp.at/wiki/index.php/The_VASP_Manual`. Online; accessed December 3,2020.

[8] A. P. Bartók, R. Kondor, and G. Csányi, "On representing chemical environments," *Phys. Rev. B*, vol. 87, p. 184115, May 2013.

[9] R. Jinnouchi, F. Karsai, C. Verdi, R. Asahi, and G. Kresse, "Descriptors representing two- and three-body atomic distributions and their effects on the accuracy of machine-learned inter-atomic potentials," *The Journal of chemical physics*, vol. 152, no. 23, p. 234102, 2020.

[10] J. Behler and M. Parrinello, "Generalized neural-network representation of high-dimensional potential-energy surfaces," *Physical review letters*, vol. 98, no. 14, p. 146401, 2007.

[11] F. Karsai, "File:mlff rad and ang descriptor.png." `https://www.vasp.at/wiki/index.php/File:MLFF_rad_and_ang_descriptor.png`. Online; accessed Feburay 26, 2021.

[12] J. Boyd, "Chebyshev and fourier spectral methods. dover publications inc," *New York*, 2000.

[13] V. L. Deringer and G. Csányi, "Machine learning based interatomic potential for amorphous carbon," *Physical Review B*, vol. 95, no. 9, p. 094203, 2017.

[14] R. Jinnouchi and R. Asahi, "Predicting catalytic activity of nanoparticles by a dft-aided machine-learning algorithm," *The journal of physical chemistry letters*, vol. 8, no. 17, pp. 4279–4283, 2017.

[15] S. Gull and J. Skilling, "Maximum entropy and bayesian methods," *J. Skilling*, 1989.

[16] M. W. Mahoney and P. Drineas, "Cur matrix decompositions for improved data analysis," *Proceedings of the National Academy of Sciences*, vol. 106, no. 3, pp. 697–702, 2009.

[17] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," in *Linear algebra*, pp. 134–151, Springer, 1971.

[18] dantopa (https://math.stackexchange.com/users/206581/dantopa), "How does the svd solve the least squares problem?." Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2173715 (version: 2020-01-15).

[19] P. C. Hansen, J. G. Nagy, and D. P. O'leary, *Deblurring images: matrices, spectra, and filtering*. SIAM, 2006.

[20] J. Crain, S. J. Clark, G. J. Ackland, M. C. Payne, V. Milman, P. D. Hatton, and B. J. Reid, "Theoretical study of high-density phases of covalent semiconductors. i. ab initio treatment," *Phys. Rev. B*, vol. 49, pp. 5329–5340, Feb 1994.

[21] M. Kaltak, J. Klimeš, and G. Kresse, "Cubic scaling algorithm for the random phase approximation: Self-interstitials and vacancies in si," *Physical Review B*, vol. 90, no. 5, p. 054115, 2014.

[22] M. P. Allen and D. J. Tildesley, *Computer simulation of liquids*. Oxford university press, 2017.

[23] X. Wu, D. Vanderbilt, and D. Hamann, "Systematic treatment of displacements, strains, and electric fields in density-functional perturbation theory," *Physical Review B*, vol. 72, no. 3, p. 035105, 2005.

[24] P. Pulay, "Convergence acceleration of iterative sequences. the case of scf iteration," *Chemical Physics Letters*, vol. 73, no. 2, pp. 393–398, 1980.

[25] G. Wulff, "Xxv. zur frage der geschwindigkeit des wachsthums und der auflösung der krystallflächen," *Zeitschrift für Kristallographie-Crystalline Materials*, vol. 34, no. 1-6, pp. 449–530, 1901.

[26] A. Togo and I. Tanaka, "First principles phonon calculations in materials science," *Scripta Materialia*, vol. 108, pp. 1–5, 2015.

[27] F. Müller-Plathe, "A simple nonequilibrium molecular dynamics method for calculating the thermal conductivity," *The Journal of chemical physics*, vol. 106, no. 14, pp. 6082–6085, 1997.

[28] C. Nieto-Draghi and J. B. Avalos, "Non-equilibrium momentum exchange algorithm for molecular dynamics simulation of heat flow in multicomponent systems," *Molecular Physics*, vol. 101, no. 14, pp. 2303–2307, 2003.

[29] A. Singraber, "n2p2 - the neural network potential package." `https://github.com/CompPhysVienna/n2p2`. Online; accessed April 6,2021.

[30] A. Singraber, J. Behler, and C. Dellago, "Library-based lammps implementation of high-dimensional neural network potentials," *Journal of chemical theory and computation*, vol. 15, no. 3, pp. 1827–1840, 2019.

[31] A. Singraber, T. Morawietz, J. Behler, and C. Dellago, "Parallel multistream training of high-dimensional neural network potentials," *Journal of chemical theory and computation*, vol. 15, no. 5, pp. 3075–3092, 2019.

[32] M. P. Bircher, A. Singraber, and C. Dellago, "Improved description of atomic environments using low-cost polynomial functions with compact support," *Machine Learning: Science and Technology*, 2021.

[33] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of computational physics*, vol. 117, no. 1, pp. 1–19, 1995.

[34] "Lammps." `http://lammps.sandia.gov`. Online; accessed April 6,2021.

[35] P. Wirnsberger, D. Frenkel, and C. Dellago, "An enhanced version of the heat exchange algorithm with excellent energy conservation properties," *The Journal of chemical physics*, vol. 143, no. 12, p. 124104, 2015.

[36] D. P. Sellan, E. S. Landry, J. Turney, A. J. McGaughey, and C. H. Amon, "Size effects in molecular dynamics thermal conductivity predictions," *Physical Review B*, vol. 81, no. 21, p. 214305, 2010.

[37] G. Xie, Y. Guo, B. Li, L. Yang, K. Zhang, M. Tang, and G. Zhang, "Phonon surface scattering controlled length dependence of thermal conductivity of silicon nanowires," *Physical Chemistry Chemical Physics*, vol. 15, no. 35, pp. 14647–14652, 2013.

[38] J. Behler, "Constructing high-dimensional neural network potentials: A tutorial review," *International Journal of Quantum Chemistry*, vol. 115, no. 16, pp. 1032–1050, 2015.

[39] J. Behler, "Atom-centered symmetry functions for constructing high-dimensional neural network potentials," *The Journal of chemical physics*, vol. 134, no. 7, p. 074106, 2011.

[40] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[41] R. E. Kalman and R. S. Bucy, "New results in linear filtering and prediction theory," 1961.

[42] G. L. Smith, S. F. Schmidt, and L. A. McGee, *Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle*. National Aeronautics and Space Administration, 1962.

[43] Y. He, I. Savić, D. Donadio, and G. Galli, "Lattice thermal conductivity of semiconducting bulk materials: atomistic simulations," *Physical Chemistry Chemical Physics*, vol. 14, no. 47, pp. 16209–16222, 2012.

[44] C. Carbogno, R. Ramprasad, and M. Scheffler, "Ab initio green-kubo approach for the thermal conductivity of solids," *Physical review letters*, vol. 118, no. 17, p. 175901, 2017.

Acronyms

DFT Density Functional Theory. 19, 21, 22, 24, 25, 27, 28, 29, 30, 31, 32, 33, 51, 52

EFF Empirical Force Fields. 1, 39, 48, 51

FP First Principles. vii, 1, 2, 3, 4, 9, 10, 11, 12, 13, 14, 19, 25, 27, 35, 39, 45

GAP Gaussian Approximation Potential. 1, 4, 6

LRC Local Reference Configurations. vii, 3, 4, 6, 7, 8, 13, 14, 15, 22, 23, 24, 45, 51

MD Molecular Dynamics. 1, 3, 4, 7, 12, 13, 21, 22, 23, 24, 31, 35, 36, 43, 44, 45, 46, 51

ML Machine Learning. 1, 3, 44, 47, 51

MLFF Machine Learned Force Fields. vii, viii, 1, 2, 3, 4, 11, 12, 13, 14, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 51, 52, 63

NEMD Non Equilibrium Molecular Dynamics. viii, 35, 37, 38, 39, 48, 49, 63, 66, 71, 74

NNP Neural Network Potential. viii, 35, 36, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 66, 75, 76

PBE Perdew-Burke-Ernzerhof. 19, 45

RSDS Reference Structure DataSet. vii, viii, 3, 4, 9, 13, 14, 22, 23, 24, 30, 45, 49, 67, 68, 70

SOAP Smooth Overlap of Atomic Positions. 1, 4, 5

SVD Singular Value Decomposition. 15, 16, 17, 21, 24, 27, 30, 39

VASP Vienna Ab *initio* Simulation Package. 1, 3, 19, 35, 49, 59

A. Modified CUR algorithm

The current VASP algorithm disposes in the sparsification the eigenvectors that are most strongly connected with the least significant eigenvalue, instead of the keeping the eigenvectors that are most strongly connected with the most significant eigenvalues. In this appendix, we ask the question whether this is a sensible approach or potentially worsens the final precision.

To show that both approaches are equivalent, we define two leverage scoring. The first equation will be used to dispose of the least significant eigenvectors (those with the largest ω_j^{low})

$$\omega_j^{\text{low}} = \frac{1}{N_B} \sum_{\xi=1}^{N_B} \gamma_{\xi j}^{\text{low}}, \quad \text{with} \quad \gamma_{\xi j}^{\text{low}} = \begin{cases} u_{j\xi}^2 & \text{if } l_\xi < \varepsilon_{\text{CUR}} \\ 0 & \text{else} \end{cases}, \tag{A.1}$$

whereas the second 'standard' one would be used to keep the most significant eigenvectors (those with the largest ω_j^{high})

$$\omega_j^{\text{high}} = \frac{1}{N_B} \sum_{\xi=1}^{N_B} \gamma_{\xi j}^{\text{high}}, \quad \text{with} \quad \gamma_{\xi j}^{\text{high}} = \begin{cases} u_{j\xi}^2 & \text{if } l_\xi \geq \varepsilon_{\text{CUR}} \\ 0 & \text{else} \end{cases}. \tag{A.2}$$

The modification to the CUR algorithm mentioned in Section 2.4.2 is to change the scaling of the leverage score as done here. Note that the scaling is arbitrary, since the number of eigenvectors that are kept/disposed is given by the number of eigenvalues above/below the threshold. To achieve this, the leverage scores are sorted in ascending order, and the list is transgressed until the appropriate number of eigenvectors is found. Eigenvectors j with the highest ω_j^{high} will be kept, or eigenvectors with the highest ω_j^{low} are disposed of. If a simple linear relation between both scoring exists, one can use either of the two criteria.

Using the orthonormality of the eigenvectors it is simple to show the linear relation

$$\sum_{j=1}^{N_B} \omega_j^{\text{high}} + \sum_{j=1}^{N_B} \omega_j^{\text{low}} = 1. \tag{A.3}$$

Since the leverage scores are normalized they have the property

$$\sum_{i=j}^{N_B} \omega_j^{\text{all}} = 1, \tag{A.4}$$

where $\omega_j^{\text{all}} > 0$ and is defined as

$$\omega_j^{\text{all}} = \frac{1}{N_B} \sum_{\xi=1}^{N_B} \gamma_{\xi j}^{\text{all}}, \quad \text{with} \quad \gamma_{\xi j}^{\text{all}} = u_{j\xi}^2. \tag{A.5}$$

Using Eqs. (A.1)-(A.2) and (A.5) we can show the relation between the different leverage scores

$$\omega_j^{\text{high}} + \omega_j^{\text{low}} = \omega_j^{\text{all}}, \tag{A.6}$$

from this in combination with the normalization condition in Eq. (A.4) obviously follows that

$$\sum_{j=1}^{N_B} \omega_j^{\text{high}} = 1 - \sum_{j=1}^{N_B} \omega_j^{\text{low}}. \tag{A.7}$$

This establishes that sorting eigenvectors according to ω^{low} or ω^{high} yields just the reverse order.

B. Additional Figures

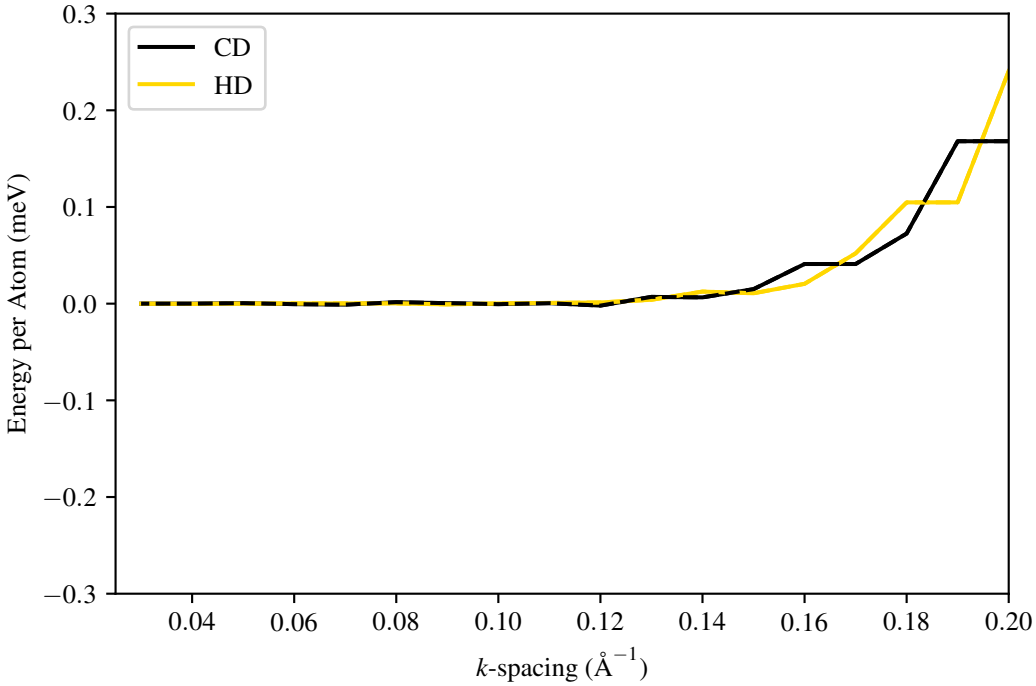
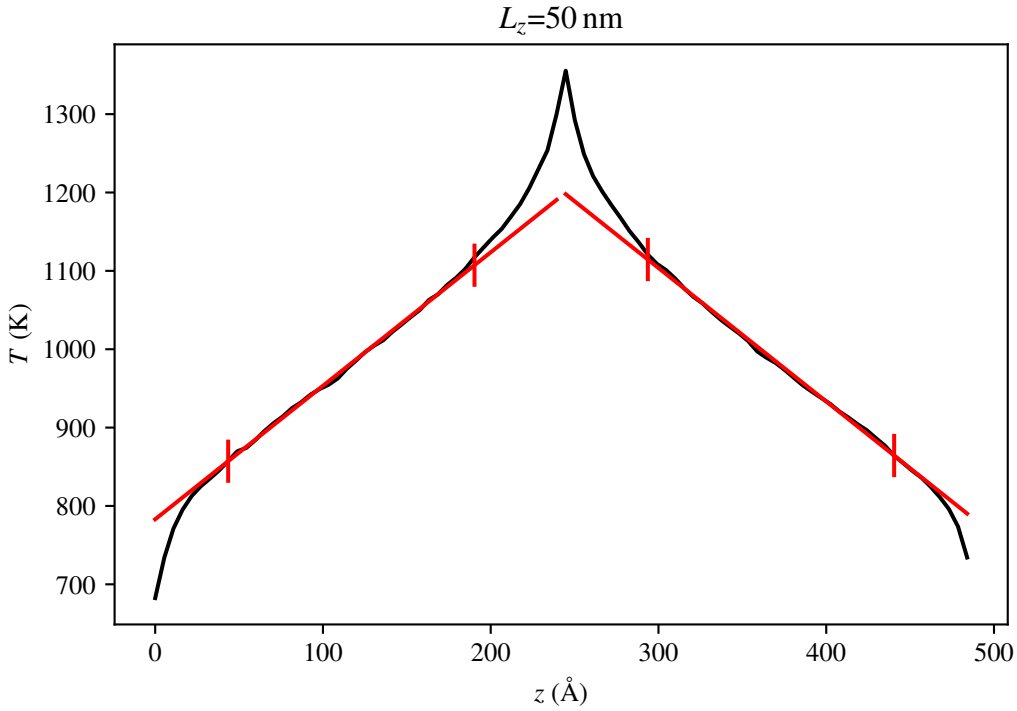
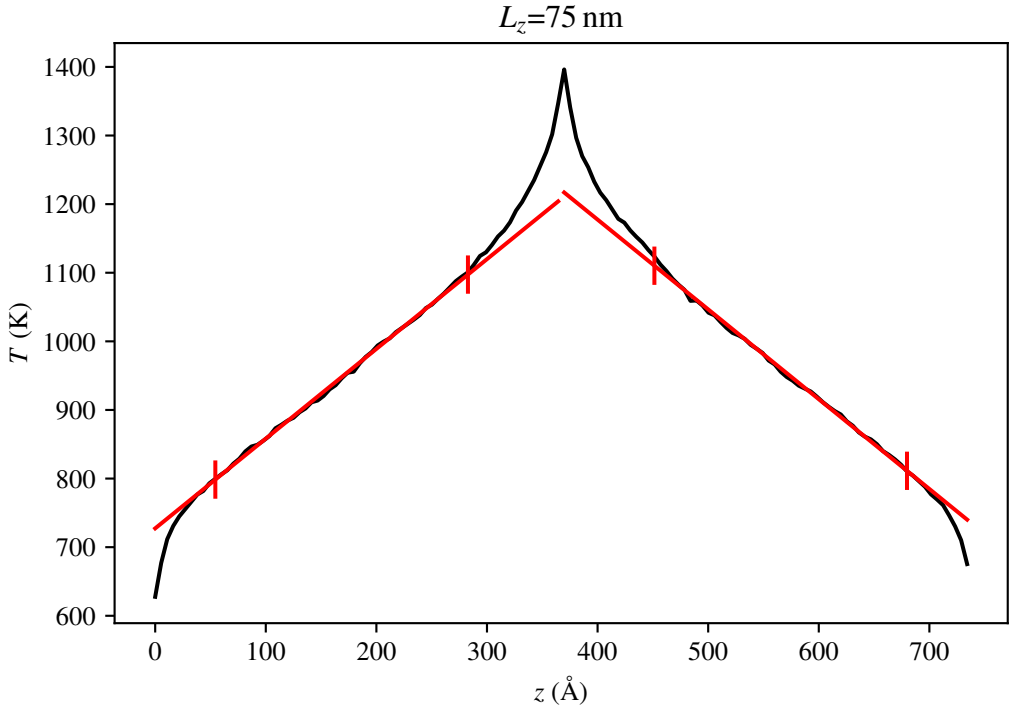


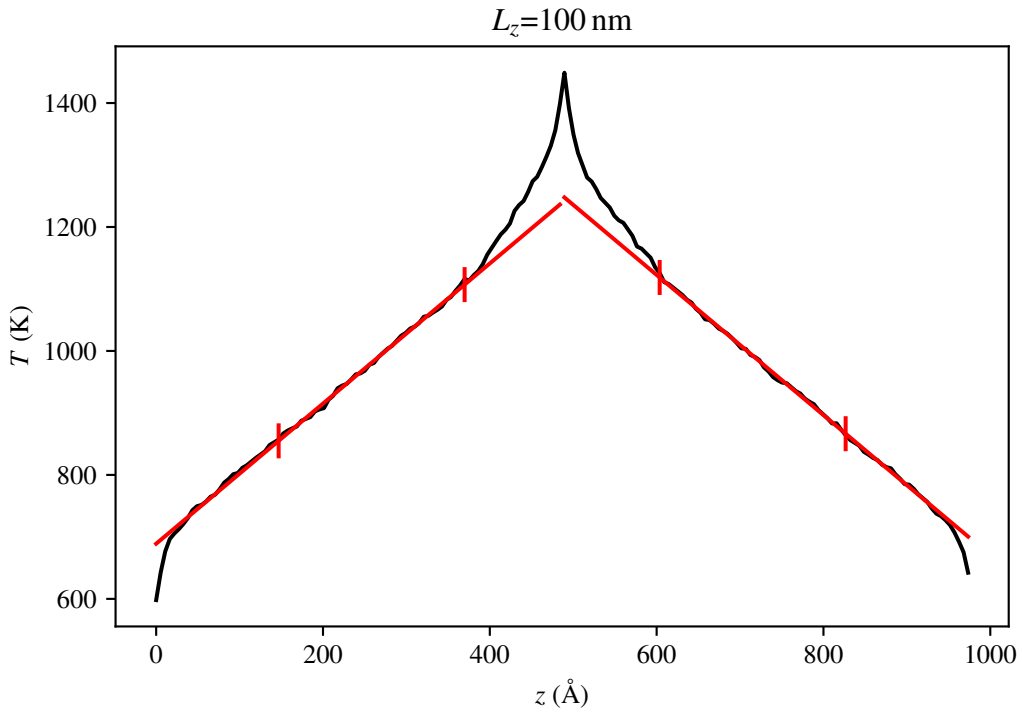
Figure B.1.: The energy difference per atom vs smallest allowed spacing between k-points for the cd and hd bulk crystal lattice structures.



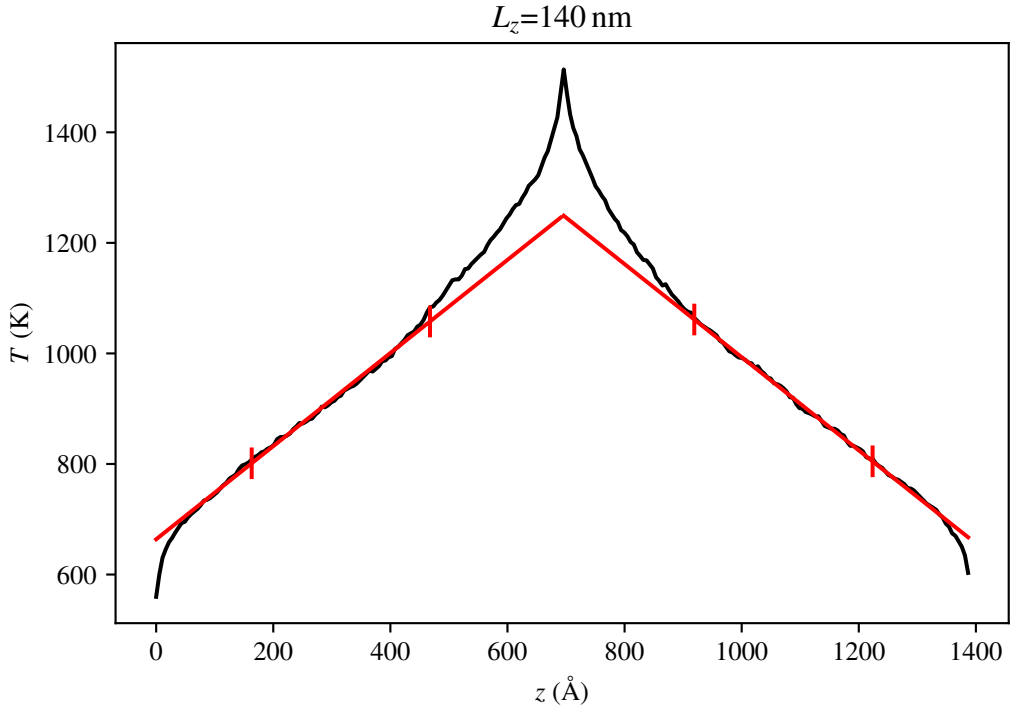
(a) The thermal profile of the 48.9 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 986\,\mathrm{K}$.



(b) The thermal profile of the 74.0 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 954\,\mathrm{K}$.

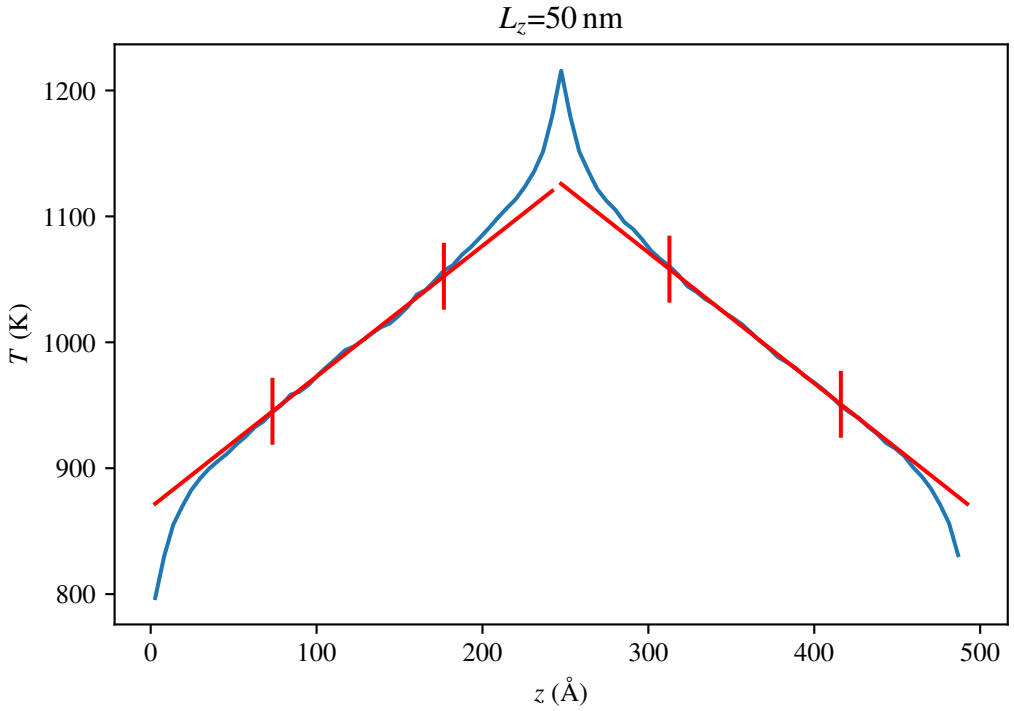


(c) The thermal profile of the 97.9 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 987\,\mathrm{K}$.

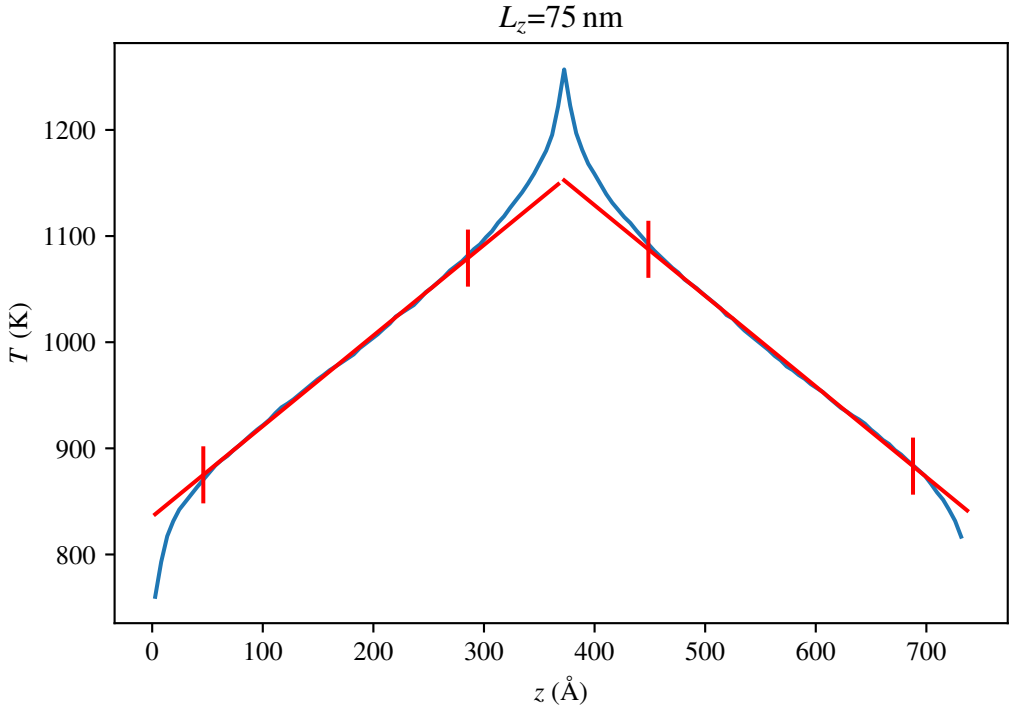


(d) The thermal profile of the 139.2 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 931\,\mathrm{K}$.

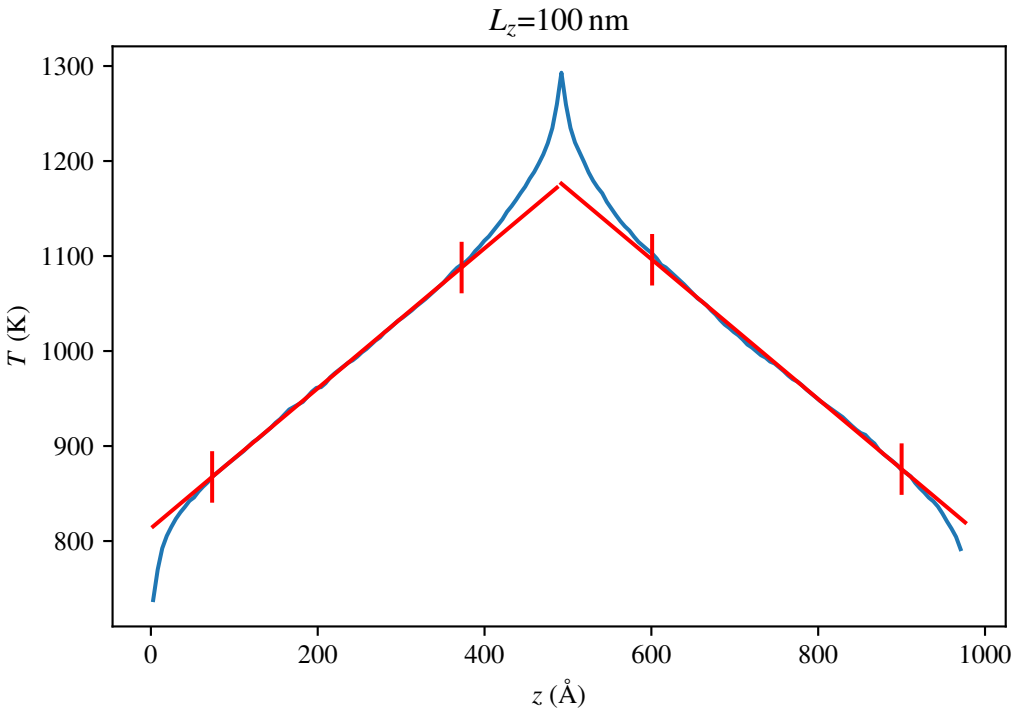
Figure B.2.: The thermal profiles of the NEMD simulation using the MLFF.



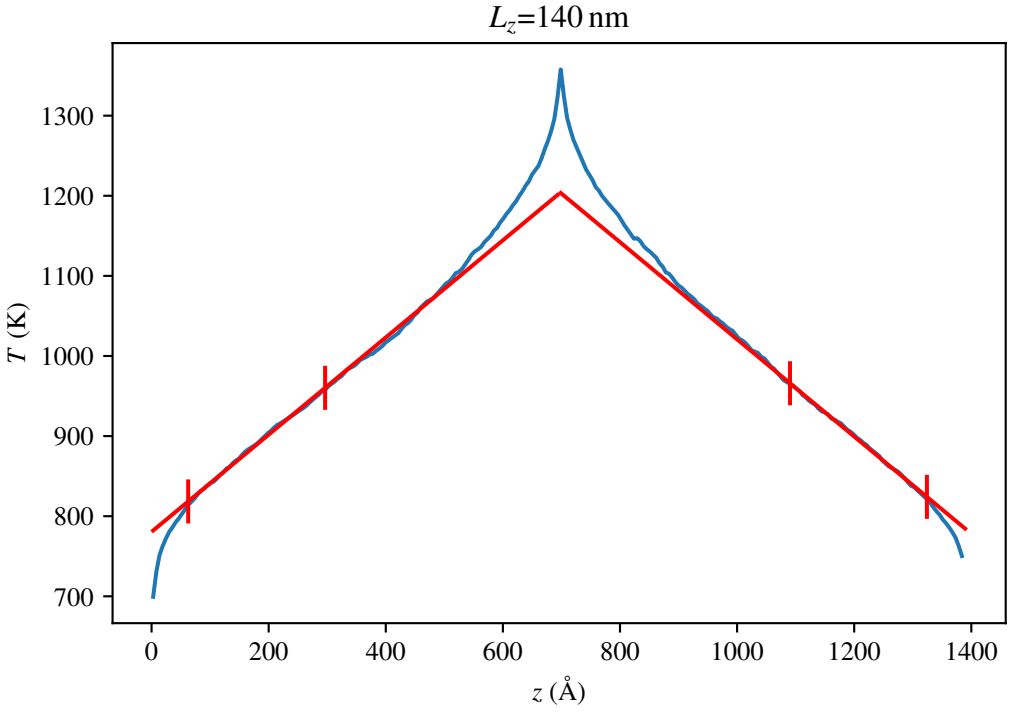
(a) The thermal profile of the 48.9 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 1002\,\mathrm{K}$.



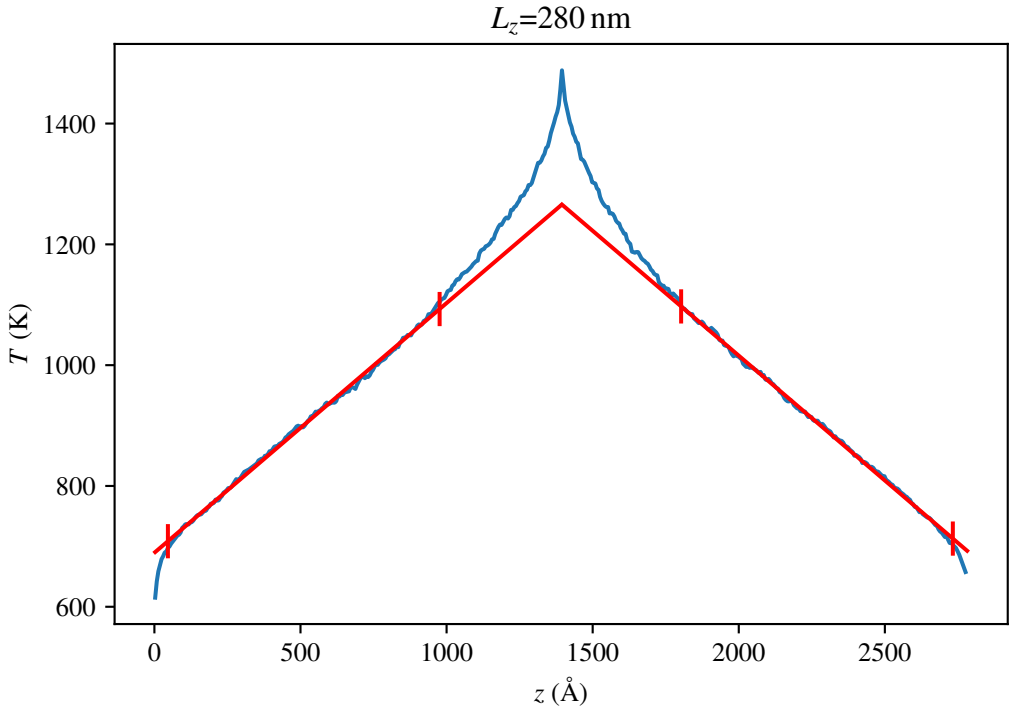
(b) The thermal profile of the 74.0 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 981\,\mathrm{K}$.



(c) The thermal profile of the 97.9 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 982\,\mathrm{K}$.



(d) The thermal profile of the 139.2 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 893\,\mathrm{K}$.



(e) The thermal profile of the 278.4 nm simulation cell. The area over which the fit was performed is highlighted by the horizontal red lines and has a mean temperature of $\langle T \rangle = 903\,\mathrm{K}$.

Figure B.3.: The thermal profiles of the NEMD simulation using the NNP.

C. Generating additional RSDS

In order to generate additional configurations for the MLFF new structures have to be generated. Here is a small code to mirror the atomic movement of a reference structure during a MD simulation (saved in "POSCAR_ref" and "XDARCAR_ref") onto a new structure (saved in another "POSCAR" file). These modified structures can be used for further training.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import numpy as np

def vec_to_polar(x):
    r = np.linalg.norm(x)
    theta = np.arccos(x[2]/r)
    phi = np.arctan2(x[1],x[0])
    return np.array([r,theta,phi])

def vec_to_cartesian(r):
    x = r[0]*np.sin(r[1])*np.cos(r[2])
    y = r[0]*np.sin(r[1])*np.sin(r[2])
    z = r[0]*np.cos(r[1])
    return np.array([x,y,z])

def to_direct(cell):
    a1 = cell[0]
    a2 = cell[1]
    a3 = cell[2]
    Pos = cell[3:]
    Out=[a1, a2, a3]
    omg = np.dot(a1,np.cross(a2,a3))
    sig_a1 = np.cross(a2,a3)/omg
    sig_a2 = np.cross(a3,a1)/omg
    sig_a3 = np.cross(a1,a2)/omg
    for i in Pos:
        Out.append(np.array([np.dot(i,sig_a1),np.dot(i,sig_a2),np.dot(i,
    sig_a3)]))
    return np.array(Out)

def to_cartesian(cell):
    a1 = cell[0]
    a2 = cell[1]
    a3 = cell[2]
    Pos = cell[3:]
    Out=[a1, a2, a3]
    for i in Pos:
        Out.append(i[0]*a1 + i[1]*a2 + i[2]*a3)
    return np.array(Out)

def make_POSCAR(path,cell,structur_name,atom_type,direct = True):
```

```
    F = open(path,'w')
    F.write(structur_name + '\n 1.0\n')
    for line in cell[:3]:
        F.write(str(line).replace("'","").replace("[","").replace("]","")+
    '\n')
    F.write(atom_type+'\n'+str(len(cell[3:]))+'\n')
    if direct :
        F.write('Direct\n')
    else:
        F.write('Cartesian\n')
    for line in cell[3:]:
        F.write(str(line).replace("'","").replace("[","").replace("]","")+
    '\n')
    F.close()

def get_POSCAR(filname = 'POSCAR',direct = True):
    F = open(filname,'r')
    POSCAR = []
    for line in F:
        POSCAR.append(line)
    F.close()
    bP = float(POSCAR[1])
    a1 = bP*np.array(POSCAR[2].split(),dtype=np.float64)
    a2 = bP*np.array(POSCAR[3].split(),dtype=np.float64)
    a3 = bP*np.array(POSCAR[4].split(),dtype=np.float64)
    omg = np.dot(a1,np.cross(a2,a3))
    sig_a1 = np.cross(a2,a3)/omg
    sig_a2 = np.cross(a3,a1)/omg
    sig_a3 = np.cross(a1,a2)/omg
    n = int(POSCAR[6])
    Pos =[a1, a2, a3]
    for i in range(n):
        buf = np.array(POSCAR[8+i].split(),dtype=np.float64)
        if (POSCAR[7][0] == 'C' or POSCAR[7][0] == 'c') and direct :
            buf = np.array([np.dot(buf,sig_a1),np.dot(buf,sig_a2),np.dot(
    buf,sig_a3)])
        if (POSCAR[7][0] == 'D' or POSCAR[7][0] == 'd') and not direct :
            buf = buf[0]*a1 + buf[1]*a2 + buf[2]*a3
        Pos.append(buf)
    return np.array(Pos)

def get_XDATCAR(filname='XDATCAR',direct = True):
    F = open(filname,'r')
    XDATCAR=[]
    for line in F:
        XDATCAR.append(line)
    n = int(XDATCAR[6])
    q = n + 8
    m = len(XDATCAR)//q
    Out = []
    for i in range(m):
        bP = float(XDATCAR[q*i+1])
        a1 = bP*np.array(XDATCAR[q*i+2].split(),dtype=np.float64)
        a2 = bP*np.array(XDATCAR[q*i+3].split(),dtype=np.float64)
        a3 = bP*np.array(XDATCAR[q*i+4].split(),dtype=np.float64)
        omg = np.dot(a1,np.cross(a2,a3))
        sig_a1 = np.cross(a2,a3)/omg
```

```
        sig_a2 = np.cross(a3,a1)/omg
        sig_a3 = np.cross(a1,a2)/omg
        Pos =[a1, a2, a3]
        for j in range(n):
            buf = np.array(XDATCAR[q*i+8+j].split(),dtype=np.float64)
            if (XDATCAR[7][0] == 'C' or XDATCAR[7][0] == 'c') and direct :
                buf = np.array([np.dot(buf,sig_a1),np.dot(buf,sig_a2),np.dot(buf,sig_a3)])
            if (XDATCAR[7][0] == 'D' or XDATCAR[7][0] == 'd') and not direct :
                buf = buf[0]*a1 + buf[1]*a2 + buf[2]*a3
            Pos.append(buf)
        Out.append(Pos)
    return np.array(Out)

def compensate_SHIFT(POSCAR,XDATCAR,direct = True):
    if direct:
        for i in range(len(XDATCAR)):
            XDATCAR[i] = to_cartesian(XDATCAR[i])
        POSCAR = to_direct(POSCAR)
    refPos = POSCAR[3:]
    OUT = []
    for Pos in XDATCAR:
        buf = Pos
        b1 = np.max([Pos[0][0],Pos[1][0],Pos[2][0]])
        b2 = np.max([Pos[0][1],Pos[1][1],Pos[2][1]])
        b3 = np.max([Pos[0][2],Pos[1][2],Pos[2][2]])
        P = Pos[3:] - refPos
        dx, dy, dz = 0, 0, 0
        for x in P:
            dx += x[0] - np.round(x[0]/b1,0) * b1
            dy += x[1] - np.round(x[1]/b2,0) * b2
            dz += x[2] - np.round(x[2]/b3,0) * b3
        r = np.array([dx, dy, dz])/len(P)
        buf[3:] = buf[3:] - r
        OUT.append(buf)
    for i in range(len(OUT)):
        OUT[i] = to_direct(OUT[i])
        OUT[i][3:] = np.where(OUT[i][3:]>=0, OUT[i][3:], OUT[i][3:]+1)
        OUT[i][3:] = np.where(OUT[i][3:]< 1, OUT[i][3:], OUT[i][3:]-1)
    if not direct:
        for i in range(len(OUT)):
            OUT[i] = to_cartesian(OUT[i])
    return np.array(OUT)

def create_DATA(POSCAR,XDATCAR,direct = True):
    XD = XDATCAR
    PO = POSCAR
    if direct:
        for i in range(len(XD)):
            XD[i] = to_cartesian(XD[i])
    if not direct:
        PO = to_direct(PO)
    Out = []
    for i in range(1,len(XD)):
        refPos = XD[i-1]
        Pos = XD[i]
```

```
151         b1 = np.max([Pos[0][0],Pos[1][0],Pos[2][0]])
152         b2 = np.max([Pos[0][1],Pos[1][1],Pos[2][1]])
153         b3 = np.max([Pos[0][2],Pos[1][2],Pos[2][2]])
154         P = Pos[3:] - refPos[3:]
155         dx, dy, dz = 0, 0, 0
156         for x in P:
157             dx += x[0] - np.round(x[0]/b1,0) * b1
158             dy += x[1] - np.round(x[1]/b2,0) * b2
159             dz += x[2] - np.round(x[2]/b3,0) * b3
160         r = np.array([dx, dy, dz])/len(Pos)
161         Pos[3:] = Pos[3:] - r
162         refPos = to_direct(refPos)
163         Pos = to_direct(Pos)
164         for j in range(3):
165             f = vec_to_polar(Pos[j])
166             ra= vec_to_polar(refPos[j])
167             f[0] = f[0]/ra[0]
168             f[1:]= f[1:] - ra[1:]
169             PO[j] = vec_to_polar(PO[j])
170             PO[j][0] = f[0]*PO[j][0]
171             PO[j][1:] = f[1:]+PO[j][1:]
172             PO[j] = vec_to_cartesian(PO[j])
173         PO[3:] = PO[3:] + Pos[3:] - refPos[3:]
174         if np.mod(i,10) == 0:
175           make_POSCAR('Pos/POSCAR'+str(i//10),d[i],'StructureName','
    AtomType',direct = True)
176 #         Out.append(PO)
177 #    return np.array(Out)
178
179 a = get_POSCAR('POSCAR_ref')
180 b = get_XDATCAR('XDATCAR_ref')
181 b = np.append([a],b,axis=0)
182 c = get_POSCAR('POSCAR')
183 d = create_DATA(c,b)
184 #for i in range(len(d)):
185 # if np.mod(i,10) == 0:
186 #    make_POSCAR('Pos/POSCAR'+str(i//10),d[i],'StructureName','AtomType',
    direct = True)
```

Listing C.1: Structur generation for suplimental training.

D. Müller-Plathe algorithm

Here we show a module written in the programming language ‘Fortran 90’ for use in NEMD simulation.

```
MODULE muller_plathe

USE prec
IMPLICIT NONE

INTEGER :: NEMD_NSLABS
INTEGER :: NEMD_MPW

CONTAINS

SUBROUTINE NEMD_READER(IU0,IU5)
USE reader_tags
USE tutor, ONLY: vtutor

IMPLICIT NONE
INTEGER IU5, IU0, IERR
LOGICAL LOPEN

CALL OPEN_INCAR_IF_FOUND(IU5, LOPEN)

! Read NEMD_NSLABS
NEMD_NSLABS=0
CALL PROCESS_INCAR(LOPEN, IU0, IU5,'NEMD_NSLABS', NEMD_NSLABS, IERR,
    WRITEXMLINCAR)
! impose even number of slabs
IF (MOD(NEMD_NSLABS,2).GT.0) THEN
NEMD_NSLABS=NEMD_NSLABS+1
CALL vtutor%alert("WARNING: MP_READER: NEMD_NSLABS entered is odd ->
    adding 1")
ENDIF
! Read NEMD_MPW
NEMD_MPW=50
CALL PROCESS_INCAR(LOPEN, IU0, IU5, 'NEMD_MPW', NEMD_MPW, IERR,
    WRITEXMLINCAR)

END SUBROUTINE


  SUBROUTINE SWAP_V(NIONS,POMASS,NTYP,ITYP,POTIM,X,V,JTMP)
USE constant

IMPLICIT NONE
INTEGER, INTENT(IN) :: NIONS, NTYP
INTEGER, INTENT(IN) :: ITYP(:)
```

```
REAL(q), INTENT(IN) :: X(:,:)
REAL(q), INTENT(IN) :: POMASS(:)
REAL(q), INTENT(IN) :: POTIM
REAL(q), INTENT(INOUT) :: V(:,:)
REAL(q), INTENT(INOUT) :: JTMP
! work variables
INTEGER :: I, IC, IH
REAL(q) :: XC(3,NIONS), DZ, V2, V2C, V2H, VC(3), VH(3)
REAL(q) :: UL, UT, FACT

!unit conversion factors
UL=1E-10_q
UT=POTIM*1E-15_q
FACT=(AMTOKG/EVTOJ)*(UL/UT)**2

! assume flux along z; rectangular slab (box -0.5:0.5)
! X is in crystal coordinates, V in cartesian
DZ = 1.0_q/DBLE(NEMD_NSLABS)
V2C = 0.0_q
V2H = 1.0E30_q

DO I=1,NIONS
! cold slab (first slab)
IF (X(3,I) .GE. 0.0_q .AND. X(3,I) .LT. 0.0_q+DZ ) THEN
V2 = POMASS(ITYP(I)) * ( V(1,I)**2 + V(2,I)**2 + V(3,I)**2 )
IF (V2C .LT. V2) THEN
IC = I
VC(1:3) = V(1:3,I)
V2C = V2
ENDIF
ENDIF
        ! hot slab (in the middle)
IF (X(3,I) .GE. 0.5_q .AND. X(3,I) .LT. 0.5_q+DZ) THEN
V2 = POMASS(ITYP(I)) * ( V(1,I)**2 + V(2,I)**2 + V(3,I)**2 )
IF (V2H .GT. V2) THEN
IH = I
VH(1:3) = V(1:3,I)
V2H = V2
ENDIF
ENDIF
ENDDO

V(1:3,IC) = -VC(1:3) + 2*( POMASS(ITYP(IC))*VC(1:3)+POMASS(ITYP(IH))*VH
    (1:3) ) / &
( POMASS(ITYP(IC)) + POMASS(ITYP(IH)) )
V(1:3,IH) = -VH(1:3) + 2*( POMASS(ITYP(IC))*VC(1:3)+POMASS(ITYP(IH))*VH
    (1:3) ) / &
( POMASS(ITYP(IC)) + POMASS(ITYP(IH)) )

JTMP = JTMP + 0.5_q*FACT*POMASS(ITYP(IH)) *( V(1,IH)**2+V(2,IH)**2+V(3,IH)
    **2 &
-VH(1)**2-VH(2)**2-VH(3)**2 )

END SUBROUTINE

!***********************************************
! For (modified) Muller-Plathe NEMD: determine the temperature in each
```

```
! slab
!*******************************************

SUBROUTINE SLAB_T (NIONS,POMASS,NTYP,ITYP,POTIM,X,V,TTMP)
USE constant

IMPLICIT NONE
INTEGER, INTENT(IN) :: NIONS, NTYP
INTEGER, INTENT(IN) :: ITYP(:)
REAL(q), INTENT(IN) :: X(:,:)
REAL(q), INTENT(IN) :: V(:,:)
REAL(q), INTENT(IN) :: POMASS(:)
REAL(q), INTENT(IN) :: POTIM
REAL(q), INTENT(INOUT) :: TTMP(:)
! work variables
INTEGER :: I, J, NAT(NEMD_NSLABS)
REAL(q) :: DZ, T(NEMD_NSLABS)
REAL(q) :: UL, UT, FACT

!unit conversion factors
UL=1E-10_q
UT=POTIM*1E-15_q
FACT=(AMTOKG/EVTOJ)*(UL/UT)**2/BOLKEV/3.0_q

T(:) = 0.0_q
DZ = 1.0_q/DBLE(NEMD_NSLABS)
NAT(:) = 0
DO I=1,NEMD_NSLABS
DO J=1,NIONS
IF ( X(3,J).GE.((I-1)*DZ) .AND. X(3,J).LT.(I*DZ) ) THEN
NAT(I) = NAT(I)+1
T(I) = T(I)+POMASS(ITYP(J))*(V(1,J)**2+V(2,J)**2+V(3,J)**2)
ENDIF
ENDDO
ENDDO

TTMP(:) = TTMP(:)+T(:)*FACT/DBLE(NAT(:))

END SUBROUTINE

SUBROUTINE GET_J(A,POTIM,NT,JTMP,JT)
USE mymath, ONLY: CROSSPROD

IMPLICIT NONE
INTEGER, INTENT(IN) :: NT
REAL(q), INTENT(IN) :: A(:,:)
REAL(q), INTENT(IN) :: POTIM
REAL(q), INTENT(IN) :: JTMP
REAL(q), INTENT(OUT) :: JT
! work variables
REAL(q) :: AREA, ATMP(3)

! here assume slab elongated along z (to change)
! area is in A^2, dt in fs
ATMP = CROSSPROD(3,A(:,1),A(:,2))
AREA = SQRT(ATMP(1)**2+ATMP(2)**2+ATMP(3)**2)
JT = 0.5_q * JTMP / (AREA*POTIM*NT)
```

```
153
154 END SUBROUTINE
155
156
157 END MODULE muller_plathe
```

Listing D.1: NEMD Müller-Plathe algorithm

E. NNP Symmetry functions

Table E.1.: The symmetry functions with the corresponding parameters are listed.

Index	Symmetry function	η	r_s	r_c	λ	ζ
1	G^2	2.778×10^{-2}	0.000	6		
2	G^2	4.843×10^{-2}	0.000	6		
3	G^2	8.445×10^{-2}	0.000	6		
4	G^2	1.473×10^{-1}	0.000	6		
5	G^2	2.568×10^{-1}	0.000	6		
6	G^2	4.477×10^{-1}	0.000	6		
7	G^2	5.302×10^{-1}	4.627	6		
8	G^2	7.806×10^{-1}	0.000	6		
9	G^2	8.917×10^{-1}	3.568	6		
10	G^2	1.361	0.000	6		
11	G^2	1.500	2.751	6		
12	G^2	2.522	2.121	6		
13	G^2	4.241	1.636	6		
14	G^2	7.133	1.261	6		
15	G^3	2.778×10^{-2}	0.000	6	-1	1.0
16	G^3	2.778×10^{-2}	0.000	6	1	1.0
17	G^3	2.778×10^{-2}	0.000	6	-1	4.0
18	G^3	2.778×10^{-2}	0.000	6	1	4.0
19	G^3	2.778×10^{-2}	0.000	6	-1	6.0
20	G^3	2.778×10^{-2}	0.000	6	1	6.0
21	G^3	5.778×10^{-2}	0.000	6	-1	1.0
22	G^3	5.778×10^{-2}	0.000	6	1	1.0
23	G^3	5.778×10^{-2}	0.000	6	-1	4.0
24	G^3	5.778×10^{-2}	0.000	6	1	4.0
25	G^3	5.778×10^{-2}	0.000	6	-1	6.0
26	G^3	5.778×10^{-2}	0.000	6	1	6.0
27	G^3	1.202×10^{-1}	0.000	6	-1	1.0
28	G^3	1.202×10^{-1}	0.000	6	1	1.0
29	G^3	1.202×10^{-1}	0.000	6	-1	4.0
30	G^3	1.202×10^{-1}	0.000	6	1	4.0
31	G^3	1.202×10^{-1}	0.000	6	-1	6.0
32	G^3	1.202×10^{-1}	0.000	6	1	6.0
33	G^3	2.500×10^{-1}	0.000	6	-1	1.0
34	G^3	2.500×10^{-1}	0.000	6	1	1.0
35	G^3	2.500×10^{-1}	0.000	6	-1	4.0

Index	Symmetry function	η	r_s	r_c	λ	ζ
36	G^3	2.500×10^{-1}	0.000	6	1	4.0
37	G^3	2.500×10^{-1}	0.000	6	−1	6.0
38	G^3	2.500×10^{-1}	0.000	6	1	6.0
39	G^9	2.778×10^{-2}	0.000	6	−1	1.0
40	G^9	2.778×10^{-2}	0.000	6	1	1.0
41	G^9	2.778×10^{-2}	0.000	6	−1	4.0
42	G^9	2.778×10^{-2}	0.000	6	1	4.0
43	G^9	2.778×10^{-2}	0.000	6	−1	6.0
44	G^9	2.778×10^{-2}	0.000	6	1	6.0
45	G^9	5.778×10^{-2}	0.000	6	−1	1.0
46	G^9	5.778×10^{-2}	0.000	6	1	1.0
47	G^9	5.778×10^{-2}	0.000	6	−1	4.0
48	G^9	5.778×10^{-2}	0.000	6	1	4.0
49	G^9	5.778×10^{-2}	0.000	6	−1	6.0
50	G^9	5.778×10^{-2}	0.000	6	1	6.0
51	G^9	1.202×10^{-1}	0.000	6	−1	1.0
52	G^9	1.202×10^{-1}	0.000	6	1	1.0
53	G^9	1.202×10^{-1}	0.000	6	−1	4.0
54	G^9	1.202×10^{-1}	0.000	6	1	4.0
55	G^9	1.202×10^{-1}	0.000	6	−1	6.0
56	G^9	1.202×10^{-1}	0.000	6	1	6.0
57	G^9	2.500×10^{-1}	0.000	6	−1	1.0
58	G^9	2.500×10^{-1}	0.000	6	1	1.0
59	G^9	2.500×10^{-1}	0.000	6	−1	4.0
60	G^9	2.500×10^{-1}	0.000	6	1	4.0
61	G^9	2.500×10^{-1}	0.000	6	−1	6.0
62	G^9	2.500×10^{-1}	0.000	6	1	6.0