Freie Universität Berlin

Accurate implementation of DeepMD-v2 Potential calculation in HALMD for single species, extension to multi-species and Automatic-Differentiation-Based Force Computation

by

Sandip Kumar Sah

Master Thesis in Computational Science

Submission: 09 January 2026 Supervisor: Prof. Dr. Felix Höfling

Freie Universität Berlin | Department of Mathematics and Computer Science
Institute of Mathematics

Statutory Declaration

Family Name, Given/First Name	Sah, Sandip Kumar
Matriculation number	5589263
Kind of thesis submitted	Master Thesis

English: Declaration of Authorship

I hereby declare that the thesis submitted was created and written solely by myself without any external support. Any sources, direct or indirect, are marked as such. I am aware of the fact that the contents of the thesis in digital form may be revised with regard to usage of unauthorized aid as well as whether the whole or parts of it may be identified as plagiarism. I do agree my work to be entered into a database for it to be compared with existing sources, where it will remain in order to enable further comparisons with future theses. This does not grant any rights of reproduction and usage, however.

This document was neither presented to any other examination board nor has it been published.

German: Erklärung der Autorenschaft (Urheberschaft)

Ich erkläre hiermit, dass die vorliegende Arbeit ohne fremde Hilfe ausschließlich von mir erstellt und geschrieben worden ist. Jedwede verwendeten Quellen, direkter oder indirekter Art, sind als solche kenntlich gemacht worden. Mir ist die Tatsache bewusst, dass der Inhalt der Thesis in digitaler Form geprüft werden kann im Hinblick darauf, ob es sich ganz oder in Teilen um ein Plagiat handelt. Ich bin damit einverstanden, dass meine Arbeit in einer Datenbank eingegeben werden kann, um mit bereits bestehenden Quellen verglichen zu werden und dort auch verbleibt, um mit zukünftigen Arbeiten verglichen werden zu können. Dies berechtigt jedoch nicht zur Verwendung oder Vervielfältigung.

Diese Arbeit wurde noch keiner anderen Prüfungsbehörde vorgelegt noch wurde sie bisher veröffentlicht.

.....

Date, Signature

Abstract

Contents

1 Introduction

1.1 Motivation

Molecular dynamics (MD) simulations play a central role in materials science by providing atomistic insight into the behavior of complex systems. Their predictive power depends on the interatomic potential used to approximate the underlying potential energy surface (PES). Classical empirical potentials are efficient but often too rigid to describe complex bonding environments, whereas *ab initio* methods offer high accuracy at prohibitive computational cost. Machine-learned interatomic potentials—particularly the Deep Potential Molecular Dynamics (DeepMD) framework—bridge this gap by achieving near–quantum mechanical accuracy with classical-MD performance.

The work by Andrés Cruz, titled *"Deep Neural Networks Potentials for Scalable Molecular Dynamics Simulations on Accelerator Hardware"* [1], represents an important step toward integrating Deep Potential models into the high-performance GPU-accelerated simulation engine HALMD. His work reconstructs the DeePMD-kit inference pipeline inside HALMD, extracts network weights from a trained TensorFlow model, and validates energy and force predictions for a *single-species Copper system*. In particular, his implementation focuses on the *two-body embedding smooth edition, DeepPot-SE descriptor* and reproduces the filter and fitting networks *only for a monoatomic system*, making it suitable for single metal.

However, the implementation in [1] remains limited to the simplest case of DeepMD-v2 architectures. It does not include multi-species support.

Additionally, several intermediate steps present in the full DeepMD-v2 computational graph—such as normalization layers, ghost body extension for periodicity in boundary condition, species-wise descriptor partitioning, and certain derivative pathways—were simplified or omitted in the previous implementation. Cruz's work successfully established a working single-species DeepMD integration within HALMD.

This work builds directly on the work of Cruz [1] and advances it substantially. The primary contributions of the present work are:

1. Generalizing the HALMD Deep Potential integration to *multi-species, multi-body DeepMD-v2 models*, enabling simulations of binary and multicomponent alloy systems.
2. Reconstructing descriptor and fitting networks for the *full multi-species DeepPot-SE architecture*, including multi-body descriptor terms.
3. Improving the *accuracy* of the HALMD implementation by adding several computational steps missing in the previous work, such as proper cutoff normalization, multi-species descriptor algebra and so on.

Through these extensions, the present thesis transforms HALMD from supporting a prototype single-component DeepMD potential into a *high-accuracy, multi-species DeepMD-v2 engine*. This significantly broadens HALMD's applicability, enabling large-scale molecular dynamics simulations of technologically relevant multicomponent materials at near–quantum mechanical accuracy.

1.2 Objectives and Scope

The objective of this thesis is to extend and generalize the Deep Potential (DeePot) implementation in HALMD beyond the single-species, two-body descriptor developed by Cruz [1]. While Cruz's work successfully demonstrated that HALMD can evaluate a DeePMD-v2 model for a monoatomic copper system, several components required for full multi-species DeepMD-v2 inference were missing. Most notably, the previous implementation did not include (i) the periodic coordinate extension and neighbor-list construction used in DeepMD [2], (ii) normalization and scaling layers defined in the DP-v2 framework [3], (iii) species-dependent descriptors, or (iv) descriptor and filter weights that depend simultaneously on the central and neighbor species, as introduced in the multi-species DeepPot-SE descriptor [3].

This thesis addresses these limitations by implementing the complete multi-species DeepMD-v2 inference pipeline, including accurate periodic handling of atomic environments. Specifically, the thesis pursues the following objectives:

1. **Implement coordinate normalization, ghost-cell extension, and multi-type neighbor list construction.** The previous implementation did not include the canonical DeePMD preprocessing steps for periodic systems—wrapping coordinates into the primary simulation cell, generating ghost atoms to cover the cutoff radius, and constructing species-grouped neighbor lists—as described in the DeePMD methodology [2, 3]. Implementing these steps ensures that HALMD reproduces the correct local environments required by the DeepPot-SE descriptors under periodic boundary conditions.

2. **Generalize the descriptor pipeline to multi-species systems.** This includes implementing species-dependent neighbor counts $N_c(a)$, species-aware embedding matrices G_i, and the full multi-species DeepPot-SE (se_e2_a) descriptor introduced in DP-v2 [3].

3. **Implement species-dependent filter networks.** Extend the single-species embedding and filter networks used in [1] to support arbitrary numbers of atomic types, following the species-indexed filter network formulation defined in the DP-v2 architecture [3].

4. **Reproduce the full DeepMD-v2 inference procedure with higher accuracy.** Incorporate several computational steps omitted in previous work, including normalization layers, descriptor scaling operations, smooth cutoff functions, and full force backpropagation through descriptor derivatives, consistent with the DeepMD-v2 formulation [3, 2].

5. **Validate energy and force predictions for multi-component systems.** Compare results from HALMD against the DeePMD-kit reference implementation for multi-species models and quantify numerical deviations, following standard validation procedures established in DeepMD literature [2, 3].

Scope. This thesis focuses exclusively on the inference stage of DeepMD-v2, i.e., the computation of energies and forces from pre-trained models. Model training is outside the scope of this work. The implementation targets the multi-species DeepPot-SE descriptor and does not cover other descriptor families such as end-to-end or message-passing potentials. The work provides support for multi-species atomic systems relevant to alloys and chemically complex materials, while the underlying molecular dynamics algorithms

(integrators, thermostats, barostats) rely on HALMD's existing infrastructure.

2 Background

2.1 HALMD Software

The High-Accuracy Large-scale Molecular Dynamics (HALMD) package is a high-performance, open-source molecular dynamics (MD) simulation framework designed to study the microscopic dynamics of liquids, glasses, and other condensed matter systems. HALMD is built around a modular C++ architecture with a strong emphasis on numerical precision, extensibility, and efficient parallel computation on graphics processing units (GPUs) [4]. It provides a versatile platform for conducting large-scale classical molecular dynamics simulations involving millions of particles.

HALMD employs the *classical molecular dynamics* approach, in which atomic motion is governed by Newton's equations of motion,

$$m_i \frac{d^2\mathbf{r}_i}{dt^2} = -\nabla_i U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N),$$

where U is a predefined analytical potential describing interatomic interactions. The accuracy of a classical MD simulation depends on the accuracy of this potential. HALMD supports several built-in empirical pair potentials, such as Lennard–Jones, Gaussian Core, and Yukawa models, and additionally allows users to supply tabulated pair potentials. These empirical models do not involve quantum-mechanical calculations and thus differ from *ab initio* molecular dynamics (AIMD), where forces are computed directly from electronic structure methods.

One of HALMD's defining strengths is its high-performance GPU backend. The software employs NVIDIA's CUDA to accelerate computationally intensive tasks such as force evaluation, neighbor-list construction, and time integration [4], and more recent versions extend support to heterogeneous computing platforms through SYCL [5]. Spatial domain decomposition and cell binning enable efficient scaling on multi-GPU systems, achieving orders-of-magnitude speedups compared to CPU-only implementations. HALMD also supports mixed-precision arithmetic, selectively applying double precision to critical operations to ensure long-term numerical stability without compromising performance.

HALMD follows a modular design philosophy. Core components—integrators, interaction potentials, neighbor-list builders, and observables—are implemented as interchangeable modules configurable through a Lua scripting interface [6]. This modularity makes it straightforward to extend HALMD with new functionality, including custom potentials, analysis routines, and data exporters. Simulation data are stored in the H5MD format [7], an HDF5-based standard widely supported by analysis tools in computational physics and chemistry.

These characteristics make HALMD an ideal platform for integrating machine-learned interatomic potentials. Its GPU-accelerated, modular architecture provides the necessary infrastructure to replace conventional analytical potentials with neural-network-based force fields. In this thesis, HALMD serves as the host MD engine into which the Deep Potential Molecular Dynamics (DeepMD) model is embedded. The goal is to achieve *ab initio*-level accuracy within HALMD's efficient large-scale simulation framework by implementing a full multi-species DeepMD-v2 inference pipeline.

2.2 Neural Network Potentials

In recent years, machine learning techniques—particularly deep neural networks (DNNs) have revolutionized the construction of interatomic potentials for molecular dynamics simulations. Traditional analytical potentials, while computationally efficient, are often limited in their ability to accurately capture many-body interactions and chemical complexities across diverse atomic environments. Neural Network Potentials (NNPs) overcome these limitations by learning the potential energy surface (PES) directly from *ab initio* reference data [8, 9].

In the NNP framework, a neural network is trained to approximate the mapping between an atomic configuration and its corresponding total energy and atomic forces. The total potential energy of a system is commonly expressed as a sum of atomic contributions [8]:

$$E = \sum_i E_i(\mathcal{R}_i),$$

where E_i denotes the atomic energy associated with atom i, and $\mathcal{R}_i$ represents its local environment within a cutoff radius. This decomposition ensures extensivity and locality and enables efficient scaling with system size.

Training a neural network potential involves minimizing a loss function that combines the errors in energies, forces, and optionally virials. Following the formulation used in DeePMD-kit [2, 3], the loss is written as

$$\mathcal{L} = p_E L_E + p_F L_F + p_V L_V, \tag{1}$$

where the terms are defined as

$$L_E = \frac{1}{N_E} \sum_{n=1}^{N_E} \left(E_n^{\mathrm{NN}} - E_n^{\mathrm{DFT}}\right)^2, \tag{2}$$

$$L_F = \frac{1}{3N_F} \sum_{n=1}^{N_F} \sum_{i=1}^{N_{\mathrm{atoms}}} \left\|\mathbf{F}_{i,n}^{\mathrm{NN}} - \mathbf{F}_{i,n}^{\mathrm{DFT}}\right\|^2, \tag{3}$$

$$L_V = \frac{1}{9N_V} \sum_{n=1}^{N_V} \left\|\mathbf{V}_n^{\mathrm{NN}} - \mathbf{V}_n^{\mathrm{DFT}}\right\|^2. \tag{4}$$

The factors $1/3$ and $1/9$ arise from averaging the force and virial errors over their three and nine Cartesian components, respectively, ensuring consistent normalization across all contributions to the loss.

During inference, atomic coordinates are transformed into symmetry-preserving descriptors that are invariant under translation, rotation, and permutation of atoms of the same type [10, 11]. These descriptors serve as input to the neural network, which predicts atomic energies and corresponding forces in real time, achieving *ab initio*-level accuracy at computational cost comparable to classical MD.

A variety of neural-network-based interatomic potentials have been proposed, including Behler–Parrinello networks [8], Gaussian Approximation Potentials (GAP) [12], SchNet [13], and the E(3)-equivariant NequIP model [14]. Among these, the Deep Potential Molecular Dynamics (DeepMD) framework has emerged as one of the most widely adopted and computationally efficient implementations due to its smooth descriptor formulation and native GPU acceleration. The following subsection focuses specifically on **DeepMD-kit version 2**, which forms the theoretical and computational foundation for the present work.

2.3 Deep Potential Molecular Dynamics (DeepMD)

This work focuses on **Deep Potential Molecular Dynamics version 2 (DeepMD-kit v2)**, the second-generation implementation of the Deep Potential framework. DeepMD-kit v2 represents a major advancement over the original DeepMD formulation [2], introducing improved descriptor smoothness, enhanced multi-species handling, and a refined software architecture that enables highly efficient GPU execution [3]. These features make DeepMD-kit v2 particularly suitable for modeling chemically complex systems such as binary alloys within HALMD.

Deep Potential Molecular Dynamics is a machine-learning approach that approximates interatomic interactions with near *ab initio* accuracy while retaining computational efficiency comparable to classical MD. A neural network is trained on quantum-mechanical reference data (typically from DFT or AIMD) to map atomic configurations to energies and forces. The total potential energy of the system is decomposed into atomic energy contributions [2, 3]:

$$E = \sum_i E_i(\mathcal{R}_i),$$

where E_i is the atomic energy of atom i, and $\mathcal{R}_i$ denotes its local environment within a cutoff radius r_c. This locality assumption yields linear scaling in the number of atoms and enables efficient parallelization on GPUs.

A defining element of DeepMD is the use of symmetry-preserving descriptors that are invariant under translations, rotations, and permutations of atoms of the same type. DeepMD-kit v2 employs the *Smooth Edition* (SE) descriptor family, provided in angular (`se_e2_a`) variants [3]. It is constructed from two matrices:

- the R**-matrix**, containing relative position vectors $R_{ij} = \mathbf{r}_j - \mathbf{r}_i$ for neighbors j within the cutoff radius;
- the G**-matrix**, a normalized representation incorporating angular information and many-body geometric correlations.

Distances $r_{ij} = |R_{ij}|$ are modulated by a smooth cutoff function to ensure continuity of energies and forces at the cutoff boundary [2].

The Deep Potential model in both DP-v1 and DP-v2 consists of two neural networks: an *embedding (filter) network* and a *fitting network* [2, 3].

1. **Embedding (Filter) Network.** This smaller feed-forward network transforms raw neighbor information into high-dimensional filter features. In DP-v2, a distinct embedding network is assigned to each atomic species, allowing the model to capture chemically specific interactions (e.g., Cu–Ag, Ni–Al) while preserving permutation invariance within each species [3]. The embedding network processes each neighbor independently.
2. **Fitting Network.** The outputs of the embedding networks are aggregated into a descriptor vector $\mathbf{D}_i$, which is passed to a larger fully connected fitting network that predicts the atomic energy E_i. Each species has its own fitting network parameters, enabling accurate modeling of multi-component materials.

Forces are obtained as the negative gradients of the total energy:

$$\mathbf{F}_i = -\frac{\partial E}{\partial \mathbf{r}_i}.$$

During training, these derivatives are computed automatically through TensorFlow's back-propagation engine. During inference—which is the focus of this thesis—the gradients are obtained by applying the chain rule explicitly through the descriptor and neural-network layers, following the DeePot-v2 computational graph [3].

DeepMD-kit v2 introduces several enhancements that are essential for accurate multi-species modeling. These include species-dependent embedding networks, species-specific filter weights, descriptor normalization layers, and refined smooth cutoff schemes [3]. Together, these improvements enable DeePot-v2 to handle chemically diverse systems with greater smoothness, transferability, and numerical stability compared to its predecessor.

A trained DeepMD model is stored as a TensorFlow *frozen_model.pb* file trained with config file `input.json` file which specifies descriptor types, cutoff radii, hyperparameters, and precision settings. During inference, atomic coordinates are transformed into descriptors, processed by species-specific embedding networks, and fed into the fitting network to produce atomic energies and forces.

In this thesis, DeepMD-kit v2 serves as the machine-learned potential that is fully integrated into HALMD. All parameter extraction, descriptor reconstruction, and neural-network inference follow the DP-v2 specification precisely. By embedding this inference process into HALMD's GPU-accelerated architecture, the present work enables large-scale simulations of multi-species systems, such as binary alloys, with near *ab initio* accuracy.

3 Methodology

3.1 Overview of Implementation

The integration of DeepMD-v2 into HALMD follows a modular workflow in which each stage provides the necessary inputs for the next. The process begins with extracting all descriptor and neural-network parameters from the `frozen_model.pb` and `input.json` files and converting them into a compact HDF5 format that HALMD can load efficiently. Using these parameters, HALMD reconstructs the local atomic environment in the exact DeepMD-v2 convention, including periodic wrapping, ghost-cell expansion, and species-ordered neighbour selection. From this environment, HALMD computes the descriptor by forming the geometric matrix R, evaluating species-dependent embedding networks to obtain G, and assembling the final descriptor vector D_i that characterises the environment of each atom. The descriptor is then passed through the species-specific fitting network to compute atomic energies, while forces are obtained by propagating derivatives through the descriptor and networks using a combination of analytic expressions and automatic differentiation. Together, these steps enable HALMD to perform full DeepMD-v2 inference—energies and forces—within its native C++/CUDA simulation loop.

3.2 Model Parameter Extraction

3.2.1 Frozen Model Structure

The trained DeepMD-v2 potential is distributed as a frozen TensorFlow graph (`frozen_model.pb`). To perform manual inference inside HALMD, all relevant model parameters must be re-

constructed from the computation graph. The node list extracted from the model (see Appendix X) contains all information required for this reconstruction.

Descriptor attributes. The descriptor section of the graph exposes the physical and structural parameters needed to build the environment matrix:

- cutoff radii (`descrpt_attr/rcut`, `descrpt_attr/rcut_smth`),
- neighbour selection size (`descrpt_attr/sel`, `descrpt_attr/original_sel`),
- type mapping and normalization tensors (`descrpt_attr/t_avg`, `descrpt_attr/t_std`, `descrpt_attr/ntypes`).

These values uniquely determine the construction of $\hat{R}_i$ and ensure that HALMD reproduces the same geometric preprocessing as DeepMD.

Embedding network. The embedding (filter) network parameters appear under `filter_type_0/*`. The suffix `_0` indicates that these parameters correspond to a central atom of species $a = 0$. DeepMD-v2 stores a separate filter (embedding) network for each central-atom species, so in a multi-species system one would observe additional blocks such as `filter_type_1/*`, `filter_type_2/*`, and so on.

Within each `filter_type_a` group, DeepMD-v2 uses node names that follow the pattern

$$\texttt{matrix\_}\ell\texttt{\_}b, \qquad \texttt{bias\_}\ell\texttt{\_}b,$$

where:

- ℓ is the layer index of the embedding MLP,
- b is the *neighbour-species index*.

Thus, nodes such as `matrix_1_0`, `matrix_2_0`, `matrix_3_0` and their corresponding `bias_1_0`, `bias_2_0`, `bias_3_0` represent the three-layer embedding network used when:

$$\text{central species } a = 0, \qquad \text{neighbour species } b = 0.$$

In a multi-species model, additional blocks would appear, for example:

- `matrix_1_1`, `matrix_2_1`, ... for neighbours of species $b = 1$,
- `matrix_1_2`, `matrix_2_2`, ... for neighbours of species $b = 2$, etc.

Each embedding layer provides:

- a weight matrix (`matrix`$\_\ell\_b$),
- a bias vector (`bias`$\_\ell\_b$),
- a nonlinear activation function node (typically `Tanh`).

Collectively, these nodes define the mapping

$$\hat{s}_{ij} \mapsto G_{ij},$$

which transforms the normalized inverse distance into an embedding vector for the descriptor.

Descriptor production. The node `ProdEnvMatA` encapsulates the internal DeepMD operation that produces the environment matrix and its derivatives:

- relative positions and angular components (`o_rmat`),
- pairwise distances (`o_rij`),
- neighbour list (`o_nlist`),
- geometric derivatives (`o_rmat_deriv`).

These outputs provide the ground truth for validating HALMD's geometric derivative implementation.

Fitting network. The atomic-energy fitting network appears under the node groups `layer_0_type_0/*`, `layer_1_type_0/*`, `layer_2_type_0/*`, and `final_layer_type_0/*`. Each layer contributes:

- a weight matrix,
- a bias vector,
- optional residual-timestep coefficients (`idt` nodes),
- a nonlinear activation function (`Tanh`).

Together, these nodes define the mapping from the descriptor D_i to the atomic energy E_i.

Energy and force outputs. The frozen graph also includes nodes providing:

- total energy (`o_energy`),
- per-atom energy (`o_atom_energy`),
- forces (`o_force`),
- virials (`o_virial`, `o_atom_virial`),

as well as all intermediate gradient nodes under `gradients/*`, used by DeepMD's automatic differentiation engine.

Together, these extracted components provide a complete specification of the trained potential: descriptor parameters, embedding transformations, environment-matrix construction, fitting-network architecture, and the final energy/force outputs. Using these elements, HALMD can reproduce DeepMD-v2 inference exactly, without relying on TensorFlow.

3.2.2 Extraction Procedure

DeepMD-kit stores trained neural network potentials as a TensorFlow `frozen_model.pb` file, accompanied by an `input.json` file containing model hyperparameters. The `.pb` file encodes all numerical weights, biases, and auxiliary tensors in the form of TensorFlow computation graph nodes, while the JSON file specifies the descriptor type, cutoff radius, number of neurons per layer, activation functions, and species layout. Following the methodology of DeepMD-kit v1 and v2 [2, 3], this thesis extracts all necessary model

parameters from these files and converts them into an HDF5 representation suitable for efficient inference inside HALMD.

The extraction process begins by loading the TensorFlow graph definition from the `.pb` file. All tensors of type `Const` are scanned, and those whose node names match descriptor or fitting-network layers are decoded using TensorFlow's low-level `tensor_util.MakeNdarray`. The DeepMD model contains one set of parameters per atomic species, and the species ordering in the output strictly follows the ordering defined in the DeepMD input configuration. For the work presented here, the descriptor type is always the angular Smooth Edition descriptor `se_e2_a`, which DeepMD-kit v2 uses for multi-species models requiring both radial and angular correlation encoding.

Descriptor Parameters. For each species, the descriptor section consists of a stack of fully connected layers with user-specified neuron counts, activation functions, and optional residual time-step connections. The extraction script identifies each descriptor layer via a naming pattern of the form

$$\texttt{filter\_type\_}\langle s\rangle\texttt{/matrix\_}\langle k\rangle\texttt{\_}\langle t_n\rangle, \qquad \texttt{filter\_type\_}\langle s\rangle\texttt{/bias\_}\langle k\rangle\texttt{\_}\langle t_n\rangle,$$

where s indexes the species, k is the layer index, and t_n enumerates neighbor-type channels. For each layer, the script records:

- weight matrices,
- bias vectors,
- number of neurons,
- activation function (typically `tanh` or `linear` in the present work),
- the presence of a residual network branch.

DeepMD-v2 allows descriptor layers to use residual updates when `resnet_dt=true`. In that case, a per-layer time-step tensor is extracted and marked in the output structure, though the use of residual updates depends on the model's training configuration.

Embedding (Filter) Network. In the DeepMD architecture, the descriptor network is conceptually equivalent to the "filter" or embedding network described in [2, 3]. Each species has its own filter-network parameters to account for species-dependent geometric correlations. The extracted filter-network weights are reshaped into a row-major layout compatible with HALMD's GPU evaluation kernels.

Fitting Network. For each species, the fitting network (also called the main network) predicts the atomic energy contribution E_i. The extraction process retrieves:

- all intermediate fitting layers,
- species-dependent activation functions,
- weight and bias tensors for each layer,
- the species-specific atomic energy bias term `bias_atom_e`,
- the parameters of the final output layer.

The fitting-network layers are identified using a template of the form

$$\texttt{layer\_LL\_type\_} < k >_< s > /\texttt{matrix}, \qquad \texttt{layer\_LL\_type\_} < k >_< s > /\texttt{bias},$$

and similarly for the final layer `final_layer_type_<s>`. The final layer uses a fixed activation function, typically `linear`, consistent with the DeepMD energy formulation.

Normalization Parameters. DeepMD-v2 introduces descriptor normalization tensors `t_avg` and `t_std`, which are required to maintain numerical stability and descriptor smoothness. These tensors are extracted from the nodes

$$\texttt{descrpt\_attr/t\_avg}, \qquad \texttt{descrpt\_attr/t\_std},$$

and stored in the HDF5 output so that HALMD can apply the same normalization as the original DeepMD model.

Organization and Output Format. All extracted parameters are written into a structured HDF5 file using a hierarchical layout:

- global descriptor constants (cutoff radii, smoothing radii, normalization tensors),
- per-species descriptor network parameters,
- per-species fitting network parameters.

This structure mirrors the multi-species design of DeepMD-kit v2 and makes the extracted parameters directly usable by HALMD for inference. Unlike the single-species extraction used in the previous HALMD implementation [1], the present work extracts and stores fully species-resolved descriptor and fitting-network data, which are required for accurate multi-species DeepMD-v2 inference.

The resulting HDF5 file serves as the unified model representation for HALMD. During the simulation, HALMD loads this file, reconstructs the descriptor and neural networks using GPU-optimized data structures, and performs inference without relying on TensorFlow. This enables the Deep Potential model to be evaluated natively within HALMD's simulation loop with high performance and full multi-species support.

3.3 Coordinate System Extension

A central contribution of this thesis is the redesign of HALMD's environment–construction pipeline so that it follows the exact conventions required by DeepMD-v2. The earlier implementation by Cruz [1] relied on HALMD's built-in periodic boundary handling and neighbour list, which correctly satisfy the minimum-image convention used in classical MD [4]. However, this approach provides only the minimal displacement between atoms and does not reproduce the full periodic environment expected by the Deep Potential (DP) descriptor pipeline. DeepMD-v2 requires explicit coordinate wrapping, ghost-cell expansion, species-aware neighbour grouping, and descriptor normalization [2, 3]. These steps are essential for reproducing the descriptor inputs used during training.

The new DeePMD-style environment constructor introduced in this work consists of the following components.

3.3.1 Explicit coordinate wrapping

DeepMD requires that all atomic coordinates are expressed inside the primary simulation cell before any descriptor quantities are computed. This is stricter than the minimum-image convention normally used in classical molecular dynamics, where only the *differences* between coordinates need to be mapped back into the simulation box. DeepMD, however, expects the *absolute coordinates* of every atom to lie within the interval

$$[0,\, L_\alpha) \qquad \text{for each Cartesian direction } \alpha \in \{x, y, z\}.$$

To ensure consistency, each coordinate component is mapped explicitly into the primary simulation box using

$$\tilde{r}_{i\alpha} = r_{i\alpha} - L_\alpha \left\lfloor \frac{r_{i\alpha}}{L_\alpha} \right\rfloor .$$

This operation performs the following steps:

- Compute the integer number of box lengths by which the particle has moved:

  $$n_\alpha = \left\lfloor \frac{r_{i\alpha}}{L_\alpha} \right\rfloor .$$

  This may be positive (particle drifted to the right), negative (particle drifted to the left), or zero.

- Subtract exactly $n_\alpha L_\alpha$ from the coordinate so that the result lies in the interval $[0, L_\alpha)$:

  $$\tilde{r}_{i\alpha} = r_{i\alpha} - n_\alpha L_\alpha .$$

- The wrapped coordinates $\tilde{\mathbf{r}}_i$ now correspond to the representation DeepMD expects as input.

This differs from the minimum-image convention, which only wraps *relative displacements*. DeepMD's descriptor depends on absolute coordinates (through the ghost-cell extension and species grouping), so the wrapping step is necessary to guarantee that the computed descriptor matches the TensorFlow implementation exactly.

3.3.2 Ghost-cell periodic extension

DeepMD constructs atomic environments by explicitly replicating the simulation cell in all spatial directions before evaluating descriptor quantities. This ensures that every atom, including those close to a periodic boundary, retains a complete neighbourhood within the cutoff radius r_c. In contrast, HALMD's native neighbour list returns only the nearest periodic image, which is sufficient for classical MD but does not supply the full set of geometric relations required by the DeepMD descriptor pipeline.

To achieve DeepMD-compatible behaviour, the present work implements full ghost-cell tiling. For each wrapped coordinate $\tilde{\mathbf{r}}_i$, the periodic images are generated as

$$\mathbf{r}_i^{(\mathbf{s})} = \tilde{\mathbf{r}}_i + s_x L_x\, \mathbf{e}_x + s_y L_y\, \mathbf{e}_y + s_z L_z\, \mathbf{e}_z, \qquad s_\alpha \in [-n_{\mathrm{buff},\alpha},\, n_{\mathrm{buff},\alpha}],$$

where L_α are the box lengths and

$$n_{\mathrm{buff},\alpha} = \left\lceil \frac{r_c}{L_\alpha} \right\rceil$$

ensures full spatial coverage of the cutoff sphere. For a cubic cell with $n_{\mathrm{buff}} = 1$, this produces $3^3 = 27$ images (the central box and its neighbouring tiles). Neighbour search is then performed over these images, and only the atoms whose replicas fall within the cutoff are retained.

This explicit tiling is necessary for descriptor correctness: it guarantees that all physically relevant neighbours are included, that the distances r_{ij} and direction vectors $\mathbf{r}_{ij}$ match those used by DeepMD-kit, and that atoms near cell boundaries obtain descriptor rows identical to those of atoms in the interior. By reproducing DeepMD's environment construction exactly, the HALMD implementation achieves full compatibility with the DeepMD-v2 descriptor generation process.

3.3.3 Environment-matrix construction interface

Previously, HALMD computed environment quantities directly from the neighbour list using minimum-image displacements. The new implementation builds environments from:

- wrapped coordinates,
- ghost-expanded coordinates,
- species-ordered neighbour lists.

The construction of the descriptor matrices (R, G) occurs later and is documented in Section 3.4.

3.3.4 Normalization using $t_{\mathrm{avg}}, t_{\mathrm{std}}$

DeepMD-v2 normalizes descriptor features using training-time statistics:

$$X' = \frac{X - t_{\mathrm{avg}}}{t_{\mathrm{std}}}.$$

The previous HALMD implementation lacked this step; the new pipeline integrates normalization directly, ensuring full compatibility with DeepMD-v2.

3.3.5 Summary of improvements

The updated environment-construction step introduces:

- explicit DeePMD-style coordinate wrapping,
- full periodic ghost-cell expansion,
- species-aware neighbour grouping based on the `sel` configuration,
- a redesigned environment interface for descriptor construction,
- descriptor normalization matching DeepMD-v2.

These extensions form the necessary foundation for computing the R and G descriptor matrices (Section 3.4) and enable HALMD to accurately evaluate multi-species DeepMD-v2 models.

3.3.6 Species-aware neighbour grouping

DeepMD-v2 requires that neighbour atoms be organised in a very specific species-dependent format before any descriptor quantities are computed. This is fundamentally different from HALMD's native neighbour list, which treats all neighbours uniformly and returns a single distance-sorted list independent of species. However, in DeepMD-v2 the descriptor is constructed under the assumption that each central atom has a fixed, preallocated number of neighbour slots for *each species type*, as defined by the model parameter `sel`:

$$\texttt{sel} = \big[N_c^{(1)},\, N_c^{(2)},\, \ldots,\, N_c^{(B)}\big],$$

where $N_c^{(b)}$ is the number of neighbours of species b that the model expects for each central atom.

This fixed layout is not optional: every block of neighbours associated with a neighbour species is passed through a species-specific embedding network $N_\theta^{(a,b)}$, and thus the descriptor depends critically on *which neighbour occupies which row* of the R and G matrices.

Constructing species-specific neighbour lists. To reproduce this behaviour, the new implementation replaces HALMD's species-blind neighbour structure with species-partitioned lists. For a central atom i and each species label a, HALMD now constructs

$$\text{neighbors\_by\_type}[a] = \big\{\, j \;\big|\; t_j = a,\; r_{ij} < r_c \,\big\},$$

where t_j is the species of neighbour j. Each such list contains only the neighbours belonging to species a.

Distance sorting within each species group. DeepMD requires each species block to be internally sorted by distance. Thus, for every species a, the list neighbors_by_type$[a]$ is sorted according to r_{ij}:

$$\text{sort}\big(\text{neighbors\_by\_type}[a],\; \text{key} = r_{ij}\big).$$

This ensures that:

- the nearest neighbours of each species appear first,
- rows of the descriptor matrices corresponding to species a match the exact order used by DeepMD-kit,
- neighbour-dependent neural networks receive inputs in the correct, deterministic order.

3.3.7 Applying the `sel` constraint

For each central atom, DeepMD-v2 requires that the descriptor contains a fixed number of neighbour entries for each species, as prescribed by the vector

$$\texttt{sel} = \big[N_c^{(0)},\, N_c^{(1)},\, \ldots\big].$$

After grouping neighbours by species and sorting them by distance, each species block is adjusted to have exactly $N_c^{(a)}$ rows:

$$\widehat{\text{neighbors\_by\_type}}[a] = \begin{cases} \text{first } N_c^{(a)} \text{ neighbours}, & \text{if more than } N_c^{(a)} \text{ are available,} \\ \text{zero-padded rows}, & \text{if fewer than } N_c^{(a)} \text{ neighbours exist.} \end{cases}$$

This behaviour matches the DeepMD-v2 implementation, where missing neighbours are represented by zero-valued geometric entries in the environment matrix. Invalid neighbour indices are masked, resulting in

$$R_{ij} = 0, \qquad \hat{s}_{ij} = 0, \qquad G_{ij} = N_\theta^{(a,b)}(0),$$

for all padded rows. No duplication of the last valid neighbour is performed.

Zero-padding ensures that the assembled R and G matrices have the fixed, model-dependent shape required by DeepMD-v2 and that the fitting network receives inputs consistent with the TensorFlow/JAX implementation.

Constructing the full neighbour ordering. The final neighbour list for descriptor construction is obtained by concatenating species blocks in the fixed species order used by the model:

$$\text{neighbors\_ordered} = \left[\widehat{\text{neighbors\_by\_type}}[1],\ \widehat{\text{neighbors\_by\_type}}[2],\ \ldots,\ \widehat{\text{neighbors\_by\_type}}[B]\right].$$

Every descriptor row in the matrices R and G now corresponds to a specific species and a specific position within that species block, exactly as expected by the DeepMD-v2 architecture.

Importance of species grouping. This organisation is not merely a convenience; it is required because:

- each species pair (a, b) uses a different embedding network $N_\theta^{(a,b)}$,
- descriptor rows must map one-to-one to embedding-network evaluations,
- the quadratic descriptor D_i implicitly assumes this grouping when computing $G^{\mathrm{T}}\hat{R}$,
- the fitting network is trained on descriptors in this exact ordering.

A deviation from this layout would result in incorrect R and G matrices, leading to mismatched descriptors D_i, incorrect predicted energies, and incorrect forces.

Thus, the introduction of species-aware neighbour grouping is a key requirement for enabling HALMD to evaluate multi-species DeepMD-v2 models correctly.

3.4 Construction of the Geometric Matrix R

Once wrapped coordinates, ghost-cell expansions, and species-ordered neighbour lists have been obtained (Section 3.3), the next task is to construct the geometric matrix R. This matrix encodes the local atomic environment of a central atom i in a way that is translationally, rotationally, and permutation invariant [2, 3]. Each of the $N_c^{(\text{total})}$ neighbour slots contributes one row of four geometric quantities.

3.4.1 Relative displacement and inverse-distance quantities

For each neighbour selected into the fixed-capacity list, the relative displacement is computed using the *wrapped and ghost-extended coordinates*:

$$\mathbf{r}_{ij} = \mathbf{r}_j^{(\mathbf{s})} - \tilde{\mathbf{r}}_i \quad \in \mathbb{R}^3, \qquad r_{ij} = \|\mathbf{r}_{ij}\| \in \mathbb{R}.$$

3.4.2 Raw geometric row

DeepMD defines the raw geometric features of neighbour j (row index k) as

$$R_k^{\mathrm{raw}} = \left[s_{ij},\ \frac{x_{ij}}{r_{ij}^2},\ \frac{y_{ij}}{r_{ij}^2},\ \frac{z_{ij}}{r_{ij}^2} \right] \quad \in \mathbb{R}^4, \qquad s_{ij} = \frac{1}{r_{ij}}.$$

Collecting all rows yields the matrix

$$R^{\mathrm{raw}} \in \mathbb{R}^{N_c^{(\mathrm{total})} \times 4}.$$

3.4.3 Cutoff behaviour and switching function

To ensure smooth decay near the cutoff radius, DeepMD applies a quintic switching function:

$$f_{\mathrm{sw}}(r) = \begin{cases} 1, & r \le r_{\mathrm{sm}}, \\ x^3(-6x^2 + 15x - 10) + 1, & r_{\mathrm{sm}} < r < r_c, \\ 0, & r \ge r_c, \end{cases} \qquad x = \frac{r - r_{\mathrm{sm}}}{r_c - r_{\mathrm{sm}}}.$$

The resulting weight vector is

$$w_k = f_{\mathrm{sw}}(r_{ij(k)}), \qquad w \in \mathbb{R}^{N_c^{(\mathrm{total})}}.$$

Applying this weight yields the weighted geometric matrix

$$R^{\mathrm{w}}_{k,\alpha} = R^{\mathrm{raw}}_{k,\alpha}\, w_k, \qquad R^{\mathrm{w}} \in \mathbb{R}^{N_c^{(\mathrm{total})} \times 4}.$$

3.4.4 Species-aware neighbour ordering

DeepMD-v2 requires neighbours to be arranged according to:

1. neighbour species,
2. increasing neighbour distance within each species,
3. a fixed per-species capacity determined by

$$\texttt{sel} = [N_c^{(0)},\ N_c^{(1)}, \dots].$$

After this procedure, every central atom receives a geometric matrix with the same shape:

$$R^{\mathrm{w}} \in \mathbb{R}^{N_c^{(\mathrm{total})} \times 4}.$$

3.4.5 Normalization of geometric features

DeepMD-v2 applies feature-wise normalization using tensors

$$t_{\text{avg}},\ t_{\text{std}} \in \mathbb{R}^{N_c^{(\text{total})} \times 4}.$$

The normalized geometric matrix is

$$\hat{R}_{k,\alpha} = \frac{R^{\text{w}}_{k,\alpha} - (t_{\text{avg}})_{k,\alpha}}{(t_{\text{std}})_{k,\alpha}}, \qquad \hat{R} \in \mathbb{R}^{N_c^{(\text{total})} \times 4}.$$

The first column,

$$\hat{s}_{ij} = \hat{R}_{ij,0},$$

serves as the scalar input to the embedding network.

3.4.6 Summary

The constructed matrix

$$\hat{R} \in \mathbb{R}^{N_c^{(\text{total})} \times 4}$$

is bitwise compatible with DeepMD-v2 and provides the geometric inputs for the next stage of the descriptor.

3.5 Construction of the Embedding Matrix G

The second stage of the DeepPot-SE descriptor transforms each normalized inverse distance $\hat{s}_{ij} = \hat{R}_{ij,0}$ into an embedding vector via species-pair–specific neural networks. Stacking these embeddings yields

$$G \in \mathbb{R}^{N_c^{(\text{total})} \times M},$$

where M is the embedding dimension defined in the DeepMD model.

3.5.1 Input and output dimensionality

Each embedding network receives a scalar input:

$$\hat{s}_{ij} \in \mathbb{R},$$

and produces a vector:

$$G_{ij} \in \mathbb{R}^M.$$

Collecting all neighbour embeddings produces the matrix

$$G = \begin{bmatrix} G_{i1}^T \\ G_{i2}^T \\ \vdots \\ G_{iN_c^{(\text{total})}} \end{bmatrix} \in \mathbb{R}^{N_c^{(\text{total})} \times M}.$$

3.5.2 Species-pair–dependent embedding networks

DeepMD-v2 specifies one embedding network per central–neighbour species pair:

$$G_{ij} = N_\theta^{(a,b)}(\hat{s}_{ij}),$$

where:

- a is the central atom species,
- b is the neighbour species for row j,
- $N_\theta^{(a,b)}$ is a dedicated MLP with its own parameters.

The HALMD implementation stores the full set of networks:

$$\texttt{neural\_networks}[a][b] \equiv N_\theta^{(a,b)}.$$

3.5.3 Runtime network selection

Since neighbour ordering already groups neighbours by species, the embedding for row j is evaluated as

$$G_{ij} = N_\theta^{(a,b_{(j)})}(\hat{s}_{ij})\,,$$

ensuring that rows of G align exactly with rows of $\hat{R}$.

3.5.4 Summary

The embedding construction stage:

- maps each scalar $\hat{s}_{ij}$ to an M-dimensional vector,
- evaluates species-pair–dependent networks,
- selects the correct network dynamically for each neighbour,
- produces a consistently ordered matrix $G \in \mathbb{R}^{N_\mathrm{c}^{(\mathrm{total})} \times M}$.

Together, the matrices $\hat{R}$ and G provide the full input required for computing the quadratic descriptor D_i in the next stage.

3.6 Descriptor Computation

The DeepPot-SE (`se_e2_a`) descriptor used in DeepMD-v2 represents the local atomic environment of each atom through a structured combination of geometric features and species-aware embedding functions. For a central atom i, the descriptor is constructed from two matrices: the normalized geometric matrix $\hat{R}_i$ and the embedding matrix G_i. These matrices are subsequently combined into a quadratic descriptor D_i, which serves as the input to the species-dependent fitting network.

DeepMD-v2 allocates neighbour slots on a per-species basis. Let $\mathcal{S}$ denote the set of all atomic species, and let $N_c^{(b)}$ denote the number of neighbour slots assigned to species $b \in \mathcal{S}$ as defined in the model's `sel` vector. The total neighbour capacity for a central atom of species a is therefore

$$N_c^{(\mathrm{total})} = \sum_{b \in \mathcal{S}} N_c^{(b)}. \tag{4.1}$$

All descriptor matrices constructed in this work use this multi-species capacity.

3.6.1 Normalized geometric matrix

Following the coordinate preprocessing steps of Section 3.3 (wrapping, ghost-cell extension, species grouping, and cutoff filtering), the geometric matrix for atom i is defined as

$$\hat{R}_i \in \mathbb{R}^{N_c^{(\mathrm{total})} \times 4}, \tag{4.2}$$

where each row corresponds to one neighbour slot, ordered by species blocks. The four columns consist of:

- the normalized inverse distance $\hat{s}_{ij}$, and
- the three normalized angular components.

The normalization is performed element-wise using the tensors t_{avg} and t_{std} extracted from the model:

$$\hat{R}_{ij,\alpha} = \frac{R_{ij,\alpha} - (t_{\mathrm{avg}})_\alpha}{(t_{\mathrm{std}})_\alpha}, \qquad \alpha \in \{0,1,2,3\}. \tag{4.3}$$

The first column,

$$\hat{s}_{ij} = \hat{R}_{ij,0}, \tag{4.4}$$

serves as the scalar input to the embedding networks.

3.6.2 Embedding matrix

For each neighbour slot j, with neighbour species $b(j)$, the embedding vector is computed by evaluating the corresponding species-pair network:

$$G_{ij} = N_\theta^{(a,b(j))}\big(\hat{s}_{ij}\big), \tag{4.5}$$

where a denotes the central-atom species. Each network $N_\theta^{(a,b)}$ maps a scalar input to an embedding vector of dimension M. Thus the embedding matrix has the form

$$G_i \in \mathbb{R}^{N_c^{(\mathrm{total})} \times M}, \tag{4.6}$$

and its rows follow the same species-block structure as $\hat{R}_i$. Only the first M' columns (the "truncated embedding channels") are used in the descriptor construction; hence we also define

$$G_i^{\mathrm{trunc}} = G_i[:,\, 1\!:\!M'] \in \mathbb{R}^{N_c^{(\mathrm{total})} \times M'}. \tag{4.7}$$

3.6.3 Quadratic descriptor construction

The DeepMD-v2 descriptor for atom i is defined as the scaled quadratic form

$$D_i = \frac{1}{(N_c^{(\mathrm{total})})^2} \left(G_i^{\mathsf{T}}\, \hat{R}_i\right)\left(\hat{R}_i^{\mathsf{T}}\, G_i^{\mathrm{trunc}}\right), \tag{4.8}$$

and therefore satisfies the dimensional relations

$$\begin{aligned} G_i^{\mathsf{T}}\, \hat{R}_i &\in \mathbb{R}^{M\times 4}, \\ \hat{R}_i^{\mathsf{T}} G_i^{\mathrm{trunc}} &\in \mathbb{R}^{4\times M'}, \\ D_i &\in \mathbb{R}^{M\times M'}. \end{aligned} \tag{4.9}$$

Finally, the descriptor matrix is flattened in row-major order into a vector

$$\mathbf{D}_i \;\in\; \mathbb{R}^{MM'}, \tag{4.10}$$

which forms the input to the species-dependent fitting network used to compute the atomic energy contribution E_i.

3.6.4 Summary of improvements

The descriptor construction in this work reproduces the DeepMD-v2 formulation exactly. In contrast to earlier HALMD implementations, the present approach:

- respects the species-resolved neighbour capacities $N_c^{(b)}$,
- evaluates species-pair embedding networks $N_\theta^{(a,b)}$,
- uses the correct DeepMD normalization $(N_c^{(\mathrm{total})})^2$,
- maintains the strict ordering required for descriptor consistency,
- and yields descriptor vectors $\mathbf{D}_i$ matching the TensorFlow implementation to numerical precision.

This provides dimensionally consistent and numerically accurate descriptor pipeline suitable for high-performance inference inside HALMD.

3.7 Potential Energy Calculation

In the Deep Potential (DeePot) formalism, the total energy of a system is expressed as a sum of atomic energy contributions,

$$E = \sum_{i=1}^{N} E_i, \tag{5}$$

where each atomic energy E_i is obtained by evaluating a species-dependent fitting neural network on the descriptor vector $\mathbf{D}_i$. For an atom of species s_i, the mapping is

$$E_i = \mathrm{NN}_{s_i}(\mathbf{D}_i), \tag{6}$$

where NN_{s_i} denotes the fitting network associated with species s_i. HALMD evaluates this network using the trained weights extracted from the `frozen_model.pb` file, including DeepMD-v2 activations, residual connections, and timestep scaling.

3.7.1 Baseline Implementation Prior to This Work

The previous HALMD implementation by Cruz supported:

- DeepMD potentials for single-species systems,
- evaluation of a single fitting network for all atoms,
- full feed-forward inference (activations, skip connections, Jacobian computation),
- exact reproduction of DeepMD energies for monoatomic systems.

However, it lacked the mechanisms required for DeepMD-v2 multi-species inference:

- no species-dependent fitting networks,
- no species-aware descriptor construction ($\hat{R}$, species-partitioned G),
- no per-species atomic energy offset (`bias_atom_e`),
- no coupling between multi-species descriptors and the appropriate fitting network.

As a result, the original HALMD implementation could not reproduce DeepMD-v2 outputs for alloy systems.

3.7.2 Extension: Species-Dependent Fitting Networks

DeepMD-v2 defines one fitting network for each atomic species. The revised HALMD implementation stores the collection

$$\{\mathrm{NN}_0,\ \mathrm{NN}_1,\ \ldots,\ \mathrm{NN}_{S-1}\},$$

and selects the appropriate network according to the species of atom i:

$$E_i = \mathrm{NN}_{s_i}(\mathbf{D}_i).$$

This required extending the HALMD operator so that species information is passed consistently through the descriptor and energy-evaluation stages.

3.7.3 Extension: Per-Species Energy Offsets

DeepMD-v2 models may include constant per-species energy shifts, stored as `bias_atom_e`. HALMD now applies these offsets explicitly:

$$E_i = \mathrm{NN}_{s_i}(\mathbf{D}_i) + b_{s_i}, \tag{7}$$

ensuring numerical agreement with the TensorFlow reference implementation.

3.7.4 Extension: Integration with Multi-Species Descriptors

The descriptor pipeline constructed in Section 3.4 provides the correct DeepMD-v2 descriptor vector $\mathbf{D}_i$. This includes:

- the normalized geometric matrix $\hat{R}$,
- neighbour rows ordered and grouped by species according to `sel`,
- species-dependent embedding networks $G^{(a,b)}$,
- normalization tensors t_{avg} and t_{std}.

The resulting descriptor has the correct shape and ordering required by the species-specific fitting network NN_{s_i}.

3.7.5 Extension: Reproduction of DeepMD-v2 Energies

With all components integrated, HALMD now reproduces DeepMD-v2 energy predictions with floating-point accuracy:

- per-atom energies match the TensorFlow evaluation,
- per-species energy offsets are applied correctly,
- descriptor-to-energy mappings follow the DeepMD-v2 computational graph,
- multi-species (alloy) energies match DeepMD's output exactly.

3.7.6 Final Formulation

The atomic energy computed in HALMD now has the DeepMD-v2 form

$$E_i = \mathrm{NN}_{s_i}(\mathbf{D}_i) + b_{s_i}, \tag{8}$$

and the total potential energy is given by

$$E_{\mathrm{tot}} = \sum_{i=1}^{N} E_i. \tag{9}$$

This formulation exactly matches the TensorFlow-based DeepMD-v2 inference for all tested single- and multi-species models.

4 Force Computation

In the Deep Potential (DeeP) framework, forces are obtained as the negative gradient of the total potential energy with respect to atomic coordinates,

$$\mathbf{F}_i = -\frac{\partial E}{\partial \mathbf{r}_i}.$$

Because the energy is produced through a multi-stage mapping involving geometric quantities (R), species-dependent embedding networks (G), and a species-dependent fitting network, the force computation requires evaluating a sequence of nested derivatives.

DeepMD expresses this as a structured chain rule passing through:

$$\mathbf{r} \xrightarrow{\text{geometry}} R \xrightarrow{\text{embedding}} G \xrightarrow{\text{fitting network}} E,$$

so that each force component involves contributions from all atoms appearing in the local environment of the corresponding descriptor.

The remainder of this section describes the derivative computation in HALMD. We begin with an overview of automatic differentiation (AD), which is used to evaluate all neural-network derivatives. We then detail the analytic derivatives of the geometry matrix R, the descriptor D, and finally show how these quantities are combined with the fitting-network derivatives to assemble the total atomic forces.

4.1 Automatic Differentiation (AD)

As outlined in the beginning of this section, the computation of atomic forces in DeepMD requires propagating derivatives through a sequence of mappings involving geometric transformations, embedding networks, descriptor algebra, and finally a species-dependent fitting network. While the geometric parts admit closed-form analytic expressions, the neural networks used in DeepMD-v2 contain nonlinear activations, residual connections, and timestep scalings that make analytical differentiation impractical. For these components HALMD employs *automatic differentiation* (AD).

Automatic differentiation is a computational technique for evaluating derivatives of functions represented as compositions of elementary operations [15, 16]. Unlike symbolic differentiation, AD avoids expression explosion, and unlike finite differences, it does not introduce truncation error. Instead, derivatives are accumulated by applying the chain rule locally at every primitive operation.

Two modes of AD are central:

- **Forward-mode AD**: propagates derivatives together with the computation, efficient when the number of inputs is small.
- **Reverse-mode AD**: propagates derivatives backward from a scalar output, efficient when the number of outputs is one. This is the mechanism underlying backpropagation [17] and the mode used in modern ML frameworks such as TensorFlow [18] and PyTorch [19].

Reverse-mode AD is particularly well suited to DeepMD, since both the embedding networks $G^{(a,b)}$ and the fitting networks NN_{s_i} have scalar inputs or scalar outputs, making derivative evaluation highly efficient.

HALMD uses Boost.Autodiff [20] to compute all derivatives internal to neural networks:

1. For each neighbour j, the filter network $G^{(a,b)}$ provides both the embedding vector G_{ij} and the derivative $\partial G_{ij}/\partial \hat{s}_{ij}$.
2. For each atom i, the fitting network provides both the predicted atomic energy E_i and the Jacobian $\partial E_i/\partial D_{i,\alpha}$.
3. All remaining derivatives—those involving the geometric mapping $R^*(\mathbf{r})$, the scaled distances $\hat{s}_{ij}$, and the descriptor construction $D(G, R^*)$—are evaluated analytically and implemented explicitly in HALMD.

AD therefore supplies exactly the neural-network parts of the chain rule, while the analytical components handle the geometric and descriptor terms. This hybrid strategy achieves:

- efficient multi-species derivative propagation,
- clean separation between neural and geometric derivatives.

The following subsections detail each analytic component of the force derivation: derivatives of the geometric mapping (R^*), descriptor derivatives, fitting-network derivatives, and finally the assembly of the total force.

4.2 Derivatives of the Geometry Matrix R

To compute forces, we require the derivative of each row of the geometry matrix R with respect to the coordinates of the central atom i. Each row depends on the neighbour displacement

$$\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i = (d_x, d_y, d_z), \qquad r_{ij} = \|\mathbf{r}_{ij}\|,$$

and on the switched inverse distance

$$\tilde{s}_{ij} = \frac{1}{r_{ij}} f_{\mathrm{sw}}(r_{ij}).$$

We now derive all components of $\partial R_{ij}/\partial \mathbf{r}_i$ using only the multivariate chain rule.

4.2.1 Step 1: Rewrite the Components of R

For neighbour j, the geometry row is

$$R_{ij} = \left[\tilde{s}_{ij},\ \tilde{s}_{ij}\tfrac{d_x}{r_{ij}},\ \tilde{s}_{ij}\tfrac{d_y}{r_{ij}},\ \tilde{s}_{ij}\tfrac{d_z}{r_{ij}}\right].$$

We denote the three directional components compactly as

$$S_\alpha = \tilde{s}_{ij}\,\frac{d_\alpha}{r_{ij}}, \qquad \alpha \in \{x, y, z\}.$$

4.2.2 Step 2: Derivative of the Distance r_{ij}

Following the derivation in Section 3.4, the distance can be written as

$$r_{ij} = (d_x^2 + d_y^2 + d_z^2)^{1/2}.$$

Using the chain rule,

$$\frac{\partial r_{ij}}{\partial d_\alpha} = \frac{d_\alpha}{r_{ij}}, \qquad \alpha \in \{x, y, z\}.$$

Since

$$d_\alpha = r_{j,\alpha} - r_{i,\alpha},$$

we also have

$$\frac{\partial d_\alpha}{\partial \mathbf{r}_i} = -\,\mathbf{e}_\alpha,$$

and combining these yields

$$\frac{\partial r_{ij}}{\partial \mathbf{r}_i} = \sum_\alpha \frac{d_\alpha}{r_{ij}}(-\mathbf{e}_\alpha) = -\,\frac{1}{r_{ij}}(d_x, d_y, d_z) = -\,\hat{\mathbf{r}}_{ij}.$$

4.2.3 Step 3: Derivative of the Switched Inverse Distance

The first component of the geometry row is

$$R_{ij,0} = \tilde{s}_{ij} = \frac{1}{r_{ij}} f_{\mathrm{sw}}(r_{ij}).$$

Differentiating with respect to r_{ij},

$$\frac{\partial \tilde{s}_{ij}}{\partial r_{ij}} = -\frac{1}{r_{ij}^2} f_{\mathrm{sw}}(r_{ij}) + \frac{1}{r_{ij}} \frac{\mathrm{d}f_{\mathrm{sw}}}{\mathrm{d}r}(r_{ij}).$$

Applying the chain rule with $\partial r_{ij}/\partial \mathbf{r}_i = -\hat{\mathbf{r}}_{ij}$,

$$\frac{\partial R_{ij,0}}{\partial \mathbf{r}_i} = \frac{\partial \tilde{s}_{ij}}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial \mathbf{r}_i} = -\frac{\partial \tilde{s}_{ij}}{\partial r_{ij}} \hat{\mathbf{r}}_{ij}.$$

4.2.4 Step 4: Derivative of the Directional Components

Consider a single directional term

$$S_\alpha = \tilde{s}_{ij} \frac{d_\alpha}{r_{ij}}.$$

We differentiate it with respect to $\mathbf{r}_i$ using the product rule. First differentiate with respect to distance:

$$\frac{\partial}{\partial r_{ij}} \left(\frac{d_\alpha}{r_{ij}} \right) = d_\alpha \left(-\frac{1}{r_{ij}^2} \right) = -\frac{d_\alpha}{r_{ij}^2}.$$

Then

$$\frac{\partial S_\alpha}{\partial r_{ij}} = \frac{\partial \tilde{s}_{ij}}{\partial r_{ij}} \frac{d_\alpha}{r_{ij}} - \tilde{s}_{ij} \frac{d_\alpha}{r_{ij}^2}.$$

Next we differentiate S_α with respect to the displacement component d_α:

$$\frac{\partial S_\alpha}{\partial d_\alpha} = \tilde{s}_{ij} \frac{1}{r_{ij}}.$$

Using $\partial d_\alpha/\partial \mathbf{r}_i = -\mathbf{e}_\alpha$, the chain rule gives

$$\frac{\partial S_\alpha}{\partial \mathbf{r}_i} = \frac{\partial S_\alpha}{\partial r_{ij}} \frac{\partial r_{ij}}{\partial \mathbf{r}_i} + \frac{\partial S_\alpha}{\partial d_\alpha} \frac{\partial d_\alpha}{\partial \mathbf{r}_i}.$$

Substituting all expressions:

$$\frac{\partial S_\alpha}{\partial \mathbf{r}_i} = -\left(\frac{\partial \tilde{s}_{ij}}{\partial r_{ij}} \frac{d_\alpha}{r_{ij}} - \tilde{s}_{ij} \frac{d_\alpha}{r_{ij}^2} \right) \hat{\mathbf{r}}_{ij} - \frac{\tilde{s}_{ij}}{r_{ij}} \mathbf{e}_\alpha.$$

This formula directly produces the derivative of $R_{ij,1}, R_{ij,2}, R_{ij,3}$.

4.2.5 Step 5: Normalized Geometry Derivative

DeepMD-v2 uses component-wise normalization:

$$\hat{R}_{ij,\alpha} = \frac{R_{ij,\alpha} - (t_{\text{avg}})_\alpha}{(t_{\text{std}})_\alpha}.$$

Since the normalization parameters are constants,

$$\frac{\partial \hat{R}_{ij,\alpha}}{\partial \mathbf{r}_i} = \frac{1}{(t_{\text{std}})_\alpha} \frac{\partial R_{ij,\alpha}}{\partial \mathbf{r}_i},$$

4.2.6 Step 6: Derivative with Respect to the Neighbour Coordinate $\mathbf{r}_j$

All formulas above were derived for the derivative with respect to the *central* atom i. The derivative with respect to the *neighbour* atom j is obtained immediately from the identity

$$\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i,$$

which implies

$$\frac{\partial d_\alpha}{\partial \mathbf{r}_j} = +\,\mathbf{e}_\alpha, \qquad \text{and} \qquad \frac{\partial d_\alpha}{\partial \mathbf{r}_i} = -\,\mathbf{e}_\alpha.$$

Using the chain rule for the distance r_{ij},

$$\frac{\partial r_{ij}}{\partial \mathbf{r}_j} = \sum_\alpha \frac{\partial r_{ij}}{\partial d_\alpha} \frac{\partial d_\alpha}{\partial \mathbf{r}_j} = \sum_\alpha \frac{d_\alpha}{r_{ij}}\, \mathbf{e}_\alpha = +\,\hat{\mathbf{r}}_{ij},$$

which is exactly the opposite of the central-atom derivative $\partial r_{ij}/\partial \mathbf{r}_i = -\hat{\mathbf{r}}_{ij}$.

As a consequence, every derivative with respect to $\mathbf{r}_j$ is simply the negative of the corresponding derivative with respect to $\mathbf{r}_i$:

$$\frac{\partial R_{ij,\alpha}}{\partial \mathbf{r}_j} = -\,\frac{\partial R_{ij,\alpha}}{\partial \mathbf{r}_i}, \qquad \alpha = 0, 1, 2, 3.$$

This follows because every dependence of R_{ij} on atomic positions appears only through the displacement vector $\mathbf{r}_{ij}$, and exchanging the differentiation variable from $\mathbf{r}_i$ to $\mathbf{r}_j$ reverses the sign of every chain-rule factor:

$$\frac{\partial}{\partial \mathbf{r}_j}(\mathbf{r}_j - \mathbf{r}_i) = +\mathbf{I}, \qquad \frac{\partial}{\partial \mathbf{r}_i}(\mathbf{r}_j - \mathbf{r}_i) = -\mathbf{I}.$$

Thus, once the derivative with respect to the central atom is known, the neighbour-atom derivative requires no further computation.

4.3 Descriptor Derivative

We first define the intermediate matrix

$$C = G^{\mathsf{T}} \hat{R} \in \mathbb{R}^{M \times 4},$$

and the *extended* descriptor

$$D_{\text{ext}} = \frac{1}{N_c^2} CC^{\mathsf{T}} \in \mathbb{R}^{M \times M}.$$

The actual DeepMD descriptor D uses only the first M' columns of D_{ext}:

$$D = D_{\text{ext}}[:, 1\!:\!M'].$$

which corresponds to the matrix gradient

$$\frac{\partial D_{\text{ext}}}{\partial C} = \frac{2}{N_c^2} C.$$

Since D is obtained by discarding columns $M'+1, \ldots, M$, the derivative of D with respect to C is simply the restriction of this gradient to those columns:

$$\frac{\partial D}{\partial C} = \left(\frac{\partial D_{\text{ext}}}{\partial C} \right) [:, 1\!:\!M'] = \left(\frac{2}{N_c^2} C \right) [:, 1\!:\!M'].$$

Finally, using $C = G^{\mathsf{T}} \hat{R}$, we have

$$\frac{\partial C}{\partial \hat{R}} = G^{\mathsf{T}},$$

so that the derivative of the (truncated) descriptor with respect to the normalized geometry is

$$\boxed{\frac{\partial D}{\partial \hat{R}} = \frac{\partial D}{\partial C} \frac{\partial C}{\partial \hat{R}} = \left(\frac{2}{N_c^2} C \right) [:, 1\!:\!M'] G^{\mathsf{T}}.}$$

4.4 Fitting Network Derivative

The final stage of the DeepMD–v2 force pipeline is the differentiation of the species-dependent fitting network that maps the descriptor of atom i to its atomic energy,

$$E_i = NN_{s_i}(D_i),$$

where s_i denotes the species of atom i. To assemble forces, we require the Jacobian of this mapping,

$$\frac{\partial E_i}{\partial D_{i,\alpha}}, \qquad \alpha = 1, \ldots, M',$$

which provides the sensitivity of the atomic energy to each descriptor component.

4.4.1 Neural-network derivative through automatic differentiation

The fitting networks in DeepMD–v2 consist of several fully connected layers with nonlinear activation functions, optional residual connections, and layer-dependent timestep scalings. The resulting composite mapping is highly nonlinear, making an explicit analytical derivation of all partial derivatives both cumbersome and error-prone.

Instead, HALMD evaluates these derivatives using *reverse-mode automatic differentiation* (AD), which computes the Jacobian of a scalar-valued neural-network output with

respect to its input in a single backward sweep. For a fitting network with descriptor input $D_i \in \mathbb{R}^{M'}$, AD returns:

$$\boxed{(J_i)_\alpha = \frac{\partial E_i}{\partial D_{i,\alpha}}} \qquad \text{for } \alpha = 1, \ldots, M'.$$

Inside HALMD this Jacobian is obtained by calling the `complete_result()` method of the neural-network module, which performs the following steps:

1. constructs autodiff variables for each input component $D_{i,\alpha}$,
2. propagates them forward through all network layers,
3. extracts the associated derivatives from the autodiff output.

Thus, for each atom,

$$\boxed{\frac{\partial E_i}{\partial D_i} = J_i}$$

is available directly and exactly, matching the derivative used in the original DeepMD implementation.

4.4.2 Contribution from the descriptor

The derivative of the descriptor D_i with respect to atomic coordinates has already been fully derived in Section 4.3. We recall only the final result, which expresses the descriptor derivative in terms of previously computed quantities:

$$\boxed{\frac{\partial D_i}{\partial \mathbf{r}_k} = \frac{\partial D_i}{\partial \hat{R}_i} : \frac{\partial \hat{R}_i}{\partial \mathbf{r}_k} + \frac{\partial D_i}{\partial G_i} : \frac{\partial G_i}{\partial \mathbf{r}_k}} \tag{4.2}$$

Each term in (4.2) is known from earlier sections:

- the derivative $\partial D_i / \partial \hat{R}_i$ follows from the analytic descriptor formula (Section 4.3),
- the geometric derivative $\partial \hat{R}_i / \partial \mathbf{r}_k$ is given in Section 4.2,
- the embedding-network derivative $\partial G_i / \partial \mathbf{r}_k$ is obtained via automatic differentiation as described in Section 4.1.

Thus, no new derivation is required at this stage: Equation (4.2) serves as the descriptor's contribution to the force, and it directly combines all derivative components computed earlier. This expression is inserted into the final force formula in the next subsection.

5 Results

This chapter presents the numerical results obtained from the HALMD implementation of the DeepMD-v2 descriptor, fitting network, and force derivatives developed in this thesis. The primary goal of the evaluation is to verify that HALMD reproduces the reference DeepMD-v2 behaviour for energies and, as far as possible, for force derivatives. Because the force pipeline is significantly more complex and highly sensitive to the analytical–AD derivative chain, special emphasis is placed on identifying where agreement is achieved and where discrepancies remain.

The tests were conducted using several trained DeepMD-v2 models:

- a monoatomic Cu model (Model A),
- a multi-component high-entropy alloy model (HEA),
- a garnet model,
- a second Cu model (Model B), trained on a different dataset and exhibiting noticeably different tensor statistics.

5.1 Energy Agreement Between DeepMD and HALMD

A key requirement for correctness is the reproduction of per-atom and total energies predicted by DeepMD. Because the HALMD implementation now includes the full descriptor, species-dependent filter networks, fitting networks, and bias terms, the predicted energies should match DeepMD up to floating-point precision.

Tables 1–3 illustrate this agreement for several trained models.

Monoatomic Copper Model (Model A)[21]

Table 1: Energy comparison for the Cu model (placeholder values).

Cells	Atoms	DeepMD Energy (eV)	HALMD Energy (eV)
$1 \times 1 \times 1$	4	-6673.83896419	-6673.84
$2 \times 2 \times 2$	32	-53412.37982893	-53412.4
$3 \times 3 \times 3$	108	-180303.22729927	-180303
$4 \times 4 \times 4$	256	-427425.47627004	-427425

Agreement is exact up to numerical precision (differences on the order of 10^{-XX}).

High-Entropy Alloy (HEA) Model [22]

Table 2: Energy comparison for the HEA model (placeholder values).

Cells	Atoms	DeepMD Energy (eV)	HALMD Energy (eV)
$1 \times 1 \times 1$	XX	-166.84218889	$-XXX.XXXX$
$2 \times 2 \times 2$	XX	-1334.73751114	$-XXXXX.XXXX$
$3 \times 3 \times 3$	XX	-4504.73910011	$-XXXXXX.XXXX$
$4 \times 4 \times 4$	XX	-10677.90008914	$-XXXXXX.XXXX$

Agreement is exact up to numerical precision (differences on the order of 10^{-XX}).

Garnet Model [23]

Table 3: Energy comparison for the garnet model (placeholder values).

Atoms	Species	DeepMD Energy (eV)	HALMD Energy (eV)
XXX	X	$-XXXXX.XXXXX$	$-XXXXX.XXXXX$
XXX	X	$-XXXXXXXX.XXXXX$	$-XXXXXXXX.XXXXX$
XXXX	X	$-XXXXXXXXX.XXXXX$	$-XXXXXXXXX.XXXXX$

Energy agreement again confirms the correctness of the descriptor and network forward pass.

Second Copper Model (Model B)[22]

Table 4: Energy comparison for the alternate Cu model (placeholder values).

Cells	Atoms	DeepMD Energy (eV)	HALMD Energy (eV)
$1 \times 1 \times 1$	4	-7.74430699	-7.7443
$2 \times 2 \times 2$	32	-89.44120168	-89.4412
$3 \times 3 \times 3$	108	-335.96002381	-335.96
$3 \times 3 \times 3$	256	-836.4141709	-836.414

This confirms that HALMD handles different training statistics and variations in `t_avg`, `t_std`, and network weight distributions.

5.2 Force Comparison and Remaining Discrepancy

5.3 GPU Profiling and Performance Analysis

To better understand the computational behaviour of DeepMD-v2, detailed GPU profiling was performed using NVIDIA Nsight Systems and Nsight Compute. All measurements reported in this section correspond specifically to the **second copper model**, evaluated on a representative atomic configuration. The goal of this analysis is to quantify the computational cost of the individual stages of the DeepMD pipeline and to identify performance bottlenecks relevant for future optimisation.

CUDA API–level behaviour (Nsight Systems)

Table 5 summarises the most expensive CUDA API calls. Nsight Systems reports both the percentage of total runtime and the absolute time in seconds (converted from nanoseconds).

The two most salient findings at the API level are:

- **Kernel launch overhead is extremely high**. A large fraction of time is spent in `cudaLaunchKernel` and `cudaDeviceSynchronize`, indicating the presence of many short-lived kernels that require explicit synchronisation.
- **JIT module loading is unusually expensive**. The cost of `cuModuleLoadData` accounts for nearly a third of the CUDA API time, suggesting that TensorFlow repeatedly loads or initialises CUDA modules for DeepMD operations.

Table 5: CUDA API time distribution from Nsight Systems for the second Cu model.

Operation	Time (%)	Total Time (s)
`cudaLaunchKernel`	35.7%	0.0313 s
`cuModuleLoadData`	29.0%	0.0254 s
`cudaDeviceSynchronize`	23.3%	0.0205 s
`cuMemAlloc_v2`	4.2%	0.0037 s
`cudaFree`	3.2%	0.0028 s
`cuMemHostAlloc`	1.3%	0.0011 s
Host–device copies	0.7%	0.00058 s
Other operations	¡1%	¡0.0005 s

GPU kernel–level behaviour (Nsight Compute)

Nsight Compute provides kernel-level measurements. Table 6 lists the dominant GPU kernels together with their relative time shares and absolute execution times.

Table 6: Dominant GPU kernels from Nsight Compute for the second Cu model.

Kernel	Time (%)	Total Time (s)
`volta_sgemm_64x64_tn`	9.7%	0.0078 s
Eigen tensor assignment kernels	8–6%	0.0054–0.0040 s
Neighbour-list construction (`build_nlist`)	6.0%	0.0048 s
Bias kernels (`BiasNHWCKernel`)	5.8%	0.0047 s
Descriptor construction (`compute_env_mat_a`)	4.8%	0.0038 s
Activation functions (`Tanh`)	4.3%	0.0034 s
Sorting kernels (`BlockSortKernel`)	3–4%	0.0029–0.0030 s
Remaining GEMM kernels	2–3%	0.0016–0.0023 s
Other kernels	¡2%	¡0.0015 s

The cost distribution demonstrates that:

- **Dense matrix multiplications (GEMM)** account for roughly 20–25% of GPU time. These operations occur in both the embedding networks and the fitting network.
- **Eigen tensor kernels**, which implement slicing, reshaping, and broadcasting, contribute 15–20% of total GPU time. These operations occur frequently when preparing neural-network inputs.
- **Neighbour-list generation and descriptor construction** (e.g. `build_nlist` and `compute_env_mat_a`) account for about 10% of GPU computation.
- **Activation functions** such as `tanh` introduce a measurable cost.
- The significant kernel launch overhead seen in Nsight Systems is consistent with the GPU-side observation that many kernels are relatively short.

CPU-side overhead from the execution summary

The Nsight execution summary reveals an important additional insight: **most of the end-to-end runtime does not occur on the GPU at all**. The dominant entries are:

- `poll` (45.7%)
- `pthread_cond_wait` (22.9%)
- `futex` (14.4%)

These functions indicate:

1. The CPU spends a large amount of time **waiting for GPU kernels to finish**. This is consistent with the many synchronisation points in the DeepMD graph.
2. TensorFlow incurs substantial **thread scheduling and locking overhead**, because the DeepMD graph consists of a large number of small operators.
3. The majority of the total runtime is therefore **CPU overhead, not GPU computation**.

This explains the discrepancy between:

$$\text{GPU kernel time} \approx 7\text{–}8\ \mathrm{s} \qquad \text{vs.} \qquad \text{end-to-end runtime} \gg 7\text{–}8\ \mathrm{s}.$$

Interpretation and implications

The combined profiling results lead to two main conclusions:

1. **DeepMD-v2 is not dominated by a single computational stage**. Cost is distributed across neighbour-list generation, descriptor computation, neural-network evaluation, tensor reshaping, activations, and GEMM kernels. Improving only one of these components will not dramatically reduce total runtime.
2. **Framework overhead and synchronisation dominate total runtime**. The profiling consistently shows that CPU-side waiting, thread synchronisation, and module loading account for the majority of wall-clock execution time. Reducing these overheads likely requires kernel fusion, batching, or a specialised C++ backend such as HALMD.

These results provide a clear performance baseline for the HALMD DeepMD-v2 integration and identify concrete areas where significant optimisations may be possible in future work.

This establishes a solid foundation for future work on completing DeepMD-v2 force compatibility in HALMD.

6 Discussion

7 Conclusions and Future Work

References

[1] Andres Cruz. “Deep Neural Networks Potentials for Scalable Molecular Dynamics Simulations on Accelerator Hardware”. Master’s Thesis. Freie Universität Berlin, Sept. 2025.

[2] Han Wang, Linfeng Zhang, Jiequn Han, et al. “DeePMD-kit: A deep learning package for many-body potential energy representation and molecular dynamics”. In: *Computer Physics Communications* 228 (2018), pp. 178–184.

[3] Jinzhe Zeng et al. “DeePMD-kit v2: A software package for deep potential models”. In: *The Journal of Chemical Physics* 159.5 (2023).

[4] Peter H Colberg and Felix Höfling. “Highly accelerated simulations of glassy dynamics using GPUs: Caveats on limited floating-point precision”. In: *Computer Physics Communications* 182.5 (2011), pp. 1120–1129.

[5] HALMD Developers. *HALMD: High-Accuracy Large-scale Molecular Dynamics*. https://github.com/halmd-org/halmd. Accessed: 2025-01-10.

[6] Peter H. Colberg and Felix Höfling. *HALMD Documentation and User Manual*. https://halmd.org/doc/. Accessed: 2025-01-10.

[7] Pierre De Buyl et al. “H5MD: A structured, efficient, and portable file format for molecular data”. In: *Computer Physics Communications* 185.6 (2014), pp. 1546–1553. DOI: 10.1016/j.cpc.2014.01.018.

[8] Jörg Behler and Michele Parrinello. “Generalized neural-network representation of high-dimensional potential-energy surfaces”. In: *Physical Review Letters* 98.14 (2007), p. 146401. DOI: 10.1103/PhysRevLett.98.146401.

[9] Frank Noé et al. “Machine learning for molecular simulation”. In: *Annual review of physical chemistry* 71.1 (2020), pp. 361–390.

[10] Albert P Bartók, Risi Kondor, and Gábor Csányi. “On representing chemical environments”. In: *Physical Review B* 87.18 (2013), p. 184115. DOI: 10.1103/PhysRevB.87.184115.

[11] Manzil Zaheer et al. “Deep Sets”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017. URL: https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf.

[12] Albert P Bartók et al. “Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons”. In: *Physical review letters* 104.13 (2010), p. 136403.

[13] Kristof T Schütt et al. “SchNet: A continuous-filter convolutional neural network for modeling quantum interactions”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017. URL: https://proceedings.neurips.cc/paper/2017/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

[14] Simon Batzner et al. “E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials”. In: *Nature Communications* 13.1 (2022), p. 2453. DOI: 10.1038/s41467-022-29939-5.

[15] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[16] Atilim Gunes Baydin et al. “Automatic differentiation in machine learning: a survey”. In: *Journal of machine learning research* 18.153 (2018), pp. 1–43.

[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[18] Martín Abadi et al. “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. In: *arXiv preprint arXiv:1603.04467* (2016).

[19] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[20] “Boost.Autodiff: Automatic Differentiation in Modern C++”. In: *Proceedings of CPP-Now 2023*. 2023.

[21] Matteo Cioni, Daniela Polino, and Daniele Rapetti. *Cu_fcc_slabs Deep Potential Model*. www.aissquare.com/models/detail?pageType=models&name=Cu_fcc_slabs. Machine-learned interatomic potential trained with DeePMD-kit. 2023.

[22] Jinzhe Zeng and Duo Zhang. *Data for DeePMD-kit v2 paper*. https://github.com/deepmodeling-activity/deepmd-kit-v2-paper. Supporting data for: DeePMD-kit v2: A software package for deep potential models, J. Chem. Phys., 159, 054801 (2023). 2023.

[23] Xin Zhong, Felix Höfling, and Timm John. “Hydrogen diffusion in garnet: Insights from atomistic simulations”. In: *Geochemistry, Geophysics, Geosystems* 26.2 (2025), e2024GC011951.