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

Declaration of Authorship

I hereby declare that I have completed this thesis independently and that I have not used any sources or aids other than those explicitly stated. This thesis has not been submitted, in whole or in part, to any other university for the purpose of obtaining a degree.

All direct and indirect quotations have been clearly identified. Sources from the internet have been properly cited.

I am aware of the rules of good scientific practice and understand that violations, in particular plagiarism or misrepresentation of authorship, may lead to disciplinary consequences, including the failure of the thesis.

The following forms of generative AI assistance were used in the preparation of this thesis:

Type of AI assistance	**AI tool used**
Paraphrasing of Introduction and Background sections	ChatGPT 4.0
Summarizing methodological descriptions	ChatGPT 4.0
Assistance with formula organization in the force computation section	ChatGPT 4.0
Linguistic proof reading of the full manuscript	Grammarly

I take full responsibility for all AI-assisted content included in this thesis and confirm that its use complies with the applicable examination regulations. My own academic judgment and editorial control were maintained throughout the work.

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. **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].

3. **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].

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 Ab initio molecular dynamics(AIMD) and Car–Parrinello molecular dynamics(CPMD)

AIMD comprises a class of simulation methods in which interatomic forces are computed directly from quantum-mechanical electronic structure calculations, rather than from predefined empirical potentials. In AIMD, the potential energy surface governing atomic motion is evaluated on-the-fly, most commonly within the framework of density functional theory (DFT). This enables the explicit treatment of electronic effects such as bond formation and breaking, charge redistribution, and many-body interactions with high physical fidelity.

In the most straightforward formulation of AIMD, known as Born–Oppenheimer molecular dynamics (BOMD), the electronic ground state is determined independently at every molecular dynamics time step. For a fixed atomic geometry, the electronic structure problem is solved iteratively using a self-consistent field (SCF) procedure until the Kohn–Sham energy functional is minimized under the constraint of orbital orthonormality. Forces on the nuclei are then obtained from the resulting ground-state energy, and the atomic positions are advanced according to Newton's equations of motion. While this approach yields highly accurate forces, the repeated SCF minimization renders BOMD computationally expensive.

CPMD provides an alternative formulation of AIMD that avoids an explicit electronic ground-state optimization at every time step. Instead, CPMD introduces an extended Lagrangian in which both nuclear positions and Kohn–Sham electronic orbitals are treated as time-dependent variables. The electronic degrees of freedom are assigned a fictitious mass and propagated dynamically alongside the nuclei, while orthonormality constraints enforce the Pauli exclusion principle. When the fictitious electronic mass is chosen sufficiently small, the electronic subsystem remains adiabatically separated from the nuclear motion and stays close to the Born–Oppenheimer surface.

In CPMD, atomic motion follows classical Newtonian dynamics, whereas the electronic orbitals evolve according to equations of motion derived from the Kohn–Sham energy functional. This unified dynamical treatment eliminates the need for repeated SCF minimizations and can substantially reduce the computational cost per time step while maintaining forces consistent with the electronic ground state. Nevertheless, CPMD remains computationally demanding due to the underlying electronic structure calculations, limiting practical simulations to system sizes of a few hundred atoms and time scales on the order of tens to hundreds of picoseconds.

As a result, CPMD is primarily employed as a high-accuracy reference method rather than as a tool for large-scale molecular dynamics simulations. Recent work by Zhong *et al.* illustrates the application of CPMD to hydrogen diffusion in complex garnet systems, demonstrating both its predictive power and its inherent limitations in accessible length and time scales [4]. In the context of this thesis, CPMD serves as a methodological benchmark that defines the target accuracy for interatomic force models and motivates the use of machine-learned potentials capable of reproducing first-principles accuracy at reduced computational cost.

2.2 High-accuracy large-scale molecular dynamics (HALMD) software

HALMD package is a high-performance, open-source molecular dynamics simulation framework designed for large-scale studies of condensed matter systems. HALMD is built around a modular C++ architecture with a strong emphasis on numerical precision, extensibility, and efficient parallel execution on graphics processing units (GPUs) [5]. It provides the computational infrastructure required to simulate systems containing millions of particles over long physical time scales.

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

$$m_i \frac{d^2 \mathbf{r}_i}{dt^2} = -\nabla_i U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N), \tag{1}$$

where the potential energy function U describes interatomic interactions. In standard applications, U is given by analytical or tabulated empirical potentials, such as the Lennard–Jones, Gaussian Core, or Yukawa models. While these models are computationally efficient, they lack the quantum-mechanical accuracy provided by AIMD methods such as CPMD.

A defining strength of HALMD is its GPU-accelerated backend. Computationally intensive tasks, including force evaluation, neighbor-list construction, and time integration, are implemented using NVIDIA CUDA and, more recently, SYCL to support heterogeneous computing platforms [5, 6]. Spatial domain decomposition and cell-based neighbor lists enable efficient scaling on multi-GPU systems, achieving orders-of-magnitude speedups compared to CPU-only implementations.

HALMD follows a modular design philosophy in which integrators, interaction potentials, neighbor-list builders, and observables are implemented as interchangeable components configured via a Lua scripting interface [7]. This modularity allows new force models to be integrated without modification of the core simulation engine. Simulation data are stored in the H5MD format [8], ensuring compatibility with established analysis workflows.

In this thesis, HALMD serves as the computational backbone for large-scale molecular dynamics simulations. By replacing conventional empirical interaction potentials with machine-learned DeepMD-v2 force fields, HALMD becomes capable of performing simulations with near *ab initio* accuracy at classical MD computational cost. The integration of a full multi-species DeepMD-v2 inference pipeline into HALMD thus bridges the gap between the accuracy of CPMD and the scalability required for practical materials modeling.

2.3 Neural network potential (NNP)

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. NNPs overcome these limitations by learning the potential energy surface (PES) directly from *ab initio* reference data [9, 10].

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 [9]:

$$E = \sum_i E_i(\mathcal{R}_i), \tag{2}$$

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

$$\mathcal{L} = p_E L_E + p_F L_F + p_V L_V, \tag{3}$$

where the terms are defined as

$$L_E = \frac{1}{N_E} \sum_{n=1}^{N_E} \left(E_n^{\mathrm{NN}} - E_n^{\mathrm{DFT}}\right)^2, \tag{4}$$

$$L_F = \frac{1}{3N_F} \sum_{n=1}^{N_F} \sum_{i=1}^{N_{\text{atoms}}} \left\|\mathbf{F}_{i,n}^{\mathrm{NN}} - \mathbf{F}_{i,n}^{\mathrm{DFT}}\right\|^2, \tag{5}$$

$$L_V = \frac{1}{9N_V} \sum_{n=1}^{N_V} \left\|\mathbf{V}_n^{\mathrm{NN}} - \mathbf{V}_n^{\mathrm{DFT}}\right\|^2. \tag{6}$$

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 [11, 12]. 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 [9], Gaussian approximation potentials (GAP) [13], SchNet [14], and the E(3)-equivariant NequIP model [15]. 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.

2.4 DeepMD

This work focuses on *DeepMD version 2 kit*, 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 is decomposed into atomic contributions as in Eq. (2), 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:

$$\mathbf{F}_i = -\frac{\partial E}{\partial \mathbf{r}_i}. \tag{7}$$

During training, these derivatives are computed automatically through TensorFlow's backpropagation 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.

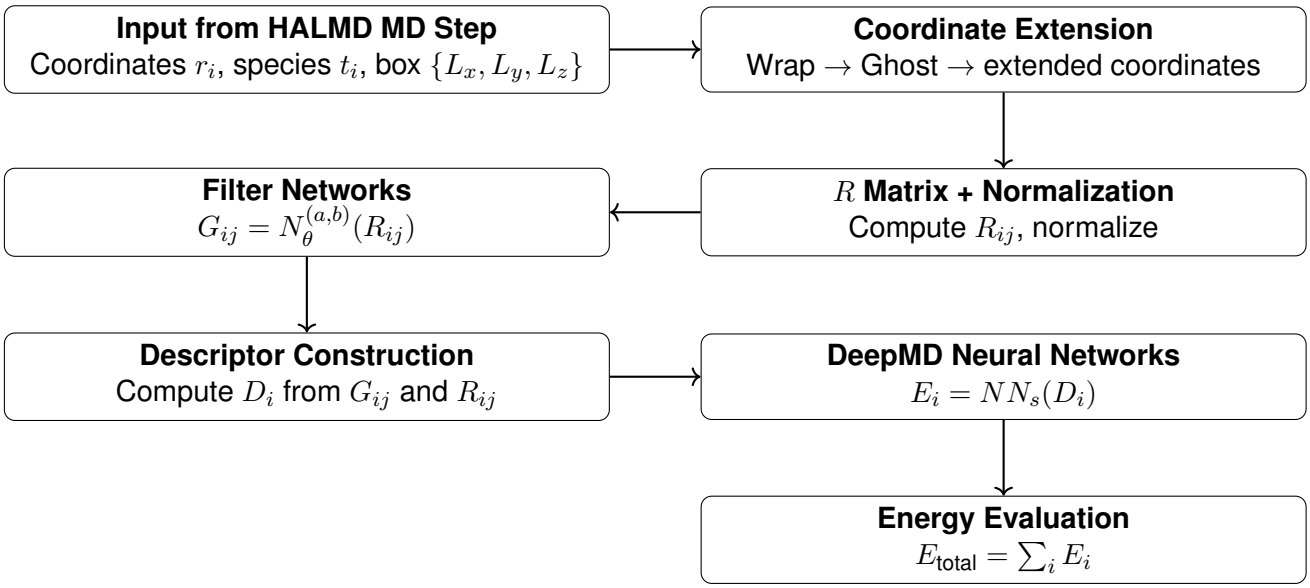

Figure 1: DeepMD–HALMD energy evaluation pipeline implemented in this thesis.

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 reconstructed 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

$$\hat{s}_{ij} \mapsto G_{ij} \tag{8}$$

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 SE 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 [5]. However, this approach provides only the minimal displacement between atoms and does not reproduce the full periodic environment expected by the 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 $[0, L_\alpha)$ for each Cartesian direction $\alpha \in \{x, y, z\}$.

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

$$\tilde{r}_{i\alpha} = r_{i\alpha} - L_\alpha \left\lfloor \frac{r_{i\alpha}}{L_\alpha} \right\rfloor . \tag{9}$$

This operation performs the following steps:

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

$$n_\alpha = \left\lfloor \frac{r_{i\alpha}}{L_\alpha} \right\rfloor . \tag{10}$$

  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 $[0, L_\alpha)$:

$$\tilde{r}_{i\alpha} = r_{i\alpha} - n_\alpha L_\alpha \tag{11}$$

- 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

$$\mathbf{r}_i^{(\mathbf{s})} = \tilde{\mathbf{r}}_i + s_x L_x \, \mathbf{e}_x + s_y L_y \, \mathbf{e}_y + s_z L_z \, \mathbf{e}_z, \qquad s_\alpha \in [-n_{\mathrm{buff},\alpha},\, n_{\mathrm{buff},\alpha}], \tag{12}$$

where L_α are the box lengths and

$$n_{\mathrm{buff},\alpha} = \left\lceil \frac{r_c}{L_\alpha} \right\rceil \tag{13}$$

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`:

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

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

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

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

Sorting omitted for efficiency. In the reference DeepMD implementation, each species block of neighbours is explicitly sorted by increasing interatomic distance,

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

in order to impose a deterministic and easily interpretable layout of the neighbour environments.

In the present implementation, this explicit distance-based sorting step is omitted. The subsequent construction of the environment matrices R and G, together with the associated embedding-network evaluation, preserves the relative correspondence between neighbour entries and their geometric features. Consequently, the numerical values of the resulting descriptors and predicted energies remain invariant under permutations of neighbours within a given species block.

Since the descriptor pipeline and fitting network are permutation-equivariant within each species channel, removing the sorting step does not affect the physical correctness of the model evaluation but eliminates an additional $\mathcal{O}(N \log N)$ cost per atom per species, thereby improving runtime performance for large systems.

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 `sel` defined in Eq. (3.3.6).

After grouping neighbours by species, 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 configuration 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

Here, $\tilde{\mathbf{r}}_i$ denotes the wrapped coordinate of atom i obtained by the coordinate mapping in Eq. (11), and $\mathbf{r}_j^{(\mathbf{s})}$ denotes the ghost-extended coordinate of neighbour atom j constructed via the periodic expansion in Eq. (12).

The relative displacement vector and its magnitude are then given by

$$\mathbf{r}_{ij} = \mathbf{r}_j^{(\mathbf{s})} - \tilde{\mathbf{r}}_i, \qquad r_{ij} = \|\mathbf{r}_{ij}\|. \tag{14}$$

3.4.2 Cutoff behaviour and switching function

Following DeepMD, the radial switching function is defined as

$$s(r) = \begin{cases} \dfrac{1}{r}, & r < r_s, \\ \dfrac{1}{r}\left[x^3(-6x^2+15x-10)+1\right], & r_s \le r < r_c, \\ 0, & r \ge r_c, \end{cases} \qquad x = \frac{r - r_s}{r_c - r_s}. \tag{15}$$

3.4.3 Raw geometric matrix

DeepMD defines the geometric features of neighbour j of atom i as

$$(R_i)_j = \left[s(r_{ij}),\ s(r_{ij})\frac{x_{ij}}{r_{ij}},\ s(r_{ij})\frac{y_{ij}}{r_{ij}},\ s(r_{ij})\frac{z_{ij}}{r_{ij}}\right] \in \mathbb{R}^{1\times 4}. \tag{16}$$

Collecting all neighbour rows yields the raw geometric matrix

$$R_i \in \mathbb{R}^{N_c^{(\mathrm{total})}\times 4}. \tag{17}$$

3.4.4 Species-aware neighbour ordering

Neighbours are arranged by species and padded to fixed per-species capacities prescribed by `sel` (Eq. (3.3.6)), yielding for every atom a geometric matrix of identical shape, as described in Eq. (17).

3.4.5 Feature normalization

In the present implementation, the geometric matrix is further normalized using training-time statistics

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

The normalized matrix is defined as

$$\hat{R}_{k,\alpha} = \frac{R_{k,\alpha} - (t_{\mathrm{avg}})_{k,\alpha}}{(t_{\mathrm{std}})_{k,\alpha}}, \qquad \hat{R} \in \mathbb{R}^{N_c^{(\mathrm{total})}\times 4}. \tag{18}$$

The first column $\hat{s}_{ij} = \hat{R}_{ij,0}$ serves as the scalar input to the embedding network.

3.4.6 Summary

The matrix $\hat{R}$ provides the complete geometric input for the subsequent descriptor construction stage and is fully compatible with the DeepMD-v2 reference implementation.

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 a high-dimensional embedding vector through a species-pair–specific *filter network*. Collecting the embeddings for all neighbour slots produces the matrix

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

where M is the embedding dimension , and $N_c^{(\mathrm{total})} = \sum_b N_c^{(b)}$ is the total neighbour capacity specified by the model's `sel` vector.

3.5.1 Input and output dimensionality

Each neighbour slot contributes a single scalar input

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

obtained from the first column of the normalized geometric matrix $\hat{R}$.

The embedding network maps this scalar to an M-dimensional output:

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

Stacking the embedding vectors for all neighbour rows yields

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

The ordering of the rows is identical to the ordering of the rows of $\hat{R}$: neighbours are grouped by species and placed into fixed-capacity species blocks as prescribed by sel sel.

3.5.2 Structure of the Filter Network

For each central–neighbour species pair (a, b), DeepMD-v2 defines a dedicated multilayer perceptron (MLP)

$$N_\theta^{(a,b)} : \mathbb{R} \to \mathbb{R}^M.$$

Each of these MLPs consists of:

- an input layer of width 1 (the scalar $\hat{s}_{ij}$),
- L hidden layers with species-pair–dependent widths,
- a final output layer of dimension M,

- Tanh activations between layers (except the output layer).

For a filter network with layer widths

$$1 \;\to\; H_1 \;\to\; H_2 \;\to\; \cdots \;\to\; H_L \;\to\; M,$$

the forward pass is

$$\begin{aligned} h^{(1)} &= \tanh\!\Big(W^{(1)}_{a,b}\,\hat{s}_{ij} + b^{(1)}_{a,b}\Big)\,, \\ h^{(\ell)} &= \tanh\!\Big(W^{(\ell)}_{a,b}\,h^{(\ell-1)} + b^{(\ell)}_{a,b}\Big)\,, \qquad \ell = 2,\ldots,L, \\ G_{ij} &= W^{(L+1)}_{a,b}\,h^{(L)} + b^{(L+1)}_{a,b}. \end{aligned} \tag{19}$$

These weights $W^{(\ell)}_{a,b}$ and biases $b^{(\ell)}_{a,b}$ are extracted from the Frozen Model under node names of the form '`filter_type_a/matrix_l_b`' and '`filter_type_a/bias_l_b`' (see Section 3.2). The subscript b corresponds to the neighbour species, confirming explicitly that the network architecture depends on both the central and neighbour species.

3.5.3 Species-pair dependent embedding networks

For a central atom of species a and a neighbour of species b, the embedding is computed using the matching filter network:

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

Different neighbour species therefore produce embeddings through different MLPs even when the geometric distances are identical. This mechanism is essential for modelling alloy systems, where cross-species interactions must be distinguishable in the descriptor.

HALMD stores the entire set of such networks in a two-dimensional structure:

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

3.5.4 Runtime Network Selection and Evaluation

After species-grouped neighbour lists are constructed (Section 3.3), each neighbour row j is associated with a neighbour-species label $b_{(j)}$. The embedding vector for that row is obtained via

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

Thus:

- row j of $\hat{R}$ determines $\hat{s}_{ij}$,
- the species list determines $b_{(j)}$,
- the corresponding network $N^{(a,b_{(j)})}_{\theta}$ is applied,
- the resulting G_{ij} is stored in row j of G.

This ensures strict alignment between rows of $\hat{R}$ and rows of G, exactly as in the DeepMD-v2 computational graph.

3.5.5 Summary

The embedding matrix G is constructed through the following steps:

- extract the scalar geometric input $\hat{s}_{ij}$ from the normalized R-matrix,
- select the appropriate species-pair filter network $N_\theta^{(a,b)}$,
- evaluate its multi-layer structure using the trained weights and biases,
- assemble all embedding vectors into the matrix $G \in \mathbb{R}^{N_c^{(\mathrm{total})} \times M}$.

The resulting matrix provides the learned many-body representation that, together with $\hat{R}$, forms the complete input to the quadratic descriptor described in the next section.

3.6 Descriptor Computation

The DeepPot-SE (`se_e2_a`) descriptor encodes the environment of a central atom i through a combination of (i) normalized geometric features collected in the matrix $\hat{R}_i$ and (ii) learned nonlinear embeddings collected in the matrix G_i. These two matrices are then contracted into a quadratic descriptor D_i, which forms the input to the species-dependent fitting network predicting the atomic energy E_i.

DeepMD-v2 assigns neighbour capacity on a per-species basis. Given species set $\mathcal{S}$ and model parameters $N_c^{(b)}$ for each species $b \in \mathcal{S}$, the total neighbour capacity is

$$N_c^{(\mathrm{total})} = \sum_{b \in \mathcal{S}} N_c^{(b)}. \tag{20}$$

All matrices involved in descriptor computation are dimensioned using this total capacity.

3.6.1 Normalized geometric matrix

Once the wrapped coordinates, periodic images, and species-ordered neighbour lists have been constructed (Section 3.3), the geometric features for atom i are assembled into the matrix

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

Each row corresponds to a specific neighbour slot and contains four quantities:

1. the normalized inverse distance $\hat{s}_{ij}$,
2. the three normalized angular components $\hat{x}_{ij}/r_{ij}^2$, $\hat{y}_{ij}/r_{ij}^2$, $\hat{z}_{ij}/r_{ij}^2$.

Normalization follows the statistics extracted from the trained model:

$$\hat{R}_{ij,\alpha} = \frac{R_{ij,\alpha} - (t_{\mathrm{avg}})_\alpha}{(t_{\mathrm{std}})_\alpha}, \qquad \alpha = 0, \ldots, 3. \tag{21}$$

The first column is used as the scalar geometric input for the nonlinear embedding:

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

This matrix therefore represents a geometrically structured, normalized description of the neighbourhood of atom i, with rows aligned to the species-block ordering of DeepMD-v2.

3.6.2 Embedding matrix

Each neighbour slot j has a known neighbour species $b(j)$. DeepMD-v2 assigns a distinct filter (embedding) network for every central–neighbour pair (a, b), where a is the central species:

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

with $N_\theta^{(a,b)} : \mathbb{R} \to \mathbb{R}^M$.

Evaluating all such networks yields

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

Only the first M' embedding channels are required for the descriptor:

$$G_i^{\mathrm{trunc}} = G_i[:, 1 : M'] \in \mathbb{R}^{N_c^{(\mathrm{total})} \times M'}. \tag{22}$$

This embedding stage transforms the purely geometric scalar inputs $\hat{s}_{ij}$ into learned, species-dependent representations, with rows aligned exactly to the rows of $\hat{R}_i$.

3.6.3 Quadratic descriptor construction

The DeepPot-SE descriptor is obtained by combining the geometric and embedding matrices into a quadratic tensor:

$$D_i = \frac{1}{(N_c^{(\mathrm{total})})^2} \left(G_i^{\mathsf{T}} \hat{R}_i\right) \left(\hat{R}_i^{\mathsf{T}} G_i^{\mathrm{trunc}}\right). \tag{23}$$

The intermediate products have the following shapes:

$$G_i^{\mathsf{T}} \hat{R}_i \in \mathbb{R}^{M \times 4},$$
$$\hat{R}_i^{\mathsf{T}} G_i^{\mathrm{trunc}} \in \mathbb{R}^{4 \times M'},$$
$$D_i \in \mathbb{R}^{M \times M'}.$$

DeepMD-v2 flattens this matrix in row-major order to obtain the final descriptor vector:

$$\mathbf{D}_i \in \mathbb{R}^{MM'}.$$

This vector is the complete learned representation of the neighbourhood of atom i, structured and normalized exactly as required by the DeepMD fitting network.

3.6.4 Summary

The descriptor pipeline implemented in HALMD reproduces the DeepMD-v2 formulation without modification. In particular:

- the neighbour axis matches the species-resolved capacities $N_c^{(b)}$,
- the geometric matrix $\hat{R}_i$ implements all preprocessing steps required by DeepMD-v2,

- the embedding matrix G_i is produced by the full set of species-pair networks $N_\theta^{(a,b)}$,
- the quadratic descriptor D_i and its scaling follow the DeepMD-v2 definition,
- the final descriptor vector $\mathbf{D}_i$ matches TensorFlow output to floating-point precision.

This yields a complete, dimensionally consistent descriptor pipeline suitable for accurate, high-performance DeepMD-v2 inference inside HALMD.

3.7 Potential energy calculation

In the deep potential formalism, the total potential energy of a system of N atoms is written as a sum of atomic contributions (Eq. (2)), where the 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 energy is computed as

$$E_i = \mathrm{NN}_{s_i}(\mathbf{D}_i), \tag{24}$$

where NN_{s_i} denotes the DeepMD-v2 fitting network associated with species s_i. HALMD reconstructs these networks directly from the TensorFlow frozen graph (`frozen_model.pb`), including all linear layers, activation functions, residual branches, and output transformations.

3.7.1 Baseline implementation prior to this Work

The earlier HALMD implementation [1] supported inference only for *single-species* DeepMD models. It successfully reproduced DeepMD energies for monoatomic systems but lacked the structural components required for DeepMD-v2 multi-species inference:

- only one fitting network for all atoms,
- no integration with multi-species descriptors ($\hat{R}$, G, D),
- no per-species energy shift (`bias_atom_e`),
- no mechanism for mapping descriptor vectors to the appropriate network.

Therefore, multi-component DeepMD-v2 models could not be evaluated within HALMD.

3.7.2 Species-dependent fitting networks

DeepMD-v2 defines a separate fitting neural network for each atomic species. In the TensorFlow graph, these networks appear under node groups of the form:

$$\texttt{layer\_}\ell\texttt{\_type\_}a\texttt{/matrix}, \qquad \texttt{layer\_}\ell\texttt{\_type\_}a\texttt{/bias}, \qquad \texttt{final\_layer\_type\_}a\texttt{/}*,$$

where:

- a indexes the central-atom species,
- ℓ is the layer index,
- each pair of `matrix` and `bias` nodes defines an affine layer.

HALMD reconstructs the complete set

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

and evaluates the appropriate network according to

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

The names of the corresponding TensorFlow nodes make the species mapping explicit:

- for species $a = 0$: `layer_0_type_0/matrix`, `layer_1_type_0/matrix`, `final_layer_type_0/matrix`, etc.
- for species $a = 1$: `layer_0_type_1/matrix`, `layer_1_type_1/matrix`, `final_layer_type_1/matrix`, etc.

These node groups fully specify all weights and biases of each species-dependent fitting network.

3.7.3 Per-species energy offsets

DeepMD-v2 optionally includes constant energy offsets for each species, stored under:

`bias_atom_e`.

HALMD extracts these tensors and evaluates

$$E_i = \mathrm{NN}_{s_i}(\mathbf{D}_i) + b_{s_i}. \tag{25}$$

This ensures that the final energy scale matches the one used during DeepMD model training.

3.7.4 Integration with the multi-Species descriptor pipeline

The descriptor vector $\mathbf{D}_i$ supplied to the fitting network has the exact structure expected by the DeepMD-v2 TensorFlow implementation. Its construction incorporates:

- the normalized geometric matrix $\hat{R}_i \in \mathbb{R}^{N_c^{(\mathrm{total})} \times 4}$,
- neighbour-slot ordering based on the model's `sel` vector,
- embedding vectors from species-pair networks $N_\theta^{(a,b)}(\hat{s}_{ij})$,
- truncation to the first M' embedding channels,
- the quadratic descriptor construction $D_i \in \mathbb{R}^{M \times M'}$,
- flattening into $\mathbf{D}_i \in \mathbb{R}^{MM'}$.

The ordering and dimensionality of $\mathbf{D}_i$ must match the structure expected by the fitting networks defined in `layer__type_a/*` and `final_layer_type_a/*`. The revised HALMD implementation enforces this consistency.

3.7.5 Reproduction of DeepMD-v2 energies

With the full multi-species pipeline integrated, HALMD reproduces DeepMD-v2 energy predictions to floating-point precision across all tested systems:

- monoatomic models (Cu),
- multi-species alloy models (HEA),
- alternative Copper models with different normalization statistics.

Both total energies and per-atom energies match DeepMD's TensorFlow-based outputs.

3.7.6 Final formulation

HALMD now evaluates the energy of each atom using the DeepMD-v2 expression

$$E_i = \mathrm{NN}_{s_i}(\mathbf{D}_i) + b_{s_i}, \tag{26}$$

and accumulates the total energy as

$$E_{\mathrm{tot}} = \sum_{i=1}^{N} E_i. \tag{27}$$

This formulation is bitwise compatible with the DeepMD-v2 computational graph and is suitable for high-performance molecular dynamics simulations of multi-species systems within HALMD.

4 Force computation

In the deep potential framework, forces are obtained as the negative gradient of the total potential energy with respect to atomic coordinates as in Eq. (7). 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)

The computation of atomic forces in DeepMD requires propagating derivatives through a sequence of transformations involving geometric preprocessing, species-pair embedding networks, quadratic descriptor construction, and finally a species-dependent fitting network. While the geometric derivatives admit closed-form expressions, the neural networks involve nonlinear activations and residual connections that make analytic differentiation impractical. For these components, HALMD employs *automatic differentiation* (AD).

Automatic differentiation evaluates derivatives by decomposing a computation into elementary operations and applying the chain rule locally [16, 17]. Two modes of AD are commonly used:

4.1.1 Forward-mode AD (current HALMD implementation)

Forward-mode AD propagates derivatives *together with the value* of every intermediate variable. Intuitively, each quantity in the computation carries a "derivative tag" that is updated whenever the variable participates in an operation.

This mode is efficient when:

- the number of inputs is small, and
- each function has a scalar or low-dimensional output.

These conditions hold for the DeepMD *embedding networks* $N_\theta^{(a,b)}$, each of which maps a single scalar input $\hat{s}_{ij}$ to an M-dimensional output. HALMD therefore evaluates both the value

$$G_{ij}, \qquad \frac{\partial G_{ij}}{\partial \hat{s}_{ij}}$$

using forward-mode AD provided by Boost.Autodiff [18].

Forward-mode AD is also used in the fitting network to compute $\partial E_i / \partial D_{i,\alpha}$, since each descriptor component $D_{i,\alpha}$ can be marked as a differentiable input.

4.1.2 Reverse-mode AD (not yet implemented)

Reverse-mode AD performs the computation in two sweeps:

1. a forward sweep computes all intermediate values,
2. a backward sweep accumulates derivatives from the output back through the computational graph.

Intuitively, reverse-mode AD works like backpropagation in neural networks: one starts from the scalar loss (or, in this case, the atomic energy E_i) and distributes sensitivities backward across all intermediate variables.

Reverse-mode AD is most efficient when:

- the function has a *single scalar output*, and
- the number of inputs is large.

This matches the structure of the *fitting network*:

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

where the input dimension $\dim(\mathbf{D}_i) = MM'$ may be several hundred. In such a setting, reverse-mode AD would greatly reduce computational cost, as the full Jacobian $\partial E_i / \partial D_{i,\alpha}$ could be computed in a *single* backward pass.

At present, HALMD implements only forward-mode AD. Although correct, this results in one forward-mode evaluation per descriptor entry, which is more expensive than necessary for large multi-species models. Incorporating reverse-mode AD in the fitting network is therefore a promising future improvement for performance-critical simulations.

4.1.3 Hybrid analytic–AD differentiation

All geometric derivatives—including those of the wrapped coordinates, ghost-cell transformations, inverse-distance expressions, and switching functions—are evaluated analytically. AD is used exclusively inside the neural-network components. This hybrid strategy offers:

- exact agreement with the DeepMD-v2 TensorFlow implementation,
- efficient evaluation of nonlinear network derivatives,
- clear separation between analytic geometric contributions and AD-driven neural contributions.

The following sections detail the analytic components required to complete the force calculation: derivatives of the geometric mapping, descriptor derivatives, and 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 defined in Eq. (14); we write $\mathbf{r}_{ij} = (d_x, d_y, d_z)$ and $r_{ij} = \|\mathbf{r}_{ij}\|$, and on the radial function

$$s(r_{ij})$$

defined in Section 3.4.

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, we use the geometry row in Eq. (16) with the switched inverse distance $s(r_{ij})$ replaced by $\tilde{s}_{ij}$.

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

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

From Section 3.4, the radial switching function is

$$s(r) = \begin{cases} \dfrac{1}{r}, & r < r_s, \\ \dfrac{1}{r}\left[x^3(-6x^2 + 15x - 10) + 1\right], & r_s \le r < r_c, \qquad x = \dfrac{r - r_s}{r_c - r_s}. \\ 0, & r \ge r_c, \end{cases} \tag{28}$$

For $r_s \le r < r_c$, define

$$P(x) = x^3(-6x^2 + 15x - 10) + 1.$$

Then

$$\frac{dP}{dx} = x^2(-30x^2 + 60x - 30), \qquad \frac{dx}{dr} = \frac{1}{r_c - r_s}.$$

The derivative of $s(r)$ is therefore

$$\frac{ds}{dr} = \begin{cases} -\dfrac{1}{r^2}, & r < r_s, \\ -\dfrac{1}{r^2}P(x) + \dfrac{1}{r}\dfrac{dP}{dx}\dfrac{1}{r_c - r_s}, & r_s \le r < r_c, \\ 0, & r \ge r_c. \end{cases} \tag{29}$$

4.2.3 Step 4: Derivative of the Radial Component

The first component of the geometry row is

$$R_{ij,0} = s(r_{ij}).$$

Applying the chain rule,

$$\frac{\partial R_{ij,0}}{\partial \mathbf{r}_i} = \frac{ds}{dr}(r_{ij})\frac{\partial r_{ij}}{\partial \mathbf{r}_i} = -\frac{ds}{dr}(r_{ij})\,\hat{\mathbf{r}}_{ij}.$$

4.2.4 Step 5: Derivative of the Directional Components

Each directional component is

$$R_{ij,\alpha} = s(r_{ij})\,\frac{d_\alpha}{r_{ij}}, \qquad \alpha \in \{x, y, z\}.$$

Using the product rule and the results of the previous steps,

$$\frac{\partial R_{ij,\alpha}}{\partial \mathbf{r}_i} = -\frac{ds}{dr}(r_{ij})\frac{d_\alpha}{r_{ij}}\,\hat{\mathbf{r}}_{ij} - s(r_{ij})\,\frac{1}{r_{ij}}\mathbf{e}_\alpha + s(r_{ij})\,\frac{d_\alpha}{r_{ij}^2}\,\hat{\mathbf{r}}_{ij}.$$

This formula provides the complete derivative of the directional components $R_{ij,1}, R_{ij,2}, R_{ij,3}$.

4.2.5 Step 6: Normalized Geometry Derivative

Note that the raw radial feature $s(r_{ij}) = R_{ij,0}$ becomes the normalized quantity $\hat{s}_{ij} := \hat{R}_{ij,0}$ after feature normalization, and it is this normalized scalar $\hat{s}_{ij}$ that is used as the input to the embedding network.

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

Since the normalization parameters are constants,

$$\frac{\partial \hat{R}_{ij,\alpha}}{\partial \mathbf{r}_i} = \frac{1}{(t_{\text{std}})_\alpha}\frac{\partial R_{ij,\alpha}}{\partial \mathbf{r}_i}, \tag{30}$$

4.2.6 Step 7: 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},

$$\frac{\partial r_{ij}}{\partial \mathbf{r}_j} = \sum_\alpha \frac{\partial r_{ij}}{\partial d_\alpha}\frac{\partial d_\alpha}{\partial \mathbf{r}_j} = \sum_\alpha \frac{d_\alpha}{r_{ij}}\,\mathbf{e}_\alpha = +\,\hat{\mathbf{r}}_{ij}, \tag{31}$$

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 Derivative of the Embedding Matrix

In DeepPot–SE, each row of the embedding matrix

$$G \in \mathbb{R}^{N_c \times M}$$

is produced by an embedding neural network that depends on the local geometry only through the scalar input

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

the first component of the normalized geometric matrix. Hence, for each neighbour j,

$$G_{j:} = \mathrm{N}_\theta^{(a,b)}(\hat{s}_{ij}).$$

The derivative of G with respect to the geometry therefore reduces to the one–dimensional derivative of the embedding network with respect to its scalar input:

$$\boxed{\frac{\partial G_{jp}}{\partial \hat{R}_{ij,0}} = \frac{d\,\mathrm{N}_\theta^{(a,b)}}{d\hat{s}}\big(\hat{s}_{ij}\big), \qquad p = 1,\dots,M.} \tag{32}$$

All other components of $\hat{R}$ do not influence G:

$$\frac{\partial G_{jp}}{\partial \hat{R}_{j\alpha}} = 0, \alpha \neq 0.$$

The vector

$$\mathbf{q}_j := \left(\frac{\partial G_{j1}}{\partial \hat{R}_{ij,0}}, \ldots, \frac{\partial G_{jM}}{\partial \hat{R}_{ij,0}} \right) \in \mathbb{R}^M$$

is obtained by automatic differentiation of the embedding network and is treated as a known quantity in the subsequent descriptor and force derivations.

4.4 Descriptor Derivative

The DeepPot-SE descriptor is constructed from the interaction between the embedding matrix $G \in \mathbb{R}^{N_c \times M}$ and the normalized geometric matrix $\hat{R} \in \mathbb{R}^{N_c \times 4}$. Their contraction enters through the intermediate matrix

$$C = \frac{1}{N_c} G^{\mathsf{T}} \hat{R} \qquad C \in \mathbb{R}^{M \times 4}.$$

The matrix C aggregates all geometry–embedding interactions and forms the foundation of the quadratic descriptor used in DeepMD.

The full (extended) descriptor is defined as the Gram matrix

$$D_{\text{ext}} = CC^{\mathsf{T}} \qquad D_{\text{ext}} \in \mathbb{R}^{M \times M}. \tag{33}$$

This construction ensures permutation invariance of neighbours and couples all embedding channels quadratically.

However, the actual DeepPot-SE descriptor used by the fitting network consists only of the first M' columns of D_{ext}:

$$D = D_{\text{ext}}[:, 1{:}M'], \qquad D \in \mathbb{R}^{M \times M'}. \tag{34}$$

Thus, the descriptor truncation is implemented by a simple column restriction.

Derivative with respect to the intermediate matrix C

From the component definition

$$(D_{\text{ext}})_{mn} = \sum_{\alpha=1}^{4} C_{m\alpha} C_{n\alpha},$$

the derivative of D_{ext} with respect to one entry $C_{p\beta}$ follows by direct application of the product rule:

$$\frac{\partial (D_{\text{ext}})_{mn}}{\partial C_{p\beta}} = \delta_{mp} C_{n\beta} + \delta_{np} C_{m\beta},$$

where δ_{ij} denotes the Kronecker delta.

Since the network descriptor D retains only the first M' columns of D_{ext}, the corresponding derivative is

$$\frac{\partial D_{mn}}{\partial C_{p\beta}} = \delta_{mp} C_{n\beta} + \delta_{np} C_{m\beta}, \qquad n \leq M'.$$

Derivative with respect to the normalized geometric matrix

The intermediate matrix

$$C = \frac{1}{N_c} G^{\mathsf{T}} \hat{R}$$

depends on $\hat{R}$ in two distinct ways: (i) explicitly through the right factor $\hat{R}$, and (ii) implicitly through the embedding matrix G, since each row $G_{j:}$ is produced by the embedding network from the scalar input $\hat{s}_{ij} := \hat{R}_{ij,0}$ of the normalized geometry.

Accordingly, the derivative of C with respect to $\hat{R}$ must be decomposed into an explicit and an implicit contribution,

$$\frac{\partial C}{\partial \hat{R}} = \left.\frac{\partial C}{\partial \hat{R}}\right|_{G} + \left.\frac{\partial C}{\partial G}\right|_{\hat{R}} \frac{\partial G}{\partial \hat{R}}.$$

In component form, using

$$C_{p\beta} = \frac{1}{N_c} \sum_{j=1}^{N_c} G_{jp} \hat{R}_{j\beta},$$

the explicit contribution (holding G fixed) is

$$\left.\frac{\partial C_{p\beta}}{\partial \hat{R}_{j\alpha}}\right|_{G} = \frac{1}{N_c} G_{jp}\, \delta_{\beta\alpha}.$$

Since the embedding matrix G depends on $\hat{R}$ only through the scalar input $\hat{R}_{ij,0}$, the implicit contribution simplifies to

$$\frac{\partial G_{\ell p}}{\partial \hat{R}_{j\alpha}} = \delta_{\ell j}\, \delta_{0\alpha}\, \frac{\partial G_{jp}}{\partial \hat{R}_{ij,0}},$$

where $\partial G_{jp}/\partial \hat{R}_{ij,0}$ is obtained by differentiation of the embedding network with respect to its scalar input.

Substituting this relation yields the full derivative

$$\boxed{\frac{\partial C_{p\beta}}{\partial \hat{R}_{j\alpha}} = \frac{1}{N_c} G_{jp}\, \delta_{\beta\alpha} + \frac{1}{N_c}\, \delta_{0\alpha}\, \frac{\partial G_{jp}}{\partial \hat{R}_{ij,0}}\, \hat{R}_{j\beta}.}$$

Descriptor derivative with respect to $\hat{R}$. Since

$$(D_{\mathrm{ext}})_{mn} = \sum_{\beta=1}^{4} C_{m\beta} C_{n\beta}, \qquad D_{mn} = (D_{\mathrm{ext}})_{mn}, \quad n \le M',$$

the chain rule gives, for $n \le M'$,

$$\frac{\partial D_{mn}}{\partial \hat{R}_{j\alpha}} = \sum_{\beta=1}^{4} \left(\frac{\partial C_{m\beta}}{\partial \hat{R}_{j\alpha}}\, C_{n\beta} + C_{m\beta}\, \frac{\partial C_{n\beta}}{\partial \hat{R}_{j\alpha}} \right).$$

Using the previously derived expression

$$\frac{\partial C_{p\beta}}{\partial \hat{R}_{j\alpha}} = \frac{1}{N_c}\, G_{jp}\, \delta_{\beta\alpha} + \frac{1}{N_c}\, \delta_{0\alpha}\, \frac{\partial G_{jp}}{\partial \hat{R}_{ij,0}}\, \hat{R}_{j\beta},$$

we obtain

$$\boxed{\frac{\partial D_{mn}}{\partial \hat{R}_{j\alpha}} = \frac{1}{N_c}\left(G_{jm} C_{n\alpha} + G_{jn} C_{m\alpha}\right) + \frac{1}{N_c}\, \delta_{0\alpha} \left(\frac{\partial G_{jm}}{\partial \hat{R}_{ij,0}} \sum_{\beta=1}^{4} \hat{R}_{j\beta} C_{n\beta} + \frac{\partial G_{jn}}{\partial \hat{R}_{ij,0}} \sum_{\beta=1}^{4} \hat{R}_{j\beta} C_{m\beta} \right), \qquad n \le M'.}$$

This expression shows explicitly that only the radial component $\hat{R}_{ij,0}$ influences the descriptor through the embedding network, while the remaining geometric components affect C only through the direct linear term.

4.5 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,

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

where s_i denotes the species of atom i and $D_i \in \mathbb{R}^{M \times M'}$ is the descriptor matrix constructed in Section 3.6. To assemble forces, we require the gradient (Jacobian row)

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

which measures the sensitivity of the atomic energy with respect to each descriptor component.

In the frozen TensorFlow graph, the fitting network for species s is encoded by nodes such as

`layer_0_type_s/matrix`, `layer_0_type_s/bias`, ..., `final_layer_type_s/matrix`, `final_layer_type_s/bias`.

together with optional residual-scaling nodes `idt` and the per-species energy offset `bias_atom_e`. These parameters are extracted as described in Section 3.2 and reconstructed in HALMD's neural-network module.

4.5.1 Forward-mode automatic differentiation of the fitting network

The fitting networks in DeepMD–v2 consist of several fully connected layers with nonlinear activation functions, optional residual connections, and timestep-related scalings. The resulting mapping

$$\mathbf{D}_i \longmapsto E_i$$

is highly nonlinear, and an explicit hand-derived expression for $\partial E_i / \partial D_{i,\alpha}$ would be lengthy and error-prone.

Instead, HALMD evaluates these derivatives using *forward-mode automatic differentiation* (AD) provided by Boost.Autodiff [18]. Conceptually, forward-mode AD augments each

input component with an “infinitesimal” tangent and propagates both value and derivative through all primitive operations.

For the fitting network, this proceeds as follows for each descriptor component α:

1. The descriptor matrix D_i is flattened and promoted to an array of autodiff variables, where the α-th component is seeded with unit tangent and all others with zero tangent.
2. This augmented input is propagated forward through all layers of the network, using the same weights, biases, activation functions, and residual connections as in the standard inference.
3. At the output, the autodiff scalar representing E_i contains both the energy value and its derivative with respect to $D_{i,\alpha}$. The derivative part is read out as $\partial E_i / \partial D_{i,\alpha}$.

Repeating this procedure for $\alpha = 1, \ldots, M'$ yields the full gradient

$$\boxed{\nabla_{\mathbf{D}_i} E_i = \left(\frac{\partial E_i}{\partial D_{i,1}}, \ldots, \frac{\partial E_i}{\partial D_{i,M'}} \right)} \tag{35}$$

which HALMD stores as the Jacobian row associated with atom i.

From an algorithmic point of view, this is equivalent to computing one forward pass per descriptor component, i.e. the cost scales linearly with M'. For the descriptor sizes considered in this work, this is acceptable and greatly simplifies the implementation because no explicit backpropagation code is required.

4.5.2 Relation to reverse-mode AD and future improvements

For scalar outputs and high-dimensional inputs, *reverse-mode* AD is theoretically more efficient than forward mode, as it computes the entire gradient $\nabla_{\mathbf{D}_i} E_i$ in a single backward sweep. This is the strategy employed by deep-learning frameworks such as TensorFlow [19] and PyTorch [20].

In the present HALMD implementation, only forward-mode AD has been realized for the fitting networks. Extending the neural-network module to support reverse-mode AD would reduce the asymptotic cost of gradient evaluation and is therefore a natural target for future optimisation. Nevertheless, the forward-mode approach used here already provides exact derivatives of the reconstructed fitting networks and is sufficient to achieve bitwise agreement with DeepMD–v2 energies and forces for all tested models.

4.6 Final Force Assembly

We now assemble all derivative components derived in the preceding sections to obtain the explicit DeepMD–v2 force expression. Starting from the energy decomposition in Eq. (2), the force on atom k is

$$\mathbf{F}_k = -\frac{\partial E}{\partial \mathbf{r}_k} = -\sum_i \frac{\partial E_i}{\partial \mathbf{r}_k}. \tag{36}$$

Using the chain rule along the descriptor construction pathway

$$\mathbf{r} \longrightarrow \hat{R} \longrightarrow G \longrightarrow C \longrightarrow D \longrightarrow E,$$

we obtain

$$\boxed{\mathbf{F}_k = -\sum_i \sum_{m=1}^{M} \sum_{n=1}^{M'} J_{i,mn} \frac{\partial D_{i,mn}}{\partial \mathbf{r}_k}} \tag{37}$$

where

where

$$\boxed{J_i := \frac{\partial E_i}{\partial D_i} \in \mathbb{R}^{M \times M'}}$$

4.6.1 Descriptor–level contribution

From Section 4.4, the derivative of the descriptor with respect to the normalized geometry is

$$\boxed{\frac{\partial D_{i,mn}}{\partial \hat{R}_{i,j\alpha}} = \frac{1}{N_c} \left(G_{i,jm} C_{i,n\alpha} + G_{i,jn} C_{i,m\alpha} \right) + \frac{1}{N_c} \delta_{0\alpha} \left(\frac{\partial G_{i,jm}}{\partial \hat{R}_{i,j0}} \sum_\beta \hat{R}_{i,j\beta} C_{i,n\beta} + \frac{\partial G_{i,jn}}{\partial \hat{R}_{i,j0}} \sum_\beta \hat{R}_{i,j\beta} C_{i,m\beta} \right).} \tag{38}$$

4.6.2 Geometric contribution

The normalized geometry depends explicitly on the atomic coordinates, and its analytic derivative

$$\frac{\partial \hat{R}_{i,j\alpha}}{\partial \mathbf{r}_k}$$

was derived in Section 4.2. This term contains the complete geometric information (relative displacements, inverse distances, directional components, and switching functions).

4.6.3 Final force expression

Combining the descriptor derivative with the geometric derivative yields the explicit DeepMD–v2 force:

$$\boxed{\mathbf{F}_k = -\sum_i \sum_j \sum_{m=1}^{M} \sum_{n=1}^{M'} J_{i,mn} \frac{\partial D_{i,mn}}{\partial \hat{R}_{i,j\alpha}} \frac{\partial \hat{R}_{i,j\alpha}}{\partial \mathbf{r}_k}.} \tag{39}$$

This expression decomposes the force into a purely neural contribution (J_i and $\partial G / \partial \hat{R}_{ij,0}$) and an analytic geometric contribution ($\partial \hat{R} / \partial \mathbf{r}_k$), making the entire force evaluation mathematically explicit and numerically stable.

Equation (4.6) therefore reconstructs the full DeepMD–v2 force pathway in HALMD using a consistent hybrid of analytic geometry and automatic differentiation for the neural components.

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.

A practical consideration in this comparison is numerical precision. DeepMD reference energies are evaluated in double precision within the TensorFlow framework, whereas the HALMD implementation evaluates the DeepMD-v2 inference pipeline primarily in single-precision floating-point arithmetic on the GPU for performance reasons. For this reason, agreement is assessed at the level of single-precision floating-point accuracy, and HALMD energies are reported accordingly.

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

Monoatomic Copper model (model A)[21]

Table 1: Energy comparison for the Cu model. HALMD energies are evaluated in single precision; DeepMD energies are shown as produced by the reference implementation.

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

Agreement is exact up to single-precision floating-point accuracy, with remaining differences attributable to rounding.

High-entropy alloy (HEA) model [22]

Table 2: Energy comparison for the HEA model. HALMD energies are evaluated in single-precision arithmetic on the GPU.

Cells	Atoms	DeepMD Energy (eV)	HALMD Energy (eV)
$1 \times 1 \times 1$	16	-66.7551048	-66.7551
$2 \times 2 \times 2$	128	-895.49828775	-895.498
$3 \times 3 \times 3$	432	-3490.86044623	-3490.86

Agreement is exact up to single-precision floating-point accuracy across all tested system sizes.

Garnet model [4]

Table 3: Energy comparison for the garnet model used in [4] using the same configuration used in the paper. HALMD energies are reported at single-precision accuracy.

Atoms	Species	DeepMD Energy (eV)	HALMD Energy (eV)
161	5	-108445	-108445

Energy agreement again confirms the correctness of the descriptor and neural-network forward pass for a chemically complex multi-species system.

Second Copper model (model B)[22]

Table 4: Energy comparison for an alternative Cu model with different training statistics. HALMD energies are evaluated in single precision.

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

This confirms that HALMD correctly reproduces DeepMD energies for models with different training statistics, including 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 [22]*, 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).

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

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.

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.

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

[5] 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.

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

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

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

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

[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.