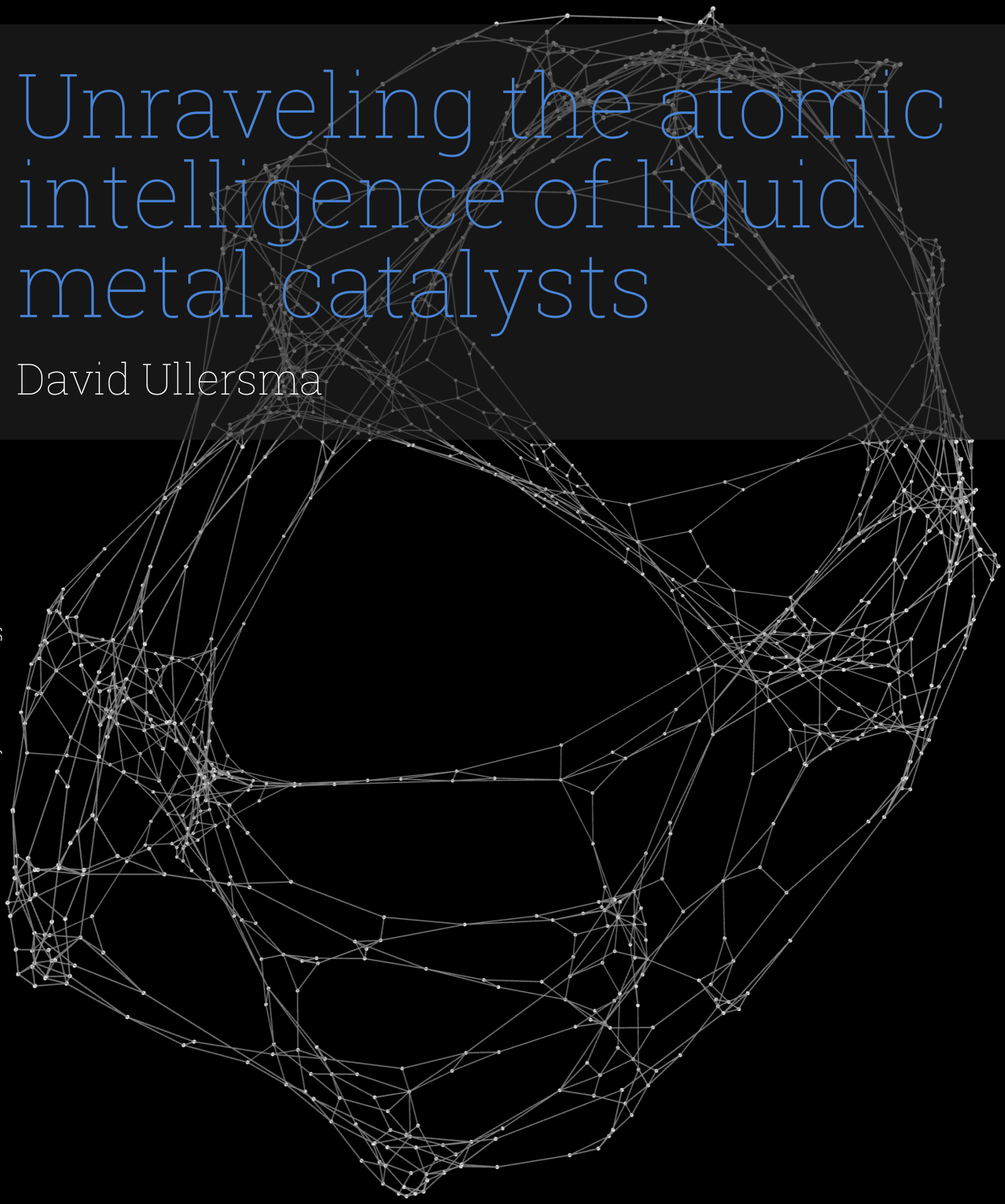
Unraveling the atomic intelligence of liquid metal catalysts

David Ullersma



Delft University of Technology

TU Delft

Unraveling the atomic intelligence of liquid metal catalysts

by

David Ullersma

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 3 february, 2026 at 15:00

Student number: 5663091
Project duration: September 1, 2025 – February 1, 2026
Thesis committee:

Prof. dr. E. Pidko,	TU Delft, Responsible supervisor
Dr. K. Rossi,	TU Delft, Daily supervisor
Dr. ir. A. Vasileiadis	TU Delft, Second examiner
Dr. L.L. Cutz IJchajcha	TU Delft, Third examiner

TU Delft

Abstract

This master thesis looks at gallium Supported Catalytically Active Liquid Metal (SCALM) systems which show great promise in different catalytic applications, specifically propane dehydrogenation is taken as its main focus. Machine learning interatomic potential models (MLIP) provide a way to accurately simulate large atomic systems at long time scales. This new and improved speed provides the possibility to investigate dynamic systems. A selection of the best currently available models were benchmarked on computational efficiency and accuracy to make an informed choice of MLIP model architecture. Models were finetuned further to improve accuracy with DFT data. Using the best model, MD simulations were performed on gallium nanoparticles of different sizes and solutes. Pt, Pd and Ag were observed to equilibrate in the subsurface as was seen in previous literature. Breaking from this trend, gold was observed to reside in the surface and bismuth was seen to migrate to the surface of the nanoparticle. Clustering MACE descriptors showed it could accurately discern 4 groups: solute atoms, surface atoms, high and low coordination bulk. Furthermore, propane was added to a GaPt SCALM system to capture the dynamic site formation using MD, but none were recorded. The entire computational workflow was designed in Python and can serve as a basis for (multi-architectural) MLIP nanoparticle computational workflows.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ACE	Atomic Cluster Expansion
ASE	Atomic Simulation Environment
AIMD	Ab initio Molecular Dynamics
CNN	Convolutional Neural Network
COM	Center Of Mass
CPU	Central Processing Unit
DFT	Density functional Theory
EAM	Embedded Atom Model
(ML)FF	(Machine Learning) Force Field
fs	Femtosecond
GNN	Graph Neural Network
GPU	Graphical Processing Unit
IMC	InterMetallic Compound
MAE	Mean Absolute Error
MD	Molecular Dynamics
ML	Machine Learning
(u)MLIP	(universal) Machine Learning Interatomic Potential
MLP	Multi Layer Perceptron
MPNN	Message Passing Neural Network
MSD	Mean Squared Deviation
NN	Neural Network
OOD	Out Of Distribution
PDF	Pair Distribution Function
PES	Potential Energy Surface
ps	Picosecond
RDF	Radial Density Function
RMSD	Root Mean Square Deviation
RMSE	Root Mean Square Error
SAC	Single Atom Catalyst
SCALM	Supported Catalytically Active Liquid Metal
SD	Squared Deviation
SOAP	Smooth Overlap of Atomic Positions
VRAM	Video Random Access Memory

1 Introduction

Heterogeneous catalysis is embedded in the chemical industry because of a wide array of advantages, which include (but are not limited to): easy product catalyst separation or recovery, good thermal stability and wide applicability [18, 11]. Despite these advantages, there are still disadvantages, such as poor selectivity, solid byproduct formation causing deactivation, use of a (relatively) large amount of noble metals, which are costly, and mass transfer limitations [24]. Supported Catalytically Active Liquid Metal Solutions (SCALMs) can be a solution to some of these challenges. The fluxionality of a liquid metal catalyst means that reactive sites can form around reactants, and site blocking as a result of coking can be negated. As a result, the lifespan and stability of a catalyst can be significantly increased [2]. Liquid catalysts can also provide an environment for multi-step catalytic reactions, which would otherwise require multiple sites, providing improved catalytic activity compared to conventional methods. Multi-step catalytic sites can dynamically be formed by the liquid metal solution by interaction with the reactant, analogous to biological catalytic systems such as enzymes [36]. Liquid catalysts can efficiently utilize rare metals used in catalysis such as platinum and palladium. Consequently, this improves their sustainability and cost effectiveness [60, 39, 48, 75, 25, 10, 50, 66, 53].

Due to the dynamic nature of liquid catalysts, experimental research of the catalytic mechanisms and structural properties is significantly harder than other conventional catalysts to achieve [31]. A potentially complementary solution is computational simulation of these liquid metal systems, but accurate and reliable methods such as DFT is computationally expensive when applied to perform ab initio molecular dynamics (AIMD). Therefore, the timescales and/or the system size being simulated are limited [5]. In contrast, classical molecular dynamics uses force fields that are significantly less computationally demanding, but this does come at the cost of significant chemical accuracy, which is an unacceptable trade-off for catalytic systems. However, this problem can be addressed by a new class of force fields that uses machine learning (ML), specifically a (equivariant) message passing neural network (MPNN). A MPNN learns a potential function that informs a force field based on reference highly accurate DFT calculations [51]. Using these machine learning models, large-scale molecular dynamics can be performed with accuracy comparable to DFT. These molecular dynamic simulations can provide an understanding of the structural and catalytic mechanisms behind liquid-metal catalysts. This will allow further research to improve current liquid metal catalysts or design new ones for existing catalytic processes. This in turn could be a step towards making the chemical industry more sustainable.

In this thesis the the structural and catalytic workings of a gallium SCALM system will be investigated. This will be done in two steps. The first step starts with the choice of architecture regarding MLIP is done based on benchmarking of computational efficiency and accuracy. This step continues on with the finetuning of a MACE model by using DFT data. In the second step this model is used to run Molecular Dynamics (MD) simulations. These MD simulations are then analyzed using various techniques to find dynamic properties and behaviors.

2

Theory

In this section a short overview of the theory and background relevant for this project will be given.

2.1. Supported Catalytically Active Liquid Metal (SCALM)

liquid catalysts were poised to display desirable properties such as increased selectivity and increased longevity due to coking resistance. In the first approach metal complexes were dissolved in organic or aqueous solvents [1]. Further development gave Supported Ionic Liquid Phase catalysis (SILP) in which metal complexes were dissolved in a liquid organic salts [43, 42]. Unfortunately the liquid organic salts set temperature limitations due to reduced stability at higher temperatures. Later advancement in the field of catalysis gave Single Atom Catalyst (SAC), this novel type of catalyst uses single atoms active sites on a support material. SAC showed widespread application in catalysis because of its many advantages [38]. Taccardi et al. combined the ideas of SAC and liquid-phase catalysis by dissolving noble metals in a liquid metal matrix and thereby introducing SCALMs as a new catalytic technique. Specifically, Taccardi et al. used gallium as a liquid matrix due to its low melting point and its ability to dissolve transition metal atoms. They first investigated palladium as a noble metal due to the large amount of research already performed into Ga-Pd InterMetallic Compounds (IMC) and its applications in catalysis. This Ga-Pd SCALM system was then shown to provide catalytic activity for butane dehydrogenation and showed excellent long term stability, selectivity, resistance to coking and active metal agglomeration resistance [60]. Further investigation of SCALMs focused on other active metals such as Pt and Rh, methodologies for producing SCALMs, coking, support influence/choice and other applications for SCALMs [21, 7, 73, 59, 37, 64, 48, 67]. One major application of SCALMS is propane dehydrogenation and will therefore be the reaction of interest in this work [45, 66]. Computational modeling can support this work by giving insight into the mechanistic workings of a SCALM system.

Two papers are of particular importance to this work, namely *Dynamic Activation of Ga Sites by Pt Dopant in Low-Temperature Liquid-Metal Catalysts* by Lambie et al. and *Atomic-Scale Dynamics at the Interface of Doped Liquid Gallium: Contrasting Effects of Gallium Oxide and Vacuum* by Steenbergen et al. These papers serve as a basis for this work. In *Dynamic Activation of Ga Sites by Pt Dopant in Low-Temperature Liquid-Metal Catalysts* by Lambie et al. they performed Ab Initio Molecular Dynamics (AIMD) simulations on a (192 atom) Ga-Pt SCALM nanoparticle to show structural properties and they suggest the catalytic activity of a SCALM is provided by the formation of a persistent geometric feature and electronic environment around the subsurface Pt atom. They also disproved the theory that Pt would diffuse to the surface to act as a catalyst as was thought before. *Atomic-Scale Dynamics at the Interface of Doped Liquid Gallium: Contrasting Effects of Gallium Oxide and Vacuum* by Steenbergen et al. expands on this work by simulating a gallium slab doped with metal solutes using the Machine Learning Force Field (ML-FF) implementation of VASP. They showed the behavior of the metal solutes to move away from the (vacuum and oxide) surface and settle in the subsurface, except for bismuth which moved towards and settled on the vacuum surface. Although these papers provided a good basis for the workings of SCALMs many questions still exist about the formation of the catalytic site and solute dynamics.

2.2. Molecular Dynamics (MD)

$$\mathbf{F} = m\mathbf{a} = m\frac{d\mathbf{v}}{dt} = m\frac{d^2\mathbf{s}}{dt^2} \tag{2.1}$$

Molecular dynamics integrates Newton's equations of motion (Equation (2.1)) to evolve any arbitrary atomic system over time. To get the forces of a system, we can use the following equation:

$$\mathbf{F} = -\nabla E \tag{2.2}$$

To get the energy of a system a lot of methods exist. They can be categorized in 3 groups: Ab Initio, Machine Learning Interatomic Potential (MLIP) models and classical force fields. Classical force fields that accurately describe the interaction between gallium atoms and between gallium atoms and solute atoms do not exist in literature and thus are not explored in this report any further.

2.2.1. Ab Initio Molecular Dynamics

Ab Inito means from beginnings and that is where this method starts from the fundamental equations of quantum mechanics such as the Schrodinger equation. Unfortunately the Schrodinger equation can only be analytically solved for the simplest of systems. Luckily, an approximate solution can be found using the self-consistent field method as described by Hartree and Fock. In this method the wave-function constants are initially guessed and iteratively optimized by solving the system of one-electron equations set up by applying the variational method. In the context of computational chemistry the variational method entails changing the wavefunction constants so a lower energy orbital configuration can be found. As this field evolved it took shape in Density Functional Theory (DFT). This reformulation of the method to find the many-electron wavefunction as described by Hartree and Fock brought with it advancements in computational efficiency that made it viable to solve these equations for practical (model) systems. The creators of DFT Kohn and Sham optimized the electron density instead of the wavefunction as was done before. This electron density found by DFT defines the ground state properties such as the energy and can be calculated [33]. Equation (2.4) shows an example of an orbital expression of the energy as defined in DFT [14].

$$E[\rho(\mathbf{r})] = \sum_i^N \left(\left\langle \chi_i \right| -\frac{1}{2}\nabla_i^2 \left| \chi_i \right\rangle - \left\langle \chi_i \right| \sum_k^{nuclei} \frac{Z_k}{|\mathbf{r}_i - \mathbf{r}_k|} \left| \chi_i \right\rangle \right) + \tag{2.3}$$

$$\sum_i^N \left\langle \chi_i \right| \frac{1}{2} \int \frac{\rho(\mathbf{r}')}{|\mathbf{r}_i - \mathbf{r}'|} d\mathbf{r}' \left| \chi_i \right\rangle + E_{xc}[\rho(\mathbf{r})] \tag{2.4}$$

Where:

$$\rho = \sum_{i=1}^{N} \langle \chi_i | \chi_i \rangle \tag{2.5}$$

As stated before this energy can be used in Equation (2.10) to find the force which can be used to evolve the system.

2.3. Machine Learning Interatomic Potentials

As the name suggests Machine Learning Interatomic Potential (MLIP) models use machine learning techniques to derive interatomic potentials from a given system. This allows for more efficient computational scaling and more efficient use of modern hardware such as GPUs. To give a short overview of the following theory, MLIPs encode the local structure of each atom in an array of embeddings which are passed around and combined. Finally all embeddings can be mapped to certain values such as energy, force and stress using a readout function. These forces can be used in MD to move the simulation a step further. DFT and classical methods use functions or other algorithms based on fundamental or higher-order theoretical foundations, this is however not the case with MLIP models. They are made to match to some ground truth such as experimental or DFT data.

2.3.1. Neural Networks (NN)

A Neural network is a type of machine learning model that can be represented by a graph structure (Figure 2.1).

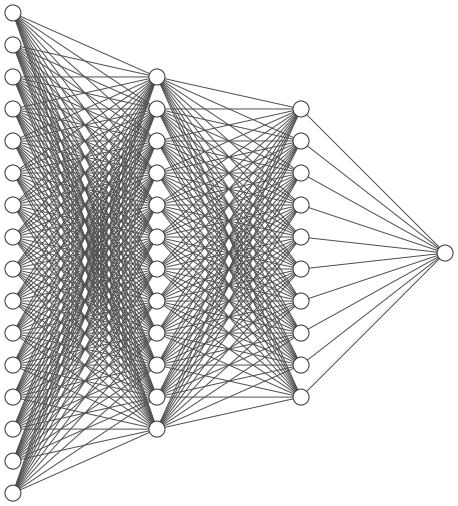


Figure 2.1: The graph representation of a neural network

In this representation, the nodes (also called neurons) contain values and the edges describe how the value of a node is constructed. Each node of the next layer is constructed by summing the products of the learnable weights and the previous node values and adding a bias. Then an activation function is applied on the calculated value, introducing non-linearity in the model. This structure can be written mathematically for the entire next layer of neurons as seen in Equation 2.6.[1]

$$x^{(1)} = \sigma\left(\begin{pmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,m} \\ w_{1,0} & w_{1,1} & \dots & w_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,0} & w_{n,1} & \dots & w_{n,m} \end{pmatrix} \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{pmatrix}\right) \tag{2.6}$$

The weights and biases for each layer can be optimized by minimizing some loss function. A loss function quantifies the difference between the expected output and the output given by the model. The most common loss function is the Root Mean Square Error ($RSME = \sqrt{(\frac{1}{n}\sum_i (X_i - \overline{x}))}$) but the data and application informs the appropriate loss function. This loss is then used by an algorithm called backpropagation which calculates the derivative with respect to all learnable weights and biases (also called parameters). Then based on the derivatives and a certain learning rate the new values for all parameters are computed. To better explore the parameter space and find the most optimal parameters in as few training cycles as possible, more advanced optimizers are commonly used. One of these optimizers is called the Adam optimizer which uses a momentum-based moving average of gradients [32].

2.3.2. Graphs, Graph neural networks and message passing

An inductive bias in machine learning is a preconceived direction or form a model can give answers in brought on by assumptions encoded into the model, forms of data used and model architecture. This means that for example, when a linear model architecture is used only linear behavior can be trained or learned by the model. When this is done accidentally it can give nonphysical behaviors and a bad performance, but when done on purpose the essence of a problem can be captured and this can result in a very efficient model. This can be seen in the application of graph neural networks on molecular

[1] Note that the σ function is taken as example activation function, this can be any activation function such as ReLU, tanh, etc. [27]

systems. Here, the atoms are encoded as nodes on a graph and the bonds/interactions between atoms are encoded in edges [6].

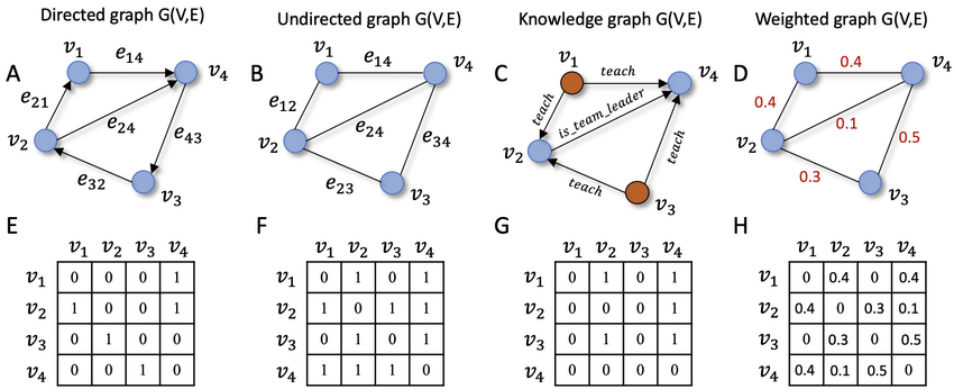


Figure 2.2: Example of different types of graphs with their respective adjacency matrix [69]

Graph Neural Networks (GNN) use as the name suggests graphs as input data compared to older types of neural networks, which can only use 1D (Multi Layer Perceptron) or 2D arrays (Convolutional Neural Networks). Graph data structures can model a wide variety of things in which the relationship between elements is significant. Common examples are websites (PageRank), social networks (linkedin) or road networks. Each node and edge can hold its own information. For the case of road networks, each node can represent a city with a specific name, and each connection can be a direct road between two cities with a certain distance between them. Edges can also be directed, meaning they only go one way or undirected which does not imply a direction and only serves to connect two nodes. Graphs can be represented using an adjacency matrix, where all nodes are on each axis and binary entries denote connections, as shown in Figure 2.2. A graph neural network can have three types of goals, namely node prediction, edge prediction and global prediction. First each known part is embedded then using these embedding graph operations can be done. A simple GNN in the example of node prediction could pool all edges of each node to form the nodes. This only uses the graph structure superficially so a more advanced technique is message passing. Message passing is generally defined by gathering all neighboring embeddings, aggregating all messages and passing the messages through a update function. Glimer et al. show these steps elegantly in Equations (2.7) to (2.9).

$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}) \tag{2.7}$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \tag{2.8}$$

$$\hat{y} = R(\{h_v^T | v \in G) \tag{2.9}$$

Here h_v^t is a embedding of node v at time step t. $w \in N(v)$ means all nodes connected to node v, e_{vw} are the edges and M_t and U_t are a learnable functions. Usually U_t is a Multi layer perceptron (MLP). This example shows the aggregation of all messages using a sum but any aggregation function can be used such as max or mean. Finally some wanted value can be found using readout function R. This framework has many variations using convolution or attention mechanisms [26, 54, 74, 68, 28]. These steps can also be see more schematically in Figure 2.3.

2.3.3. Foundational models

Message passing neural networks are the at the core of the latest MLIP models such as MACE, Orbv3 and NequIP. Despite this commonality large architectural and practical differences still remain. The older architecture NequIP uses equivariant embeddings, 2 body messages and 4 to 6 message passing layers. The newer MACE uses equivariant embeddings (ACE), n-body messages and k message passing layers. Although the architecture is flexible the significant benefits of the MACE architecture

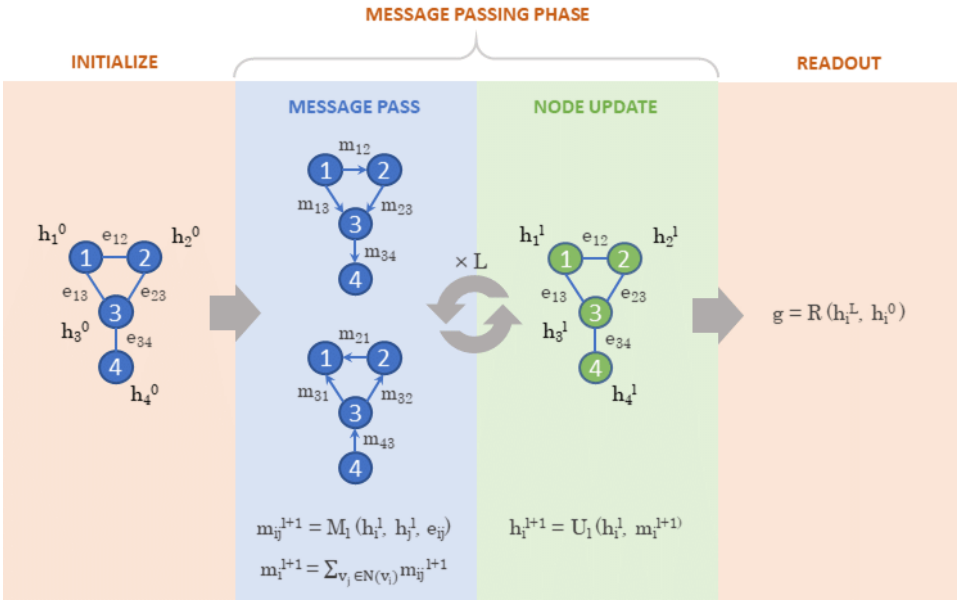


Figure 2.3: Steps of a message passing architecture[44]

are seen when using 4 body order messages and 2 message passing layers. Finally Orbv3 uses no architectural measures to achieve equivariant messages or conservative forces in the direct models. The conservative models still compute the force using a multi layer perceptron model but they are trained with *equigrad* which uses a roto-equivariance-induced regularization scheme developed by orb.

Equivariant and invariant

an observable is invariant if it does not change under translation or rotation of the observed system. Energy is invariant, the internal coordinate system does not matter.

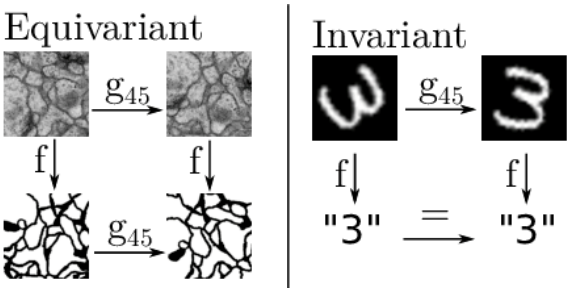


Figure 2.4: A visual explanation of the difference between equivariant and invariant functions [41].

An observable is equivariant if it changes with the system under translation or rotation of the observed system. Internal forces are equivariant, when the system is rotated or translated all forces do the same.

This physical basis can be used as an inductive bias in MLIP models to improve stability and accuracy. Practically translational equivariance is easily achieved by only considering relative distances between atoms. To build rotational equivariant embeddings more involved mathematical formulations must be used. Examples of these formulations are Smooth Overlap of Atomic Positions (SOAP) [3], Moment Tensor Potential basis functions (MTP) [61] or Atomic Cluster Expansion (which is the basis for MACE) [19, 17].

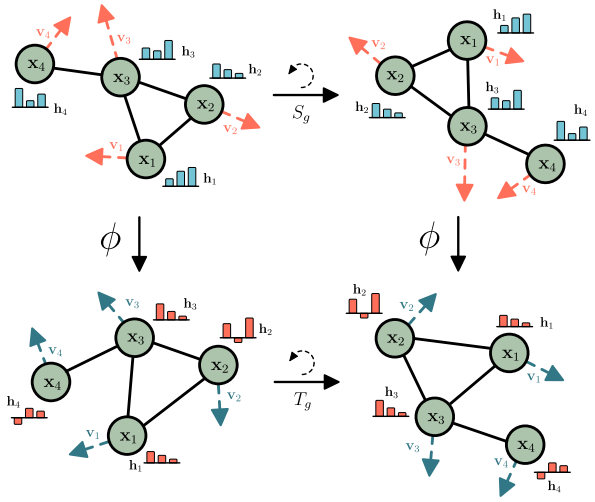


Figure 2.5: Rotation equivariance in graph networks [55]

Conservative forces vs non-conservative forces

MLIP models can compute forces using two different methods. In the conservative method we first pass the node embeddings to a readout function (MLP) which predicts the energy of each atom. These energies can be seen as points on a potential energy surface (PES). The force is then computed as the negative gradient as described by Equation (2.10) on this PES.

$$\mathbf{F} = -\nabla E \tag{2.10}$$

The second non-conservative method uses a different readout function to directly predict the force from the node embeddings. This means forces do not need to follow the fundamental Equation (2.10) which can cause non-physical forces. This concern is tempered by the fact that the ground truth on which all models is trained does respect this relation. Care must be taken in scenarios that can exacerbate these flaws, for example large timescales can cause errors to accumulate or small datasets can cause the need for far out of distribution predictions which can be unstable.

3

Methods

3.1. Digital architecture

A wide variety of digital tools were utilized. To start at the top, WSL2 was used as the OS for this project for two reasons. Firstly this was required for the addition of ORB as a possible model architecture and secondly this prevented possible incompatibility issues between DelftBlue and the system on which the code was developed. DelftBlue was used to run all heavy workloads such DFT calculations, training the MLIP models, running batched MLIP inference, etc. [15]. In order to accelerate the development cycle mid-range consumer hardware (CPU: 5700X3D, GPU: 5070 Ti, RAM: 32 GB DDR4) was used to test and initially run the heavy workloads. For versioning and sharing code Github was used, larger files such as DFT datasets required the use of the TU Delft OneDrive. Python is the main coding language due to its accessibility and its widespread use in research (MLIPs, ML, data science, etc.). Each MLIP model architecture has been separated in its own environment because of conflicting dependencies. Although the standard Python virtual environments can work fine, it is highly recommended to use Uv. Uv is significantly faster in resolving the large MLIP environments and its aggressive caching reduces the amount of duplicate installations of large ML dependencies such as `torch` [47]. The separation of these different modules does increase the complexity of the code because only one environment can be active at once, which means that interacting with multiple models requires running child processes with their respective virtual environments. Other tools include OVITO [58] for visualization and Overleaf for written reports and presentations.

The codebase can be split up into 4 sections: geometry, benchmarking, dynamics and HPC. Geometry includes a script called **create_Ga_NP.py** which can be used to generate gallium nanoparticles of differing sizes and a script called **structural_cluster.py** which clusters a dimensionality reduced collection of descriptors of an MD trajectory generated by MACE. Benchmarking includes the **benchmark_comp_parent.py** script, which is used to test the computational efficiency of various models and it includes the **benchmark_FF_parent.py** script, which can be used to recompute a ground truth MD trajectory using a wide array of models. Hardcoding model paths is prevented by using a model config file. In order to handle the large amount of data a custom HDF5 file implementation was used to efficiently store and process large force field benchmarks. The dynamics package includes the **run_dynamics_parent.py** script which can be used to run MD and run dynamics analysis. Finally the HPC directory contains scripts and Slurm submit scripts for running the heavy workloads and DFT on a HPC system (specifically Delftblue). A simple overview of the locations of all scripts can be seen in Figure 3.2. For more info on the parameters and features of each script see Appendix D.

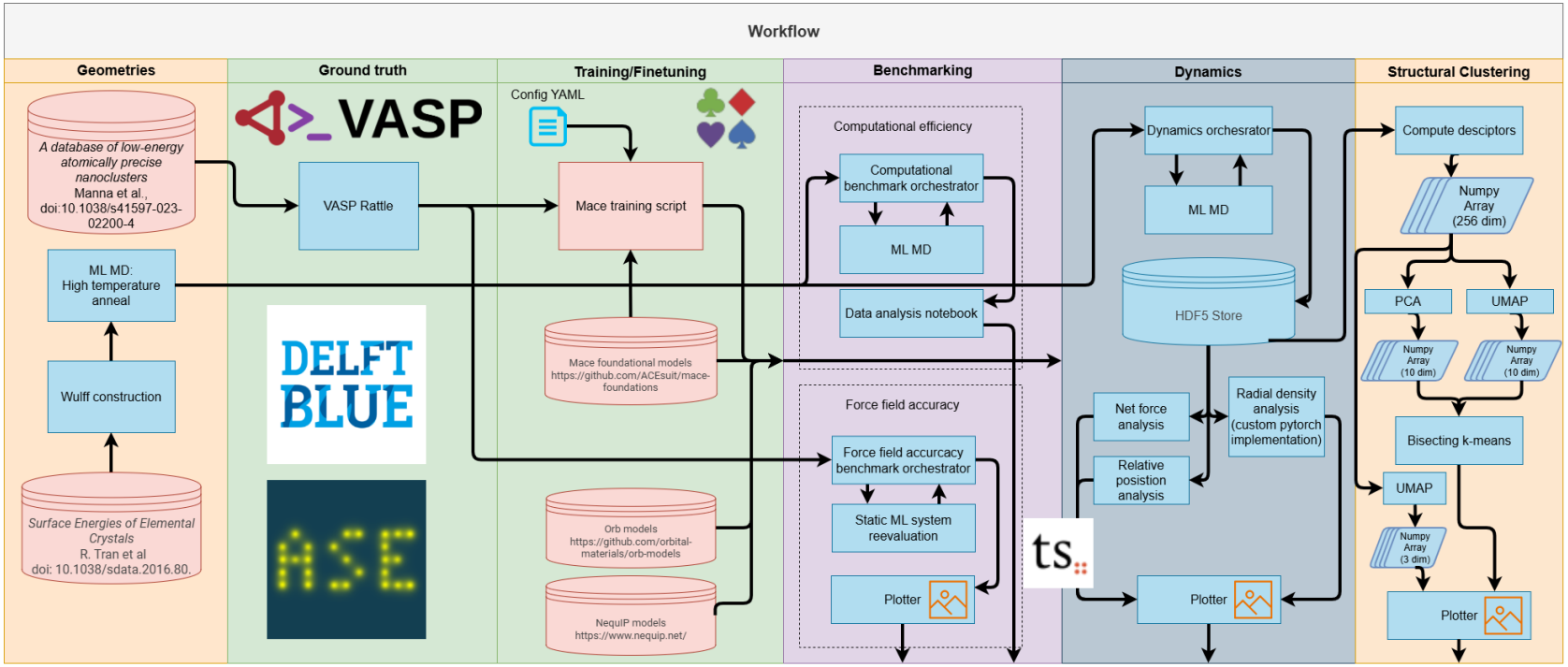


Figure 3.1: A simplified representation of the latest computational workflow. Each section is colored based on the location in the codebase; orange is the geometries directory, green is the HPC directory, purple is the benchmarking directory and gray is the dynamics directory. Block color denotes source; red is external, blue is created for this work.

```
Atomic_intelligence_gallium_SCALMs/
├── dynamics/
│   └── run_dynamics_parent.py
├── geometry/
│   ├── create_Ga_NP.py
│   ├── structural_cluster.py
│   └── add_organic_molecules_to_NP.py
├── benchmark/
│   ├── benchmark_comp_parent.py
│   └── benchmark_FF_parent.py
└── HPC/
    ├── VASP_nanoparticle_rattler_v2.py
    ├── VASP_nanoparticle_organic_mol.py
    └── dataset_packager.py
```

Figure 3.2: A simplified version of the folder structure.

All scripts except the computational benchmarking are implemented with 2 backends: ASE calculators [30] and TorchSim [12]. Code was first written for ASE and later adapted for TorchSim for its autobatching feature. This provides a good fallback because not all models are implemented in TorchSim and if TorchSim is unable to accurately assess the VRAM usage it can crash.

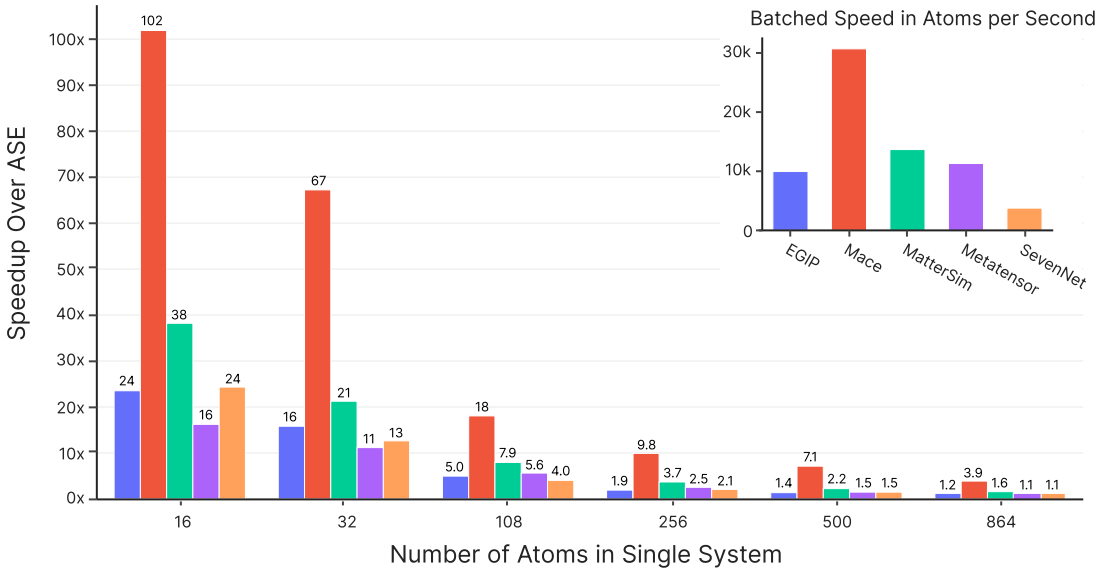


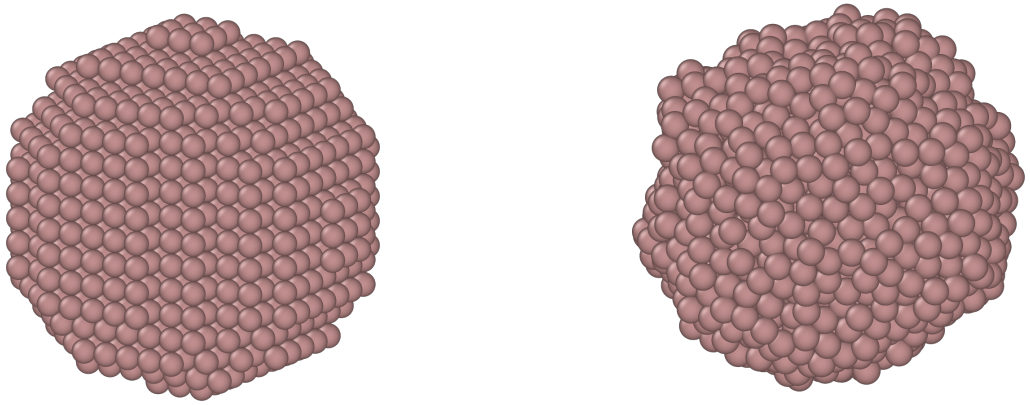
Figure 3.3: Speedup using TorchSim batched inference on a 80 GB VRAM A100 adapted from the TorchSim github [13]

TorchSim is a relatively new MD framework and is interoperable with frameworks such as ASE and pymatgen which makes it adaptable to existing code. It has an easy high-level interface comparable to ASE, but it is build on PyTorch instead of numpy. It is currently focused on 3 tasks (called runners), namely **Static**, **Integrate** and **Optimize**. Static evaluates a system statically, integrate runs different kinds of MD and Optimize relaxes atomic positions. The advantage of TorchSim over ASE is the autobatching feature. Autobatching is the automatic batching of a set of geometries (called State in

TorchSim) for efficient MLIP inference on a GPU. This can be efficiently done due to the architectural differences between CPUs and GPUs. In essence, autobatching efficiently utilizes all available GPU resources (primarily VRAM), where serial computation can under-utilize the GPU causing significantly decreased speeds. This can be clearly seen in the speed-up graph in Figure 3.3. Small atomic systems require a small amount of VRAM and can therefore be batched in greater numbers compared to large systems, resulting in larger speedups. Figure 3.3 also shows that MACE especially handles batched inference rather well. TorchSim therefore reframes the computational efficiency benchmark to rely more on VRAM usage as this is now the bottleneck at all system sizes.

3.2. Geometry

Starting geometries were needed for the creation of additional ground truth data. These starting geometries were obtained from Manna et al.[40]. The size range of these clusters is limited to a maximum of 55 atoms; this rather small range is insufficient to investigate the gallium nanoparticle structural dynamics. To solve this a computational workflow was developed to generate valid molten metal nanoparticles.



(a) Before **(b)** After

Figure 3.4: The effect of annealing is shown; On the left a solid 1568-atom Ga nanoparticle constructed using Wulff construction and on the right the same nanoparticle but now molten after annealing.

To create arbitrarily large nanoparticles the Wulff construction function of ASE was used in the **create_Ga_NP.py** script. For Wulff construction surface energies were needed. These values were found on the Crystalium website provided by Tran et al. [63, 62, 72]. The crystal structure and lattice constant distances were taken from the stable bulk form of the element in question. Using a MLIP model the geometry was first converged at the target temperature ($T > T_{melt}$) with a predefined simulation time of ≈2-5 ps, this was followed by a longer MD simulation of ≈ 6-10 ps at high temperature ($T_{vaporization} >> T >> T_{melt}$) and finally was completed by another MD simulation of ≈2-5 ps at the target temperature. The annealing steps were performed to increase the speed at which the nanoparticle forgets its crystalline structure. When a melted bulk geometry needed to be generated a similar procedure was followed. The main differences are the use of a crystal structure from the materials project and the use of the NPT ensemble (P=1atm) to allow the cell to change while melting.

More relevant to the catalytic dynamics of gallium SCALMs is the addition of reactants. In **add_organic_molecules_to_NP.py** organic molecules can be added close to the nanoparticle surface to simulate a real catalytic setting. To determine the surface of the nanoparticle the Euclidean distance from the Center of Mass (COM) is computed for all atoms, then this list is sorted and the n highest atoms are averaged to determine the surface. By default n is set at 10 to suppress outliers while still containing as many atoms inside the hypothetical sphere. This assumes an approximately spherical nanoparticle, more advanced techniques to approximate the surface exist that do not need this assumption. Examples include the alpha-shape method [57] or gaussian density method [34] as implemented by OVITO, but these require significantly more (GPU) resources. Other possibilities include a 3D convex hull as implemented by Scipy. Finally, the currently implemented method could be extended to instead of using only 1 radius for the entire nanoparticle subdividing a sphere and defining a radius for each face.

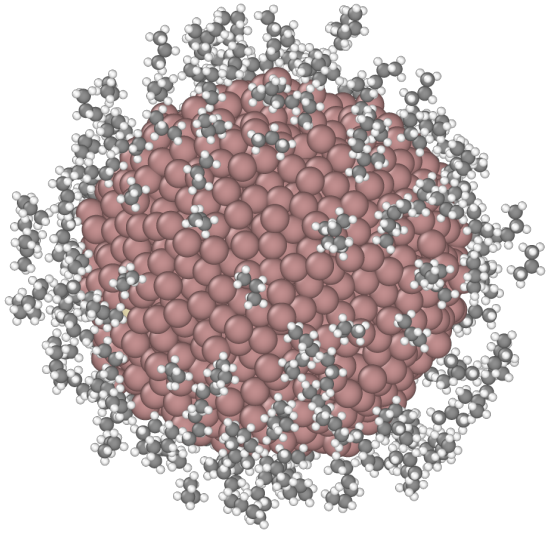


Figure 3.5: 1568 atom Ga Pt SCALM including 200 propane atoms.

To place the organic molecule, a random direction is chosen and extended to the radius of the hypothetical sphere containing the nanoparticle plus a predefined distance (by default 2 Å). Then an organic molecule is translated to this position and (if enabled) randomly rotated in 3D. To prevent nonphysical structures the pairwise distances between the to be placed atoms of the organic molecule and all already defined atoms in the systems are computed, if the smallest computed interatomic distance is below a predefined threshold (by default 2 Å) the placement of the organic molecule is deemed invalid and the organic molecule is translated by a predefined distance (by default 0.5 Å) away from the nanoparticle surface in the direction defined before. This is repeated a predefined number of times, when this number is exceeded this direction is deemed invalid and a new direction is chosen.

The optimal balance between these parameters is defined by the organic molecules used (the defaults are set for C3H8) and the user's preference. For example, when a dense layer around the nanoparticle is required, the distance increment and max attempts to place the molecule should be reduced as much as possible, respecting that the surface of the nanoparticle can be saturated with less than the defined amount of organic molecules to be placed. Multiple organic molecules can be defined in the script input, when this is the case the script splits the total organic molecules to be placed evenly over the defined organic molecules. The script also requires an input nanoparticle in ASE readable format, when a trajectory is given a random frame will be selected. This can for example be used to generate multiple catalytic systems with similar nanoparticles by using a trajectory generated by MD. An example of a SCALM system created using the tools described in this section can be seen in Figure 3.5.

3.3. HPC

Due to the high computational requirements, DFT calculations were run on DelftBlue (SLURM HPC system). Again the ASE interface was used to start the DFT calculators. Many scripts were deprecated when the project progressed; only the scripts of the latest workflow will be discussed.

VASP_nanoparticle_rattler_v2.py
This script was used to generate the rattled structures and run DFT on them. When a trajectory is given it selects a random one to rattle. Additionally an alloy element and alloy fraction can be defined as arguments, which randomly replaces atoms based on these arguments. To prevent large forces during the rattling the provided structure is optimized using the ASE BFGS geometry optimization implementation before the rattling of the system. This is important when the structure provided by the user is out of equilibrium or gets out of equilibrium by alloying. By default, the structure is not preconverged because it is expected that the user provides a low-energy structure; only when asked for or when the structure

is alloyed will the script preconverge the structure. The rattling is done using the rattle function invoked on an ASE Atoms object. This function displaces the atoms randomly based on a normal distribution defined by the user with a standard deviation. Depending on the system, the standard deviation needs to be adjusted; by default it is set at 0.1 Angstrom. In order to facilitate the tuning of the standard deviation and for general monitoring logging of the energy, absolute mean force, mean force, max force and rattle time is implemented.

VASP_nanoparticle_ratteler_organic_mol.py
This script serves a very similar purpose to the previous one, but is extended to include the possibility of placing organic molecules around a (approximately) round nanoparticle. The implementation is similar to the one in **add_organic_molecules_to_NP.py** and therefore exposes similar arguments. Note that only the nanoparticle is rattled and is preconverged.

dataset_packager.py
Training/finetuning a MLIP model requires the entire dataset to be split into 3 categories. The first one is the train dataset, which contains a significant portion of all data. The second is the validation set, this smaller dataset is used during training to give a measure of the improvement of the model. The final dataset is the test set, this one is not included in the training and is only used after training to benchmark the model. This script combines all files generated by the previous two scripts and splits them into the three different categories based on a user-defined ratio. Frames with large forces are discarded using a force threshold of (by default) 5 eV/Å, due to the possibility of the frame containing invalid forces. An option exists to define all elements that must be in the test and valid set. This is to get the most representative test and valid set as possible, but it can skew the ratio between train, valid and test datasets.

3.3.1. DFT (VASP) computational details
Accurate or congruent data is paramount for the training of MLIP models. Lax convergence settings and even certain DFT calculators can prove disastrous of MLIP training. CP2K was the first choice for DFT calculator as it is fast, scalable, open, free and is able to simulate non-periodic systems due to its Gaussian Plane Wave (GPW) approach. Unfortunately the data generated by CP2K proved to be unsuitable for MLIP training in agreement with the findings of Kuryla et al. [35]. More information about the effects of the CP2K data on the benchmarks and the reason why VASP was chosen in Appendix B.

Using VASP the k-points were set to (1,1,1) because a non-periodic system was rattled. The convergence criteria for the self-consistency cycle was set at 1E-7 (EDIFF), the plane wave energy cutoff was set at 400 eV (ENCUT) and a parameter was set so spin DFT would be performed (ISPIN=2).

3.4. Benchmarking
3.4.1. Computational time/resources
The computational time and resources utilized by each model were investigated by running a set number of MD steps with each model on a wide array of system sizes. When a model is running, an extra thread is launched to measure the resource utilization over time. This data is saved to a csv file, which can be used to visualize the data. In order to accommodate multiple virtual environments for different model architectures the script is split in two. The parent script runs a child script within the correct environment to prevent clashing dependencies when using multiple model architectures. An additional script called **benchmark_comp_plot.py** can be used to plot the data (script can be called using a typer interface)

3.4.2. Force field accuracy
The force field benchmark is split up into 3 different files. The first file is the parent this is the main script and orchestrates which model needs to run and does various tasks around the actual force field benchmark. It also includes the functions to recompute a given system. It does not use this function itself; it delegates this to child processes, which are run using specific virtual environments that correspond to the virtual environment with the dependencies for this model architecture. The models that are run are defined in a model config yaml file that includes the python executable path to the dedicated virtual environment for each model architecture and a path to specific models. This

makes it possible to add custom models and test a wide range of different model architectures. For long term storage a custom compact HDF5 data structure was created. Unfortunately this hierarchical data structure cannot be used for plotting to solve this a flat HDF5 file is compiled.

3.5. MLIP training

Two approaches to obtaining an accurate MLIP model exist. The first approach involves training a model from scratch; in this case all model weights are randomly initialized following a predetermined distribution. In the second approach, a model is fine-tuned. In this case an existing (foundational) model is used and is trained using a lower learning rate to tune the model to a specific problem. In both approaches the dataset and learning hyperparameters are important, but have a greater influence on the respective approach. The dataset is more important to the scratch training because the model has seen no other data and all higher-order concepts need to be extracted from this data. In comparison finetuning is done on top of a dataset that is significantly bigger than the one used to finetune, so general concepts are already captured, and only the fine detail of your problem needs to be trained. The training hyperparameters are especially important to the finetuning approach because the existing weights need to be maintained as much as possible as to prevent catastrophic forgetting [4, 65].

A new version of finetuning is implemented by MACE called multiheaded replay finetuning. This means a foundational model is trained on multiple training sets at once. For each head (readout function) a training set is used that is calculated with similar DFT settings: exchange correlation functional, convergence, etc. The replay training set is a subset of the training set used to train the foundational model. This should give guard rails against catastrophic forgetting because the original dataset is replayed at the same time as the new data.

MACE includes a training/finetuning script that was used to train/finetune MACE models. The rattle data first needs to be compiled into 3 different (xyz) files: train, test and val. Then it needs to be preprocessed. For the preprocessing a E0 file needs to be given. This E0 file is a JSON that has the energy of each element computed as an isolated atom. This is done with spin-polarized DFT to get the correct energies. The replay set also needs to be compiled with a certain number of samples and a ratio between the influence of the new and the replay set. For these values, the MACE recommended amounts of 30000 samples and a 10:1 ratio were used.

When all data is preprocessed a config yaml must be defined for the finetuning or training using the provided script. Generally, default hyperparameters were used except for a few: learning rate, max epoch and batch size.

Although no extensive hyperparameter sweep was performed due to computational resource limitations, multiple models with different datasets were trained so general qualitative inferences can be made about the balance between learning rate and max epoch. The default learning rate for multiheaded replay fine-tuning is 1E-4 which is an order of magnitude lower than the training default and is set this low in order to prevent catastrophic forgetting as mentioned before. Higher lr values than 1E-4 were observed to destabilize learning and significantly increase the replay head error metrics. To make sure the training was stable the learning rate was reduced to 2.5E-5 with increased epochs.

3.6. Dynamics

Similar to the benchmarking the dynamics script is split into a parent that orchestrates the running of the models and hosts the important functions, a child script that actually runs the MD using the correct environment/dependencies and a plotting script that can analyze and plot the data. In the following sections the implementation of each analysis is explained.

3.6.1. Radial/Pair density function

As the name suggest the radial density function defines the density (e.g. per volume) of atoms around atoms in a system. This can be done for all different atoms at the same time, also called full RDF or this can be done for a specific pair such as Ga-Pt; in that case it is called a partial RDF. Essentially the radial density function defines the local structure of a given system. For regular systems like crystals

only a few radial values have density due to the regularity; liquid systems however are disordered and do not have a regular structure. This gives liquid systems continuous radial density functions. The scattering function that can be found using X-ray diffraction can be transformed into a Pair Density Function (PDF) using a truncated Fourier transform [9]. This way, computational simulations can be experimentally verified. Another feature of the radial density function is that due to normalized $g(r) \to 1$ as $r \to \infty$. A variation on radial density functions is a pair distribution function (PDF), which does not necessarily require normalization so $g(r) \to 1$ as $r \to \infty$. The PDF is used in this project because the non-periodicity of the nanoparticle means the density at $r \to \infty$ is not well-defined. For monometallic nanoparticles the PDF equation can be written as Equation (3.1) [52].

$$g(d) = \frac{1}{N(N-1)} \sum_{i}^{N} \sum_{j \neq i}^{N} \delta^{'}(r_{ij} - d) \tag{3.1}$$

Where d is the interatomic distance, N is the number of atoms in the system, $\delta^{'}$ is defined as:

$$\delta^{'} = \begin{cases} 1 & d - \epsilon \leq r_{ij} \leq d + \epsilon \\ 0 & otherwise \end{cases} \tag{3.2}$$

And ϵ is the distance from the bin midpoint to the bin edge, giving bin widths of 2ϵ.

TorchSim saves the positional data of all frames and atoms in one tensor. This positional data is fed into a custom PyTorch PDF function to calculate the PDF of large datasets efficiently. First, the position difference between all atom locations is taken and the L2-norm is applied to get the magnitudes or distances of all relative positions. In the second step the bin edges are defined as a simple linear interpolation between 0 and the predefined maximum radial distance with an amount of points equal to the predefined number of bins plus 1. All position difference distances can then be binned using the defined bin edges. To make sure self-interaction is removed, the first bin is set to 0. The binned values are not yet density; they need to be normalized using the volume of the shell between consecutive bin edges. To be able to compare the density between different system sizes and trajectory lengths, the binned counts are normalized using the number of frames and atoms in the system. Finally, the radial values are computed as the midpoints between all bin edges.

3.6.2. Atomic trace

In this work we define an atomic trace as the tracing of a certain set or group of atoms over time. To get the atomic trace plot a custom function was made. The function starts by computing the Center of Mass (COM). This is computed by taking the weighted average of the positions using the atomic masses as weights. Now, all positions can be made relative to the COM by subtracting the COM position from all the atomic positions. Because it is a reference we take the COM of the first frame. Then using the provided mask the atoms that are traced are selected. The purpose of this plot is to see the relative location of the traced atom from the surface and the COM over the trajectory. We therefore need to define the surface of the nanoparticle but due to the fluxionatlity of liquid Gallium nanoparticles the surface is not fixed and not constant in each direction. One adaptive solution is to project all position vectors on the position vectors of each traced atom and take the mean of the n largest projected vectors. practically this means the n furthest atoms in each solute direction are found and averaged to get an approximate surface. A too low value of n gives a very noisy surface and suffers from outliers and a too high value of n gives an overly damped representation of the surface and lowers the surface position into the nanoparticle. In this code the value of n is set to 5, but other values such as 20 are also used in literature [56].

3.6.3. Diffusion constant

The diffusion constant is a measure that describes the dynamic motion of particles. Using the Einstein relation (Equation (3.3)) and a numeric way to calculate the MSD from the position data generated by an MD simulation (Equation (3.4),) we can find the diffusivity constant D. The random nature of molecular systems means we can only observe this effect at longer timescales. This timescale is called the linear

regime. Because of the random (thermal) noise it is common to fit a linear line using least squares, where the slope is equal to $6D$. Rewriting the equation gives $D = \frac{slope}{6}$, which can be used to get the diffusivity constant [29, 56]. Due to the confined nature of a metal solute in a nanoparticle the MSD will saturate and the linear relation will be lost. This means to get an accurate MSD a timescale must be chosen to start high enough to be in the linear regime and end before the MSD saturates. This saturation time depends on the size of the nanoparticle as larger nanoparticles have more space before the displacement saturates.

$$MSD(t) = 6Dt \tag{3.3}$$

Where:

- D is the diffusivity constant (cm^2/s)
- t is time (s)
- MSD is Mean Square deviation in cm^2

$$MSD(t) = \frac{1}{N_{t_0}} \sum_{t_0=0}^{N_{t_0}-1} |\mathbf{r}(t_0 + t) - \mathbf{r}(t_0)|^2 \tag{3.4}$$

Where:

- t is the time at which the MSD is evaluated (s)
- t_0 is the start time (s).
- N_{t_0} is the amount of time origins.
- $\mathbf{r}$ is the position (cm).

To compute the diffusion as described a custom function was made. This diffusion constant function starts by making a mask based on the target atomic number that is traced for its diffusion constant and applying this mask on the position tensor. Then it constructs the lag time array using a predefined linear time maximum. Then a time origin array is made with all possible time origins and is subsampled to include every other time origin in order to reduce computation. This choice is arbitrary and depends on the amount of available data and computational resources. All combinations of lag times and time origins are used to compute squared deviations (SD). Then we take the mean of these SD values to get the mean squared deviation (MSD). These values are then used to fit a linear line using a least squares method. Large nanoparticles can include multiple solute atoms. The MSD values of the multiple solute atoms can again be aggregated using a mean to obtain a more statistically strong MSD value.

3.6.4. Net force

The net force over frames is calculated as a consistency check. MACE has an equivariant MPNN architecture and computes its forces conservatively so in theory the net forces should be negligible. Models such as ORB v3 direct are not architecturally bound to equivariance or conservative forces as discussed in Section 2.3.3, so in that case this graph should be monitored closely for significant net forces.

3.7. Structural analysis

Due to the rich descriptors of MACE, data analysis can be done on the learned representation of a molecular dynamics simulation. This is done in a script called **structural_cluster.py**. The structural analysis script first computes the descriptors; if requested and available, it uses the cached descriptors instead. Depending on the model this gives a differently dimensional object, but clustering is still not feasible directly even if lower dimensional embeddings such as 128 are used. These high-dimensional descriptors are reduced in dimensionality using PCA or UMAP to 7-15 dimensions [71, 20]. Then, hierarchical k-means clustering is used to label each atom [70]. For visualization purposes the original descriptors are mapped on a 3D space using UMAP and given a color based on their label. To investigate the meaning of the clusters three other visualizations are made. The first plots the positions of all atoms and uses the labels as colors, the second plots the radial distribution of the different labels and

the third plots the PDF of each label. Caching of the descriptors and the data processing pipeline can be enabled using an argument and is done to remove the need to recompute unnecessary parts when tuning hyperparameters.

4

Results and discussion

4.1. Benchmarking

4.1.1. Computational resources

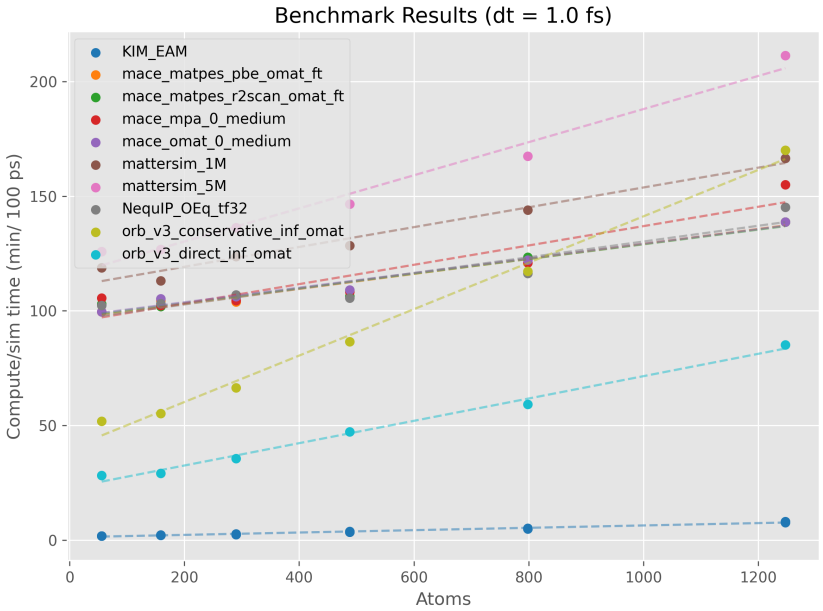


Figure 4.1: The computational efficiency benchmark shown as compute time in minutes versus simulation time in 100 ps. This benchmark was performed using CPU: 5700x3d, GPU: 5070 Ti, RAM: 32 GB DDR4.

To support the choice of model architecture we will look at the computational efficiency and resource utilization. These metrics are important aspects of MLIP models, because they limit the size of the atomic systems and determine the timescales that can reasonably be investigated. All models are MPNN models except KIM EAM, which is a classical force field model with parameters from the KIM repository [23, 22], which is used as a baseline reference. MatterSim, NequIP, and MACE show very similar results only the 5M MatterSim model has a significantly worse computational time and GPU utilization scaling. This is probably due to similar practical implementation of the equivariant MPNN architecture including the use of e3nn to create equivariant embeddings and GPU acceleration for e3nn such as cuEquivariance (from Nvidia) or OpenEquivariance (open source).

In contrast, ORBv3 has suboptimal system scaling but a computational inference time overhead of half or less compared to the other models. The computational resource utilization is very high: GPU, CPU

and VRAM utilization show suboptimal system size scaling. This is unexpected because in the release paper of this model, the model is positioned to be at the front of the performance-speed-memory Pareto frontier [51]. Possibly, an additional package called cuml is silently failing due to software stack problems, as better performance numbers were noted in earlier tests. Because these failings are silent, debugging is especially hard.

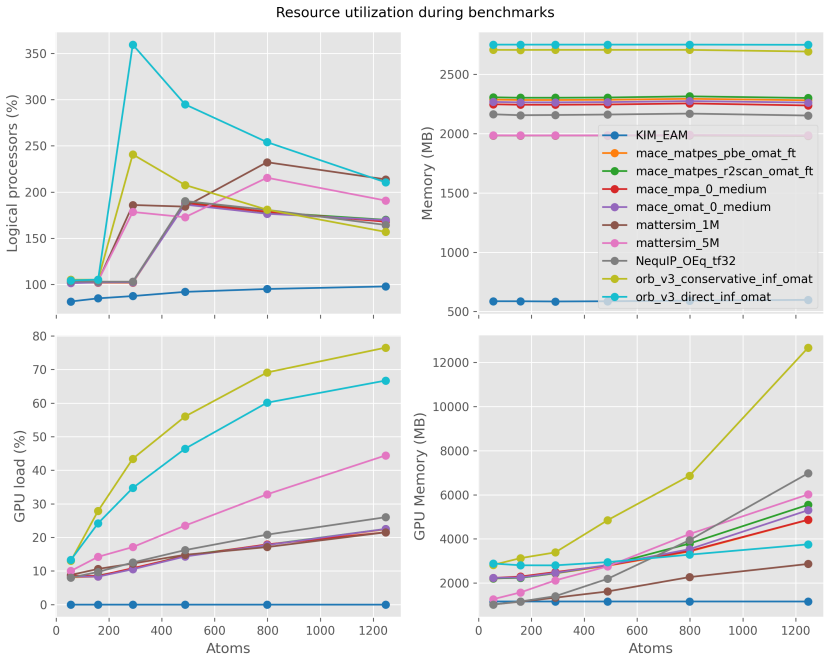


Figure 4.2: The computational resource utilization benchmark. The resource utilization can be seen divided into CPU (top left), RAM (top right), GPU (down left) and VRAM (down right) usage. This benchmark was performed using CPU: 5700x3d, GPU: 5070 Ti, RAM: 32 GB DDR4.

As expected, the RAM usage is observed to be independent of system size due to the usage of the GPU for model inference. CPU usage shows some initially unexpected scaling. This scaling is likely a result of the I/O operations of the script as the amount of data being written to storage depends on inference speed and system size. GPU load scales linearly with system size, only the ORB models seem to be scaling differently. In reality, this is an artifact of the averaging caused by the relatively long preprocessing (for example graph creation) steps from orb where the GPU load is 0. This also explains why the GPU load plateaus below 100%. VRAM usage again is primarily linear with system size, with only the conservative orb model showing a significant deviation from the trend. A possible explanation is that due to the absence of a neighbor limit (”inf” part of the name) the increased density of larger systems causes the super-linear scaling. The density increases as the system size grows due to the increased ratio between bulk and surface atoms.

One thing to note is that the scaling of these models depends on the hardware used as the bottleneck sets the speed of the entire system (CPU: 5700x3d, GPU: 5070 Ti, RAM: 32 GB DDR4). All but the orb models show that GPU utilization has significant headroom. This means that the system size scaling of the GPU inference time could be worsened when fully utilizing the GPU.

To minimize confounding CPU utilization and RAM usage were sampled based on process id, but unfortunately no such option was available for GPU utilization and VRAM usage. The confounding effects can be mitigated by the KIM EAM baseline which is a classical force field and is expected to utilize only the CPU and RAM. The KIM EAM baseline shows that minimal to no confounding effects are expected for GPU utilization. This does not apply to VRAM, the KIM EAM is measured to use some amount of VRAM which can be attributed to background processes. These background processes can vary in VRAM usage so some error is expected but scaling is unaffected as these background

processes do not interact with the model process(es). In order to prevent VRAM usage of previously benchmarked structures, the PyTorch cache was cleared after each run. This removes unnecessary torch objects on the GPU restoring the starting state as much as possible.

Figure 4.1 and Figure 4.2 evaluate serial computation of systems (using ASE), but as explained in Chapter 3 this is changed by the Torchsim implementation of the code. This means that the large VRAM scaling of the conservative ORBv3 model is very detrimental to its performance when using autobatching of TorchSim.

4.1.2. Force field accuracy

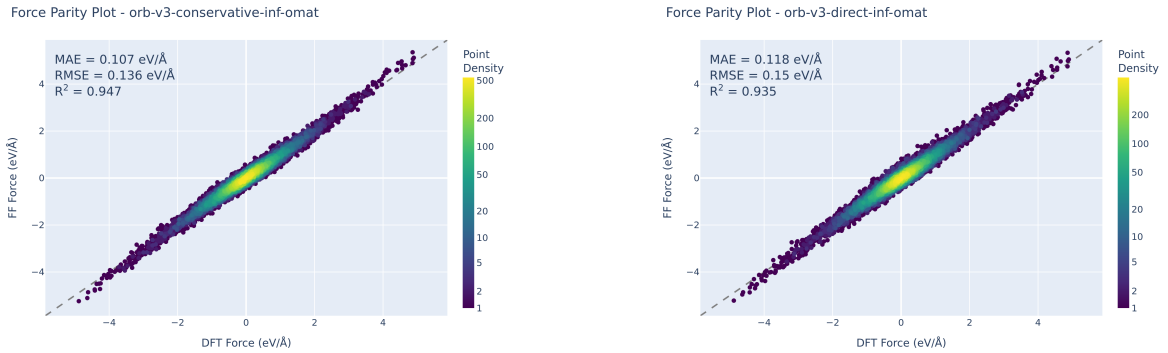


Figure 4.3: Force field accuracy benchmark on the conservative-inf-omat (left) and direct-inf-omat (right) orbv3 model. Ground truth data is the VASP DFT test set created for the fine-tuning of the latest MACE model (v2v1).

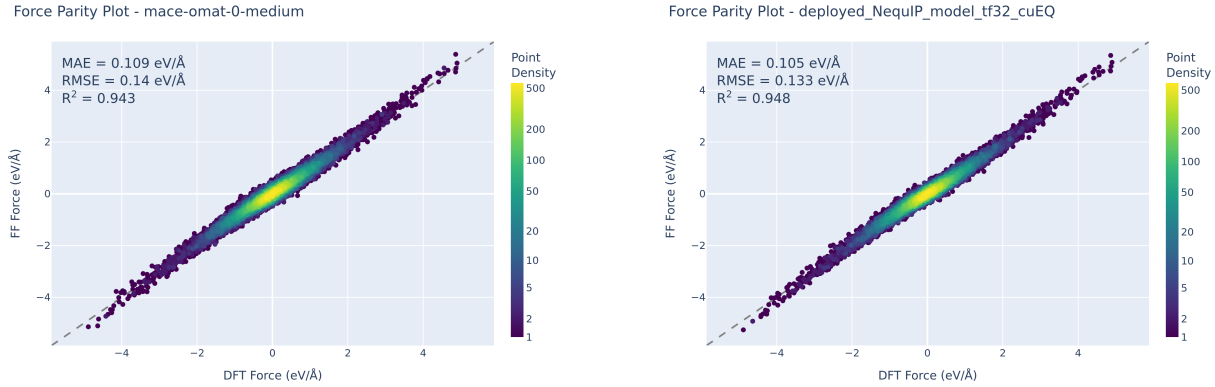


Figure 4.4: Force field accuracy benchmark on the mace-omat-medium foundational MACE model (left) and foundational NequIP model (right). Ground truth data is the VASP DFT test set created for the fine-tuning of the latest MACE model (v2v1).

The final aim of benchmarking these models or model architectures is to make an informed choice based on all important aspects of an MLIP model. These aspects include the discussed computational efficiency but more importantly accuracy. As can be seen in Figure 4.3 and Figure 4.4, All models have similar accuracies so no real advantage is to be gained by using one model architecture over another. As noted in the methods section the CP2K data proved to be inadequate as ground truth so VASP DFT data was used. In these accuracy benchmarks DFT values of VASP were taken as ground truth but because of the PBE level of DFT and convergence setting it is known that this ground truth can have some inaccuracies. These inaccuracies are a confounding factor when assessing the model accuracy. This confounding will only increase as the MLIP models move closer to DFT level accuracy. At a certain point increased accuracy will therefore only be overfitting to a certain DFT level and dataset. This overfitting can lead to nonphysical behaviors and unstable molecular dynamics when predicting out-of-distribution structures.

Based on the benchmarking results, we can remove the Mattersim models from consideration. They both show relatively slow model inference with no significant accuracy benefit and are primarily designed for periodic systems. NequIP shows promise and performs very similarly to MACE only showing somewhat worse VRAM scaling. ORBv3 shows great promise due to its very fast inference speeds while using the direct model. However, the inference speed and resource utilization scaling with system size, are unfavorable compared to MACE. These benchmarks do not give a clear winner. Depending on the system size different models excel at serial inference for MD. As stated before the autobatching feature of Torchsim puts more emphasis on the VRAM usage due to the possibility to run multiple MD sims in parallel. This fact in combination with the access to high VRAM GPUs as provided by DelftBlue [15], such as A100 GPUs (80 GB VRAM), means MACE can reach unparalleled throughput at large and small system sizes. Finally, MACE provides the possibility to do multiheaded replay finetuning which can be a significant help in preventing catastrophic forgetting. When all factors are considered, MACE was chosen as the model architecture for this project.

4.2. Final model

In this section, we will discuss the fine-tuning and performance of the final model. A high-accuracy model is important for the validity of the results generated by the model. The VASP rattle data of gallium nanoparticles with 5 different solutes (Ag, Au, Bi, Pd and Pt) and low-energy gallium clusters of Manna et al. were split into a train, validation and test set [40]. The train set includes the bulk of the data and is divided into 20854 frames with gallium clusters ranging from 3 to 55 atoms, some alloyed with the aforementioned solutes. In order to get a representative validation and test set all elements used in training were represented in each set. The validation and test set contains 371 and 811 frames respectively.

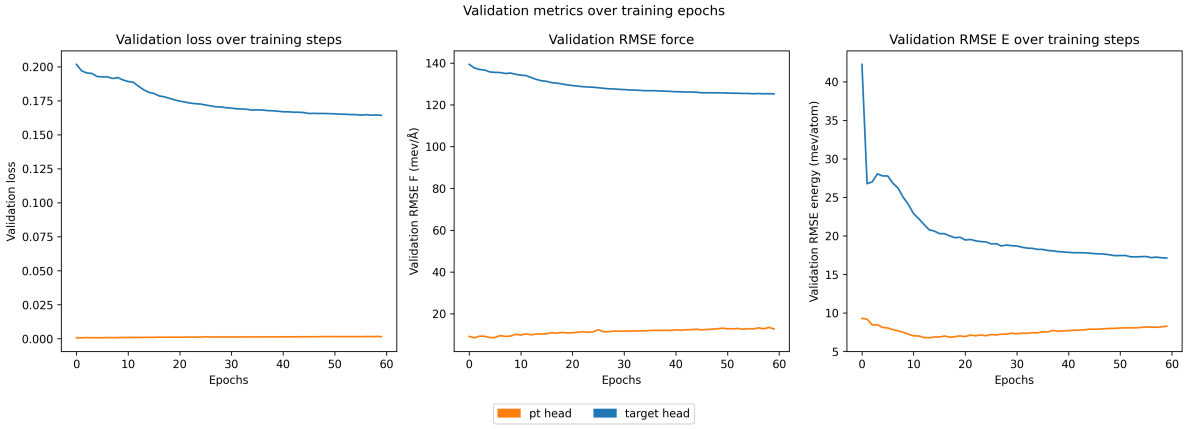


Figure 4.5: Training metrics over epochs for both heads; left is the loss as measured by MACE on the validation dataset, in the middle the force root mean square error in meV per angstrom and the right-most plot shows the root mean square error of the energy in meV.

Figure 4.5 shows that the base foundational model (omat medium) with the new head (based on the computed E0) starts with a relatively good force RMSE of ≈140 meV/Å and energy RMSE of ≈42 meV. The final force RMSE is set at ≈125 meV/Å and energy RMSE of ≈17 meV. The small improvement in RMSE shows that foundational models provide a good baseline but some improvement is still possible. It is probable that the relatively small decrease in RMSE is caused by the low learning rate. Increasing the learning rate could yield better accuracy metrics, but as previously discussed this would possibly come at the cost of generalizability and stability.

When we evaluate the test set we see a similar improvement from a Mean Absolute Error (MEA) of 107 meV/Å (mace-omat-medium) to 90.7 meV/Å (v2v1). Due to the larger size of the test set it is expected that these values better represent the model accuracy.

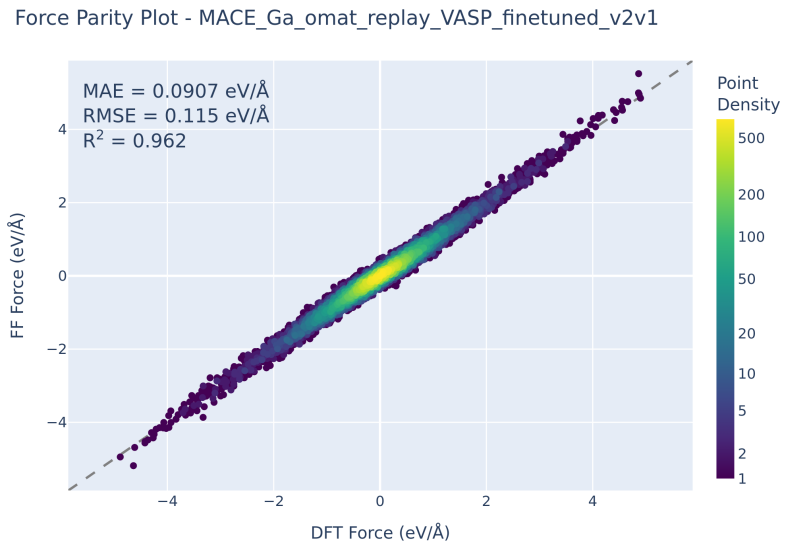


Figure 4.6: Force field benchmark of mace-omat-medium model finetuned using multi-headed replay finetuning and rattle data from VASP DFT calculations of Ga NPs and alloyed with Ag, Au, Bi, Pd and Pt.

4.3. Structural dynamics

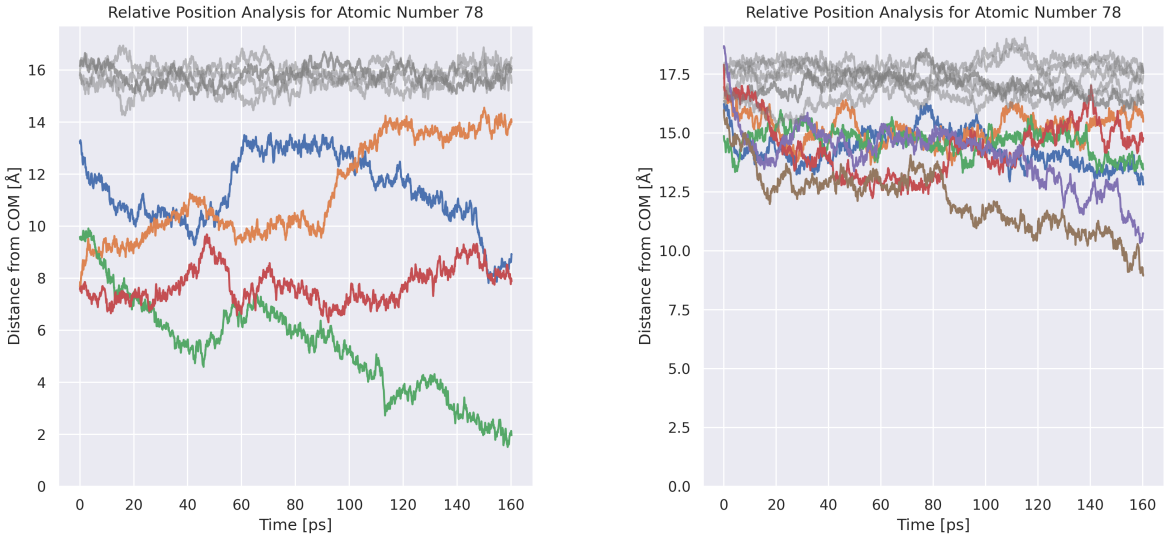
Early theories of the catalytic workings of SCALMS suggested that active metal solutes such as Pt and Pd to diffuse to the surface where they can form an active site [50, 7]. This was later disproven by Lambie and Steenbergen et al. In the following section, similar experiments are performed using different methods as addition to the investigation in the catalytic mechanisms of SCALMS. A nanoparticle set was made with 342, 653, 987, 1247 and 1568 atoms. This set was duplicated 5 times and alloyed with different metal solutes. Pt and Pd are the first solutes. They were chosen for their application in catalysis such as the propane dehydrogenation reaction. The other three solutes are Au, Ag and Bi, these were chosen because Steenbergen et al. investigated a wide array of solutes, including these three. As stated before they show that bismuth preferably moves to the vacuum interface while other solutes prefer to stay inside the nanoparticle or just below its surface [56]. An MD simulation of $\approx$ 160 ps with a timestep of 1 fs was run on all alloyed NP systems at a temperature of 450 Kelvin.

Two important differences exist between the approach of Steenbergen et al. and the one taken in this work. Firstly, the figures produced by Steenbergen et al. are averages of 30 trajectories but due to time and resource constraints only one trajectory was generated for each size solute combination, consequently the following observed behaviors are affected by random noise/thermal fluctuations and have reduced statistical strength. Secondly, Steenbergen et al. use a slab model with periodic boundary conditions in the x and y axes, which significantly differs from the nanoparticle system simulated in this report. For example, this changes the depth trace of Steenbergen et al. into a radial depth trace and can significantly alter the metal solute trace visualization due to the increased available volume with respect to the distance to the COM.

4.3.1. Pt

Figure 4.7 shows two examples of Pt solute trace plots. Two general trends can be seen. The first is the tendency of surface seeded Pt atoms to move to the subsurface where they stay relatively immobilized compared to the bulk seeded Pt atoms.

A possible reason for the energetic favoring of Pt to the subsurface compared to the surface can be found using Figure 4.8. The peak of the Pt-Ga partial PDF is located before and is higher compared to the Ga-Ga partial PDF. This means Pt forms a dense coordination sphere with a smaller interatomic distance between Pt and Ga than Ga and Ga. The combination of the smaller Pt-Ga interatomic distance in the first coordination sphere and the energetic favorability of the Pt-Ga interaction can explain the move from the surface to the subsurface.



(a) 987 atoms

(b) 1247 atoms

Figure 4.7: Metal solute trace plot of a Ga nanoparticle with Pt as metal solute.

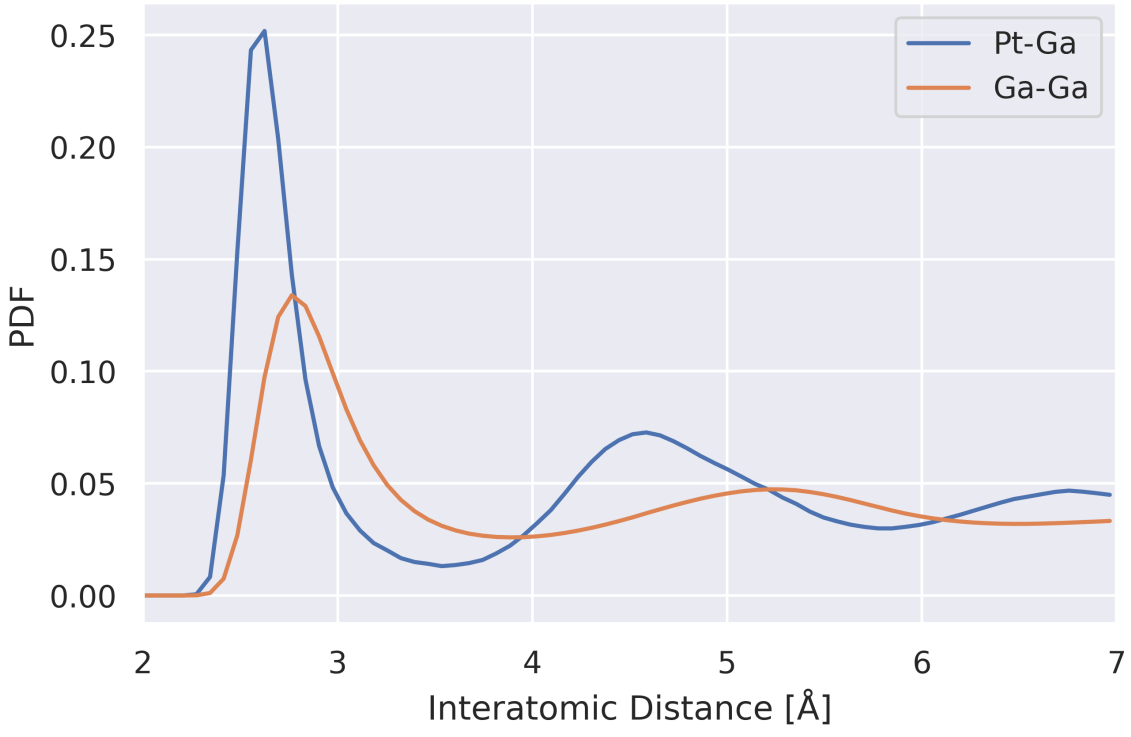


Figure 4.8: PDF generated based on all MD data of all Ga NP sizes alloyed with Pt.

The diffusion constant can be seen to remain the same between Figure 4.7b, which contains primarily surface seeded Pt atoms, and Figure 4.7a which primarily contains bulk seeded Pt atoms (Table 4.1). This is unexpected on longer timescales, but due to the confined nature of a nanoparticle, the maximum lag time was taken to be 1 ps which limits the scope of the diffusion constant analysis. Despite this limitation, the similarity in diffusion constants can be interpreted as a validation of the underlying method to produce this trajectory e.g. the MACE model.

The relative immobilization of the subsurface Pt atoms compared to the bulk Pt atoms at long time scales can again be explained with the PDF plot (Figure 4.8). Due to the surface tension of gallium the density at the surface is expected to be higher and therefore enable the dense coordination sphere.

Despite the general trends aligning between this work and the work of Steenbergen et al. specific equilibrium depths and the equilibrium radial distances do significantly deviate but this is likely a direct consequence of the differences discussed before.

4.3.2. Ag, Au, Bi and Pd

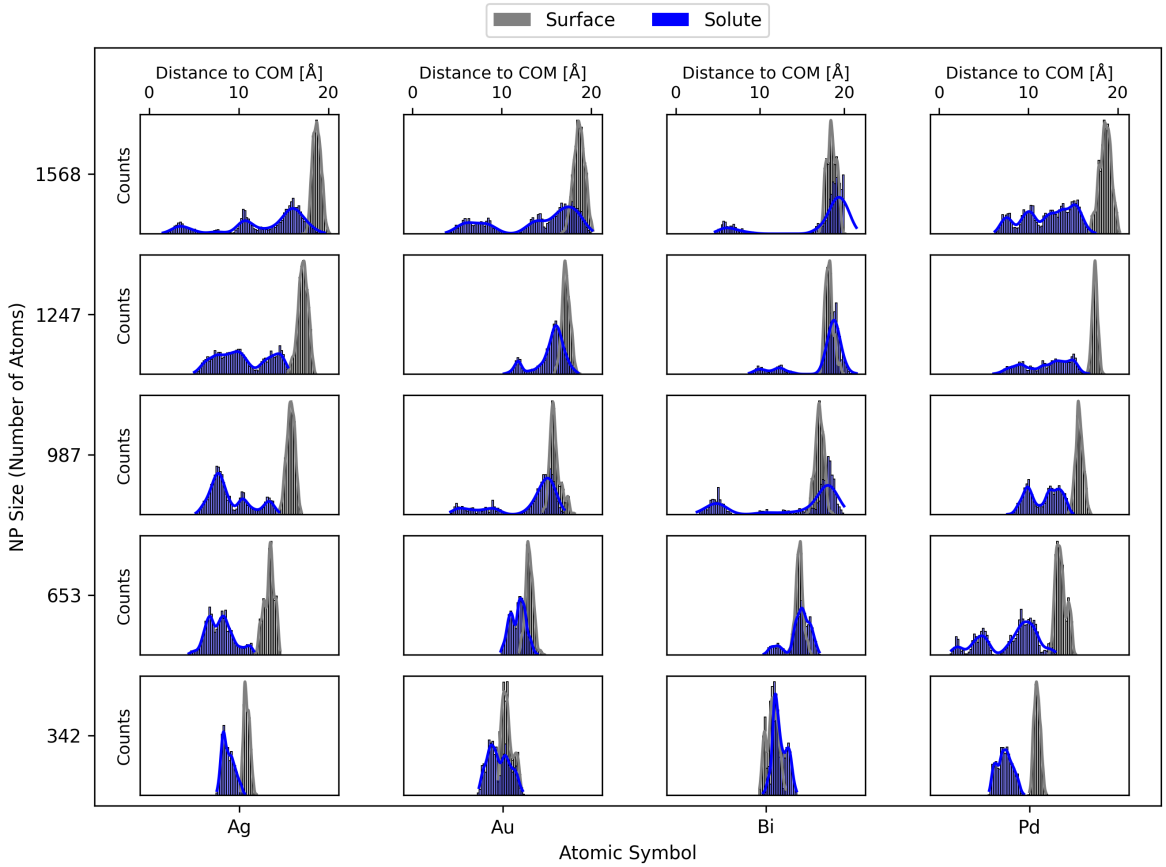


Figure 4.9: Metal solute trace summary plot. The remaining solutes against the nanoparticle size. Each inset plot shows the distribution of the distance from the COM of the nanoparticle. In the inset plots the (second half of the) MD trajectory is depicted in blue and the approximate surface in gray.

Figure 4.9 shows a summary of the trace plots for all other solutes (Figures A.5 to A.7). Similar to Pt, Ag and Pd show the tendency to stay in the subsurface when seeded close to the surface and randomly diffuse when seeded in the bulk. Despite this similarity, the PDF of Ag (Figure A.1) is significantly different to that of Pt. It shows a very similar distribution between Ga-Ga and Ag-Ga with the only two minor differences a small increase in density in the first coordination sphere and a slightly lower distance to the second coordination sphere. The relatively small energetic difference of Ag compared to Pt can also explain the absence of immobilization in the subsurface (Figure A.5).

As previously shown by Steenbergen et al. the bismuth solutes stay at the surface of the nanoparticle

and are attracted by the surface if seeded in the subsurface. They posit that this is because of a large amount of factors such as effective size and interaction with the structured gallium surface. This behavior is noted in this work as well as can be seen in the overlap of the distributions in Figure 4.9. Figure A.3 shows the peak of the first coordination sphere of Bi-Ga to be at a longer interatomic distance and to be less populated due to the reduced height. The increase interatomic distance shows the possible effect of effective size on solute behavior.

It would be expected that gold acts similar to all other solutes and would move to the subsurface due to similar effective size and chemical properties to Pt. Figure A.6 show that although the gold atoms do not move to be on the surface as can be seen with Bi but to be a part of the surface. Figure 4.9 shows the peak of the radial distance distribution to be before the peak of the surface distribution but still significant overlap between the two distributions exists. This is further complicated by the partial PDF of gold with gallium (Figure A.2), which shows a similar but reduced form compared to Pt and Pd. Further investigation is needed to validate this result.

4.3.3. Diffusion constant

Table 4.1: All measured diffusion constants of metal solutes in Ga NP in 10^{-6} cm^2/s.

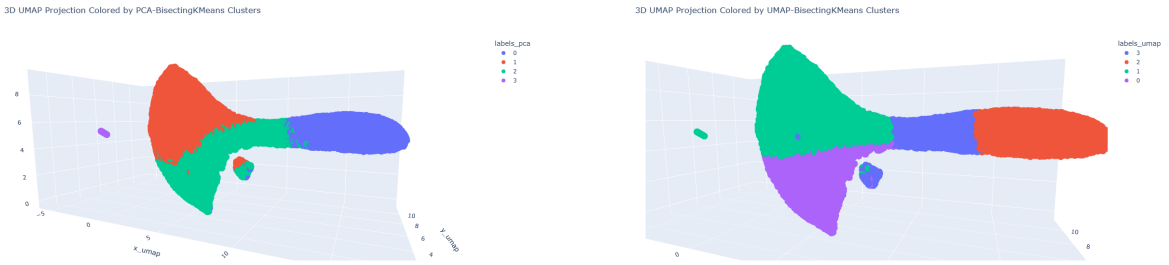
Atoms / Solutes	Ag	Au	Bi	Pd	Pt
342	10.5	8.75	8.45	7.10	5.39
653	10.2	8.56	7.57	6.99	5.51
987	10.9	8.05	8.27	7.05	5.20
1247	10.5	8.45	8.12	7.05	5.45
1568	10.9	8.04	8.14	7.38	5.42

The diffusion constant could play a significant role in the catalytic workings of the SCALM systems due to the dynamic nature of the system and the catalytic site. In this section, we show all the measured diffusion constants.

All metal solutes are confined to the gallium nanoparticle, which results in the saturation of the SD at long lag times so the maximum lag time was set to one picosecond. After one picosecond R^2 values significantly reduce as the linear relation vanishes. Diffusion constants found in Table 4.1 (Figures A.9 to A.13) align with the diffusion constants found by Steenbergen et al. [56] which further validates the ability of the model to accurately simulate dynamic environments.

4.4. Structural analysis

Insight in the structure and different environments in a system can be gained by exploring the high dimensional representation of MLIPs in lower dimensions. Care must be taken when interpreting these results because of the lack of inherent meaning in the dimensions generated by UMAP.

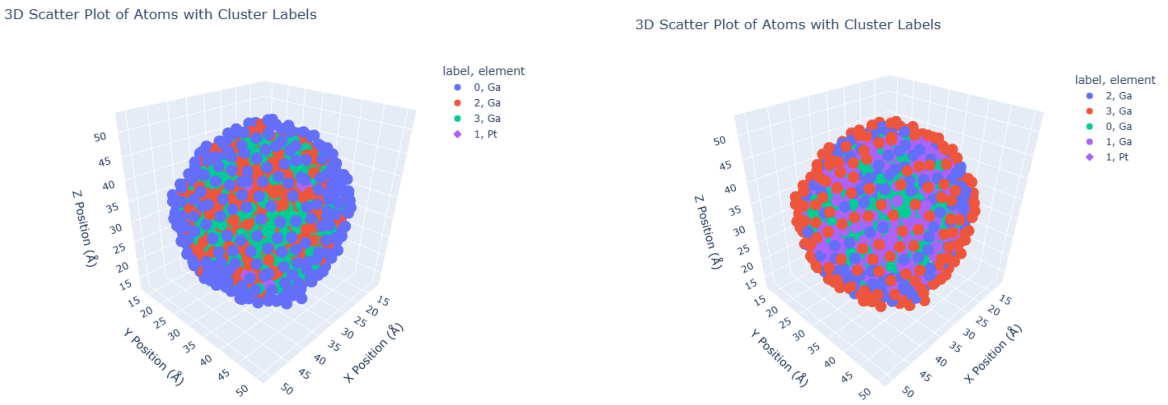


(a) PCA **(b)** UMAP

Figure 4.10: 3D representation of 256-dimensional MACE descriptors. These descriptors were extracted from the Ga Pt MD sim as shown before in the Pt solute trace. They are labeled using hierarchical KMeans clustering on UMAP/PCA reduced descriptors (10 dimensions).

In Figure 4.10 we see that UMAP maps the 256-dimensional space into 3 distinct areas. One big area shaped somewhat similar to a hammer and for the rest of the report named as such, one small area

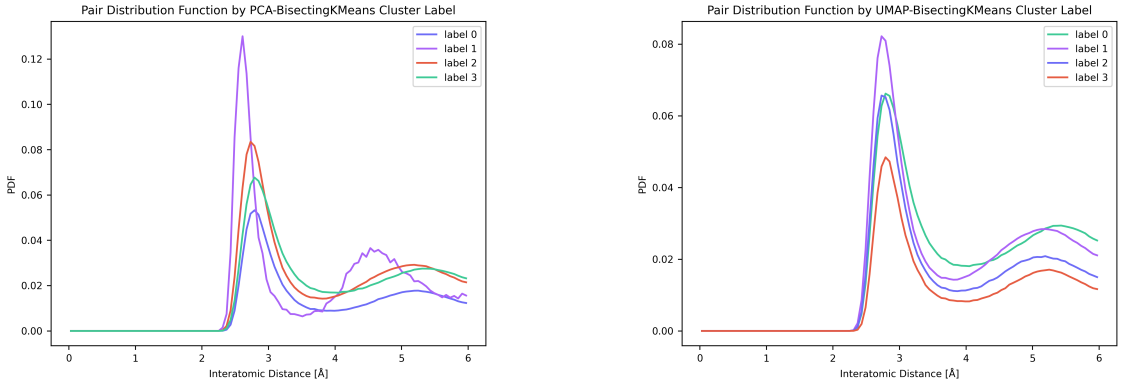
far left of the hammer and one blob below the hammer. One main difference between the PCA and UMAP is the labeling of the small island on the left of the hammer. On investigation of Figure 4.11a and the data we can see that this small island contains all Pt atoms. The PCA separates the Pt from the gallium while UMAP adds it to the top lobe of the hammer.



(a) PCA **(b)** UMAP

Figure 4.11: 3D scatterplot of atomic locations labeled by hierarchical k-means clustering on PCA and UMAP reduced descriptors (10 dimensions).

The horizontal axis of the hammer can be interpreted as a density or coordination of gallium atoms. This can be seen in Figure 4.11b where the gallium atoms labeled orange are at the outermost layer of the nanoparticle and therefore the least coordinated. On the left of the orange cluster in Figure 4.10b is the blue cluster; this cluster contains atoms that are part of the surface but have an increased coordination number compared to the orange label.



(a) PCA **(b)** UMAP

Figure 4.12: Pair density function of labels found by hierarchical k-means clustering on UMAP and PCA reduced descriptors (10 dimensions).

Figure 4.11 shows the local structure of each of the labels for both PCA and UMAP. To start with PCA, the largest peak is the cluster labeled 1, which corresponds to the cluster that only contains Pt atoms and therefore also shows a very similar form as seen before in the partial Pt-Ga PDF (Figure 4.8). The lowest peak corresponds as we stated before to the surface atoms due to the low coordination. Zeni et al. investigated the melting of gold nanoparticles they showed multiple different liquid environments [70]. Two of which are high and low coordination liquid environments. This can again be see in the data of this work and is visually represented as the two ends of the head of the hammer. In both

UMAP and PCA, the two liquid environments are separated and both show the high coordination liquid environment having a larger first coordination peak and an earlier second coordination peak.

4.5. Towards Catalytic dynamics

The paper by Lambie et al. describes the formation of a persistent geometric feature created by Pt which could be an active site of this SCALM system. To capture the possible formation of an active site and dehydrogenation reaction of propane, five 1568-atom nanoparticle GaPt SCALM systems with 200 propane molecules were simulated for 50 ps (dt = 1 fs) at 823 Kelvin which corresponds with the temperature used by Raman et al. [49]. With the five 1568 nanoparticles all having different starting positions and orientations regarding the reactants. Unfortunately, no hydrogen formation or geometric feature as shown by Lambie et al. was seen in the simulations. With regard to the absence of the propane dehydrogenation reaction, many possible reasons exist. One of the more likely reasons is that the PES of the reaction is not sufficiently explored by the finetuned foundational OMAT medium MACE model. Datasets like OMAT do contain non-equilibrium structures and data from converging rattled systems, but due to the general PES softening described by Deng et al. and the combinatorial explosion of possible PES environments, the propane dehydrogenation reaction is likely too far out-of-distribution [16].

The range of diffusion constants found for these MD simulations was between 1.25E-05 cm^2/s and 1.34E-05 cm^2/s, which is significantly higher than the values noted in Table 4.1. This is probably due to the increase in temperature, but another factor could be influencing it. If a reactant such as propane successfully binds to the surface it would reduce the surface tension and therefore change the surface structure. Using the assumption that Pt stays in the subsurface because of its relatively high density of gallium, it could allow Pt to move more freely as seen in the bulk.

5

Conclusion

Benchmarking orbv3 conservative/direct, MACE omat/mpa/MATPES, NequIP and Mattersim 1M/5M MLIP models showed their computational efficiency, accuracy and finally practical differences. MACE proved to be the best architecture for the large-scale MD on SCALMS using HPC hardware. In agreement with literature, it was observed that CP2K has some net force issues which made it hard to use for MLIP benchmarking and finetuning (Appendix B). Finetuning a mace-omat-medium model gave a final model with a test set mean absolute error of 0.0907 ev/Å. Furthermore, tools and methods are explored for the entire computational workflow. In particular, the new framework torch-sim was instrumental in efficiently running multiple large MD simulations in parallel on HPC hardware.

The structural dynamics of a liquid gallium nanoparticle alloyed with Ag, Au, Bi, Pd and Pt were investigated by tracing the solutes over time. When solutes were seeded on the surface, they were seen to generally avoid the surface and find an equilibrium in the subsurface. Bismuth was observed to actively move towards the surface. Unseen before in literature, gold was observed to find an equilibrium position close to or as a part of the surface. Diffusion constants were consistent with literature data and showed no statistically significant relation with system size. The investigation of the descriptors generated by MACE showed significant structural differences between different parts of the nanoparticle. If we look at both PCA and UMAP, they identified the following groups: solute atoms, surface atoms, high coordinated bulk atoms and low coordinated bulk atoms. Unfortunately the propane dehydrogenation reaction could not be captured and therefore the formation of the catalytic site was not observed as well.

5.1. Outlook

A lot of possible research avenues still exist. The following list summarizes possible paths for further research into SCALM dynamics in order of decreasing priority:

- Capturing catalytic site formation
- Reactant/support influence study on SCALM dynamics
- Microkinetic model of propane dehydrogenation
- Quantifying solute difference in descriptor space
- Active learning methods
- Committee of models

Capturing the catalytic site formation using MD would shed light on the workings of SCALMs. Due to the fluxionatlity of the gallium nanoparticle it is expected that a whole ensemble of these catalytic sites exits, definition of this structure could be given by investigating the related MACE descriptors. Research into the influence of reactants and supports would give insights into the experimentally noted differences which could be further exploited for improvement of SCALM systems. Furthermore when the catalytic site formation is captured and the reaction is observed microkinetic models could directly explain the

experimentally observed activity of SCALM systems. In general all the discussed research avenues will increase the understanding of SCALM systems and allow for rational design and improvement of current SCALM systems [46].

Steenbergen et al. noted that the different solute dynamics recorded could not be explained by any one or 2 parameter combination, because all apparent correlations had a counterexample. Quantifying solute differences in descriptor space could be an efficient method to explain the different behaviors noted in this report and could provide insight into the fundamental reasons for these differences.

To support these scientific efforts, improvements to the methods can be made. To more efficiently gather DFT data for model training, active learning methods can be explored. When large DFT datasets are used the possibility of improving robustness and definition of uncertainty would make a committee of models the preferred way to use MLIP models. Unfortunately this would currently significantly increase computational cost due to the repeated inference of the same system. This issue has been recently solved by Beck et al. by only repeating the inference of the readout functions and sharing the MPNN part of the model as is done in multiheaded models, but no implementation of this exists in MACE to this day [8].

References

[1] G.J.K Acres et al. “The Use of Supported Solutions of Rhodium Trichloride for Homogeneous Catalysis”. In: *Journal of Catalysis* (Aug. 1966).

[2] Mariam Ameen et al. *Liquid Metal Alloy Catalysis – Challenges and Prospects*. Nov. 2023. DOI: 10.1002/cctc.202300814.

[3] Albert P. Bartók et al. “Gaussian approximation potentials: The accuracy of quantum mechanics, without the electrons”. In: *Physical Review Letters* 104.13 (Apr. 2010). ISSN: 00319007. DOI: 10.1103/PhysRevLett.104.136403.

[4] Ilyes Batatia et al. “Cross Learning between Electronic Structure Theories for Unifying Molecular, Surface, and Inorganic Crystal Foundation Force Fields”. In: (Oct. 2025). URL: http://arxiv.org/abs/2510.25380.

[5] Ilyes Batatia et al. “MACE: Higher Order Equivariant Message Passing Neural Networks for Fast and Accurate Force Fields”. In: (Jan. 2023). URL: http://arxiv.org/abs/2206.07697.

[6] Peter W. Battaglia et al. “Relational inductive biases, deep learning, and graph networks”. In: (Oct. 2018). URL: http://arxiv.org/abs/1806.01261.

[7] Tanja Bauer et al. “Operando DRIFTS and DFT Study of Propane Dehydrogenation over Solid- and Liquid-Supported GaxPty Catalysts”. In: *ACS Catalysis* 9.4 (Apr. 2019), pp. 2842–2853. ISSN: 21555435. DOI: 10.1021/acscatal.8b04578.

[8] Hubert Beck et al. “Multi-head committees enable direct uncertainty prediction for atomistic foundation models”. In: *The Journal of Chemical Physics* 163.23 (Dec. 2025). ISSN: 0021-9606. DOI: 10.1063/5.0302097. URL: https://pubs.aip.org/jcp/article/163/23/234103/3374754/Multi-head-committees-enable-direct-uncertainty.

[9] Simon J.L. Billinge. *The rise of the X-ray atomic pair distribution function method: A series of fortunate events*. June 2019. DOI: 10.1098/rsta.2018.0413.

[10] Mengyang Cao et al. “Liquid metal based flowable regenerative catalyst for electrochemical nitrate reduction”. In: *Chemical Engineering Journal* 499 (Nov. 2024). ISSN: 13858947. DOI: 10.1016/j.cej.2024.156005.

[11] I. Chorkendorff and J.W. Niemantsverdriet. *Concepts of Modern Catalysis and Kinetics*. Wiley, 2003, pp. 0–10. ISBN: 9783527602650. DOI: DOI:10.1002/3527602658.

[12] Orion Cohen et al. “TorchSim: an efficient atomistic simulation engine in PyTorch”. In: *AI for Science* 1.2 (Dec. 2025), p. 025003. DOI: 10.1088/3050-287x/ae1799.

[13] Orion Cohen[1] et al. *TorchSim: An efficient atomistic simulation engine in PyTorch*. Tech. rep.

[14] Christopher J Cramer. *Essentials of Computational Chemistry Theories and Models Second Edition*. ISBN: 0-470-09182-7.

[15] Delft High Performance Computing Centre (DHPC). *Delft Blue Supercomputer (Phase 2)*. 2024. URL: https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.

[16] Bowen Deng et al. “Systematic softening in universal machine learning interatomic potentials”. In: *npj Computational Materials* 11.1 (Dec. 2025). ISSN: 20573960. DOI: 10.1038/s41524-024-01500-6.

[17] Ralf Drautz. “Atomic cluster expansion for accurate and transferable interatomic potentials”. In: *Physical Review B* 99.1 (Jan. 2019). ISSN: 24699969. DOI: 10.1103/PhysRevB.99.014104.

[18] James A. Dumesic, George W. Huber, and Michel Boudart. “Principles of Heterogeneous Catalysis”. In: *Handbook of Heterogeneous Catalysis*. Wiley, Jan. 1996. DOI: 10.1002/9783527610044.hetcat0001.

[19] Genevieve Dusson et al. “Atomic Cluster Expansion: Completeness, Efficiency and Stability”. In: (May 2021). URL: http://arxiv.org/abs/1911.03550.

[20] Sathya Edamadaka et al. “Universally Converging Representations of Matter Across Scientific Foundation Models”. In: (Dec. 2025). URL: http://arxiv.org/abs/2512.03750.

[21] Felix Egger et al. “Supported Catalytically Active Liquid Metal Solutions (SCALMS) for Propane Dehydrogenation–Intermetallic Phases and Liquid Alloys Studied by Pair Distribution Function Analysis and Density Functional Theory”. In: *Advanced Science* 12.44 (Nov. 2025). ISSN: 21983844. DOI: 10.1002/advs.202511498.

[22] Ryan S Elliott and John Lennard-Jones. *Efficient multi-species Lennard-Jones model with truncated or shifted cutoff v003*. OpenKIM, https://doi.org/10.25950/ac258694. 2018. DOI: 10.25950/ac258694.

[23] Ryan S Elliott and Ellad B Tadmor. *Knowledgebase of Interatomic Models (KIM) Application Programming Interface (API)*. https://openkim.org/kim-api. 2011. DOI: 10.25950/ff8f563a.

[24] Ali Z. Fadhel et al. *Combining the benefits of homogeneous and heterogeneous catalysis with tunable solvents and nearcritical water*. 2010. DOI: 10.3390/molecules15118400.

[25] Syeda Saba Fatima et al. “Current state and future prospects of liquid metal catalysis”. In: *Nature Catalysis* 6.12 (Dec. 2023), pp. 1131–1139. ISSN: 25201158. DOI: 10.1038/s41929-023-01083-3.

[26] Justin Gilmer et al. “Neural Message Passing for Quantum Chemistry”. In: (June 2017). URL: http://arxiv.org/abs/1704.01212.

[27] Grant Sanderson and Josh Pullen. *But what is a Neural Network?* Aug. 2025. URL: https://www.3blue1brown.com/lessons/neural-networks#title.

[28] William L Hamilton, Rex Ying, and Jure Leskovec. *Representation Learning on Graphs: Methods and Applications*. Tech. rep. 2017.

[29] E R Hernández. *Molecular Dynamics: from basic techniques to applications (A Molecular Dynamics Primer) 1*. Tech. rep.

[30] Ask Hjorth Larsen et al. *The atomic simulation environment - A Python library for working with atoms*. June 2017. DOI: 10.1088/1361-648X/aa680e.

[31] Kourosh Kalantar-Zadeh, Torben Daeneke, and Junma Tang. “The atomic intelligence of liquid metals”. In: *Science* 385.6707 (July 2024), pp. 372–373. ISSN: 10959203. DOI: 10.1126/science.adn5871.

[32] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: (Jan. 2017). URL: http://arxiv.org/abs/1412.6980.

[33] W Kohn and L J Sham. “PHYSICAL REVIEW Self-Consistent Equations Including Exchange and Correlation Effects*”. In: ().

[34] Michael Krone et al. “Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories”. In: (2012). DOI: 10.2312/PE/EuroVisShort/EuroVisShort2012/067-071.

[35] Domantas Kuryla et al. “How Accurate Are DFT Forces? Unexpectedly Large Uncertainties in Molecular Datasets”. In: (Oct. 2025). URL: http://arxiv.org/abs/2510.19774.

[36] Stephanie Lambie, Krista G. Steenbergen, and Nicola Gaston. “Dynamic Activation of Ga Sites by Pt Dopant in Low-Temperature Liquid-Metal Catalysts”. In: *Angewandte Chemie - International Edition* 62.19 (May 2023). ISSN: 15213773. DOI: 10.1002/anie.202219009.

[37] Shuting Liang. “Liquid Metal Catalysis”. In: *Handbook of Liquid Metals*. Springer Nature Singapore, 2024, pp. 1–26. DOI: 10.1007/978-981-19-2797-3{_}48-1.

[38] Jingyue Liu. *Catalysis by Supported Single Metal Atoms*. Jan. 2017. DOI: 10.1021/acscatal.6b01534.

[39] Feng Long et al. “Unrevealing liquid metal Ga isolated Pt atom catalysts efficient hydrogenation capability of acid to alkanes”. In: *Applied Catalysis B: Environmental* 365 (May 2025). ISSN: 09263373. DOI: 10.1016/j.apcatb.2024.124915.

[40] Sukriti Manna et al. “A database of low-energy atomically precise nanoclusters”. In: *Scientific Data* 10.1 (Dec. 2023). ISSN: 20524463. DOI: 10.1038/s41597-023-02200-4.

[41] Diego Marcos et al. *Rotation equivariant vector field networks*. Tech. rep. URL: http://github.com/di-marcos/RotEqNet.

[42] Cedric Maton, Nils De Vos, and Christian V. Stevens. “Ionic liquid thermal stabilities: Decomposition mechanisms and analysis tools”. In: *Chemical Society Reviews* 42.13 (June 2013), pp. 5963–5977. ISSN: 14604744. DOI: 10.1039/c3cs60071h.

[43] Christian P. Mehnert et al. “Supported ionic liquid catalysis - A new concept for homogeneous hydroformylation catalysis”. In: *Journal of the American Chemical Society* 124.44 (Nov. 2002), pp. 12932–12933. ISSN: 00027863. DOI: 10.1021/ja0279242.

[44] Rocío Mercado et al. *Graph Networks for Molecular Design*. Aug. 2020. DOI: 10.26434/chemrxiv.12843137. URL: https://chemrxiv.org/articles/preprint/Graph_Networks_for_Molecular_Design/12843137.

[45] Michael Moritz et al. “Supported Catalytically Active Liquid Metal Solutions: Liquid Metal Catalysis with Ternary Alloys, Enhancing Activity in Propane Dehydrogenation”. In: *ACS Catalysis* 14.9 (May 2024), pp. 6440–6450. ISSN: 21555435. DOI: 10.1021/acscatal.4c01282.

[46] Ali Hussain Motagamwala and James A. Dumesic. *Microkinetic Modeling: A Tool for Rational Catalyst Design*. Jan. 2021. DOI: 10.1021/acs.chemrev.0c00394.

[47] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: (Dec. 2019). URL: http://arxiv.org/abs/1912.01703.

[48] Md Arifur Rahim et al. “Low-temperature liquid platinum catalyst”. In: *Nature Chemistry* 14.8 (Aug. 2022), pp. 935–941. ISSN: 17554349. DOI: 10.1038/s41557-022-00965-6.

[49] Narayanan Raman et al. “Gapt supported catalytically active liquid metal solution catalysis for propane dehydrogenation-support influence and coking studies”. In: *ACS Catalysis* 11.21 (Nov. 2021), pp. 13423–13433. ISSN: 21555435. DOI: 10.1021/acscatal.1c01924.

[50] Narayanan Raman et al. “Highly Effective Propane Dehydrogenation Using Ga-Rh Supported Catalytically Active Liquid Metal Solutions”. In: *ACS Catalysis* 9.10 (Oct. 2019), pp. 9499–9507. ISSN: 21555435. DOI: 10.1021/acscatal.9b02459.

[51] Benjamin Rhodes et al. “Orb-v3: atomistic simulation at scale”. In: (Apr. 2025). URL: http://arxiv.org/abs/2504.06231.

[52] Kevin Rossi. *Multiscale modelling of metallic nanoparticles structural and catalytic properties*. Tech. rep. URL: https://kclpure.kcl.ac.uk/portal/.

[53] Lorenzo Russotto et al. *Size-dependent alloy formation of nanoscale liquid metal solutions using Bayesian active learning and equivariant force fields 2*. Tech. rep.

[54] Benjamin Sanchez-Lengeling et al. “A Gentle Introduction to Graph Neural Networks”. In: *Distill* 6.8 (Aug. 2021). ISSN: 2476-0757. DOI: 10.23915/distill.00033.

[55] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. “E(n) Equivariant Graph Neural Networks”. In: (Feb. 2022). URL: http://arxiv.org/abs/2102.09844.

[56] Krista G. Steenbergen et al. “Atomic-Scale Dynamics at the Interface of Doped Liquid Gallium: Contrasting Effects of Gallium Oxide and Vacuum”. In: *Small Science* 5.6 (June 2025). ISSN: 26884046. DOI: 10.1002/smsc.202500153.

[57] Alexander Stukowski. “Computational analysis methods in atomistic modeling of crystals”. In: *JOM* 66.3 (Mar. 2014), pp. 399–407. ISSN: 10474838. DOI: 10.1007/s11837-013-0827-5.

[58] Alexander Stukowski. “Visualization and analysis of atomistic simulation data with OVITO-the Open Visualization Tool”. In: *Modelling and Simulation in Materials Science and Engineering* 18.1 (2010). ISSN: 09650393. DOI: 10.1088/0965-0393/18/1/015012.

[59] Xi Sun and Hui Li. *Recent progress of Ga-based liquid metals in catalysis*. Sept. 2022. DOI: 10.1039/d2ra04795k.

[60] N. Taccardi et al. “Gallium-rich Pd-Ga phases as supported liquid metal catalysts”. In: *Nature Chemistry* 9.9 (Sept. 2017), pp. 862–867. ISSN: 17554349. DOI: 10.1038/NCHEM.2822.

[61] Philipp Thölke and Gianni De Fabritiis. *TORCHMD-NET: EQUIVARIANT TRANSFORMERS FOR NEURAL NETWORK BASED MOLECULAR POTENTIALS*. Tech. rep.

[62] Richard Tran et al. “Anisotropic work function of elemental crystals”. In: *Surface Science* 687 (Sept. 2019), pp. 48–55. ISSN: 00396028. DOI: 10.1016/j.susc.2019.05.002.

[63] Richard Tran et al. “Surface energies of elemental crystals”. In: *Scientific Data* 3.1 (Sept. 2016), p. 160080. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.80.

[64] Umah Nnamdi Jeremiah et al. “Ga-Pt bimetallic catalysts for propane dehydrogenation: Synergy, stability and strategies for industrial advancement: A literature review”. In: *World Journal of Advanced Research and Reviews* 27.1 (July 2025), pp. 2728–2744. DOI: 10.30574/wjarr.2025.27.1.2819.

[65] Gido M. van de Ven, Nicholas Soures, and Dhireesha Kudithipudi. “Continual Learning and Catastrophic Forgetting”. In: (Mar. 2024). DOI: 10.1016/B978-0-443-15754-7.00073-0. URL: http://arxiv.org/abs/2403.05175%20http://dx.doi.org/10.1016/B978-0-443-15754-7.00073-0.

[66] Moritz Wolf et al. “Capturing spatially resolved kinetic data and coking of Ga-Pt supported catalytically active liquid metal solutions during propane dehydrogenation: In situ”. In: *Faraday Discussions* 229 (Feb. 2021), pp. 359–377. ISSN: 13645498. DOI: 10.1039/d0fd00010h.

[67] Moritz Wolf et al. “Dry reforming of methane over gallium-based supported catalytically active liquid metal solutions”. In: *Communications Chemistry* 6.1 (Dec. 2023). ISSN: 23993669. DOI: 10.1038/s42004-023-01018-w.

[68] Zonghan Wu et al. “A Comprehensive Survey on Graph Neural Networks”. In: (Dec. 2019). DOI: 10.1109/TNNLS.2020.2978386. URL: http://arxiv.org/abs/1901.00596%20http://dx.doi.org/10.1109/TNNLS.2020.2978386.

[69] Mengjia Xu. “Understanding graph embedding methods and their applications”. In: (Dec. 2020). URL: http://arxiv.org/abs/2012.08019.

[70] Claudio Zeni et al. “Data-driven simulation and characterisation of gold nanoparticle melting”. In: *Nature Communications* 12.1 (Dec. 2021). ISSN: 20411723. DOI: 10.1038/s41467-021-26199-7.

[71] Claudio Zeni et al. “Exploring the robust extrapolation of high-dimensional machine learning potentials”. In: *Physical Review B* 105.16 (Apr. 2022). ISSN: 24699969. DOI: 10.1103/PhysRevB.105.165141.

[72] Hui Zheng et al. “Grain Boundary Properties of Elemental Metals”. In: (July 2019).

[73] Xing Zhi et al. *Fundamentals, Applications, and Perspectives of Liquid Metals in Catalysis: An Overview of Molecular Simulations*. Dec. 2023. DOI: 10.1021/acs.energyfuels.3c02754.

[74] Jie Zhou et al. *Graph neural networks: A review of methods and applications*. Jan. 2020. DOI: 10.1016/j.aiopen.2021.01.001.

[75] Karma Zuraiqi et al. “Unveiling metal mobility in a liquid Cu–Ga catalyst for ammonia synthesis”. In: *Nature Catalysis* 7.9 (Sept. 2024), pp. 1044–1052. ISSN: 25201158. DOI: 10.1038/s41929-024-01219-z.

A

Additional figures

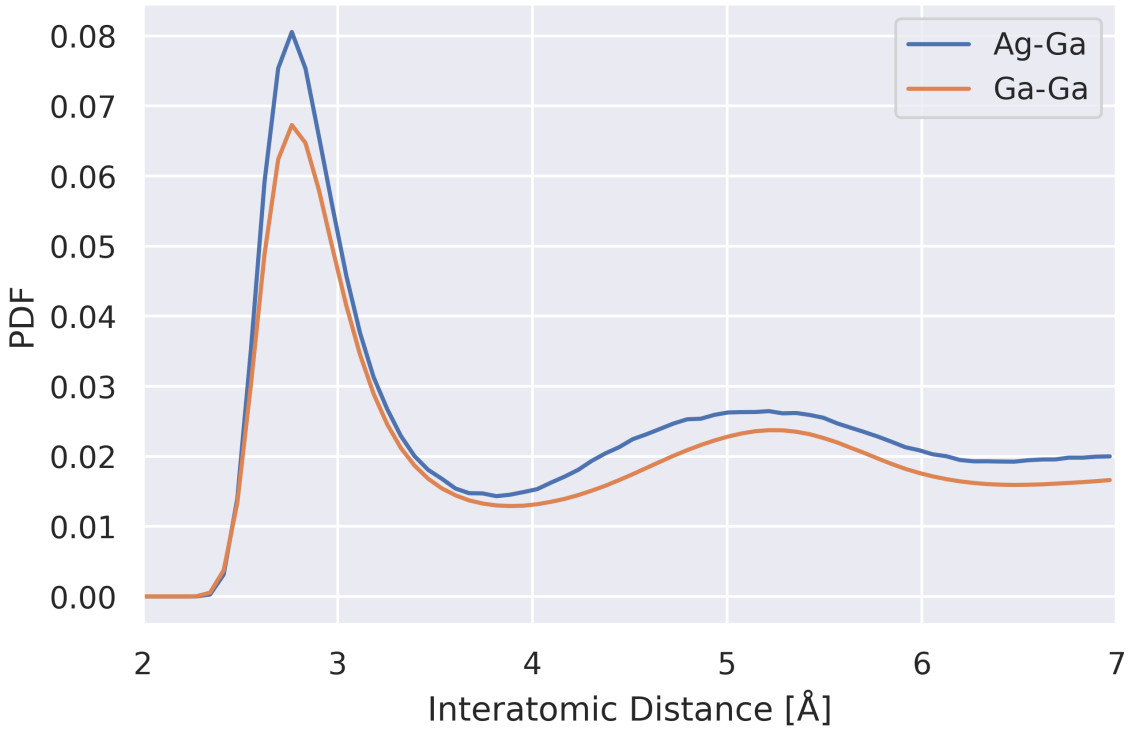


Figure A.1: PDF plot of Ga NP alloyed with Ag averaged over all NP sizes (342,653,987,1247,1568 atoms)

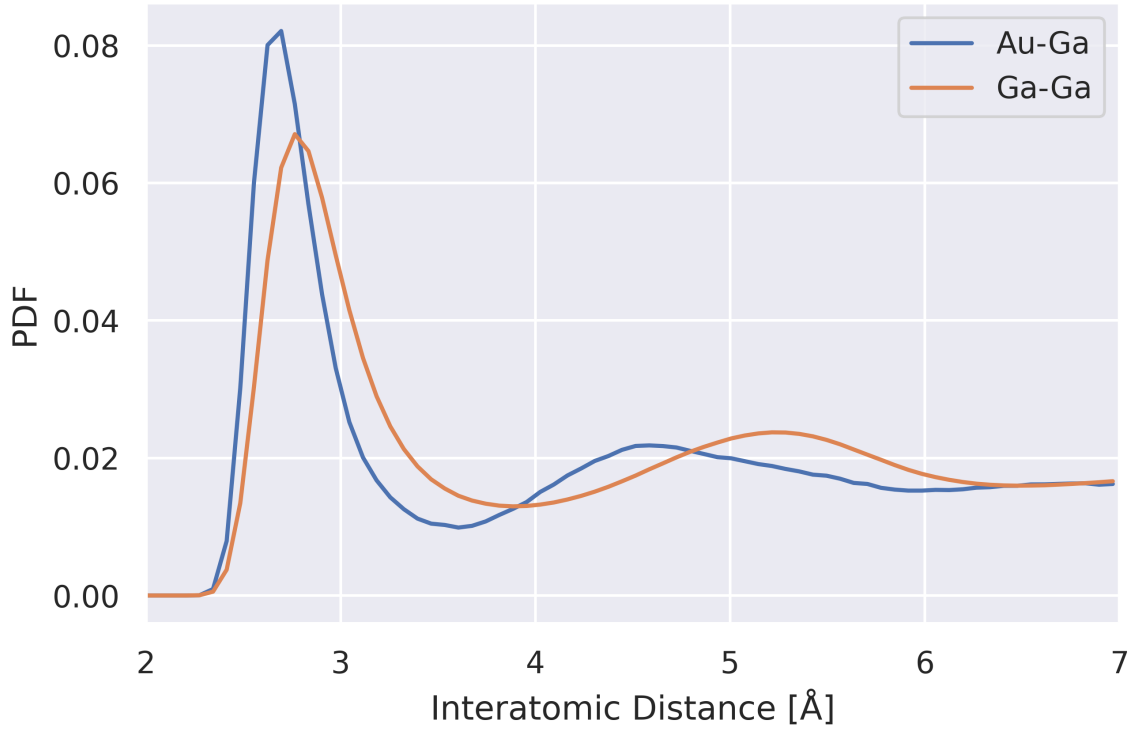


Figure A.2: PDF plot of Ga NP alloyed with Au averaged over all NP sizes (342,653,987,1247,1568 atoms)

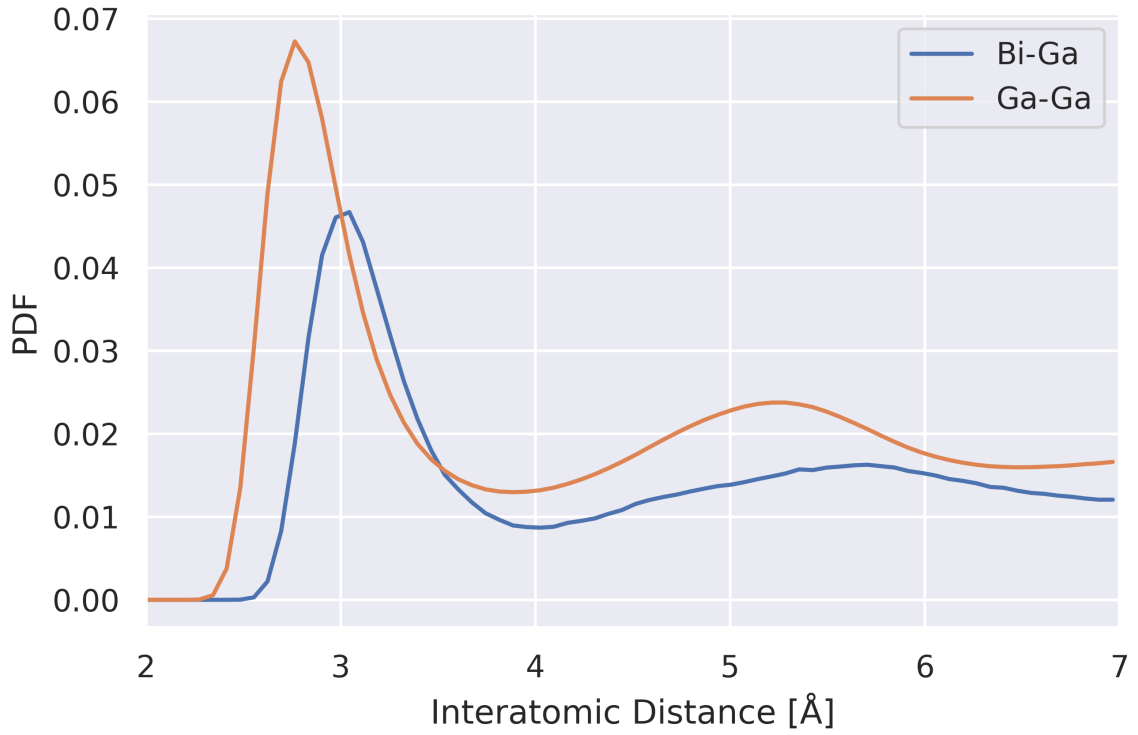


Figure A.3: PDF plot of Ga NP alloyed with Bi averaged over all NP sizes (342,653,987,1247,1568 atoms)

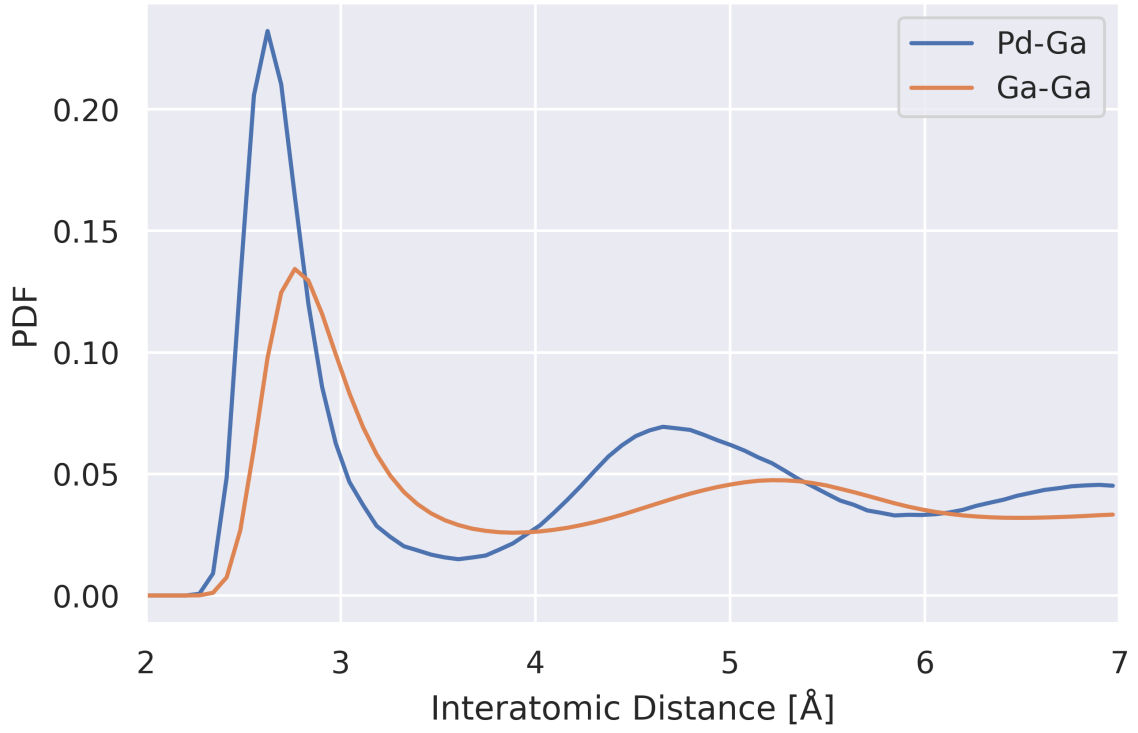
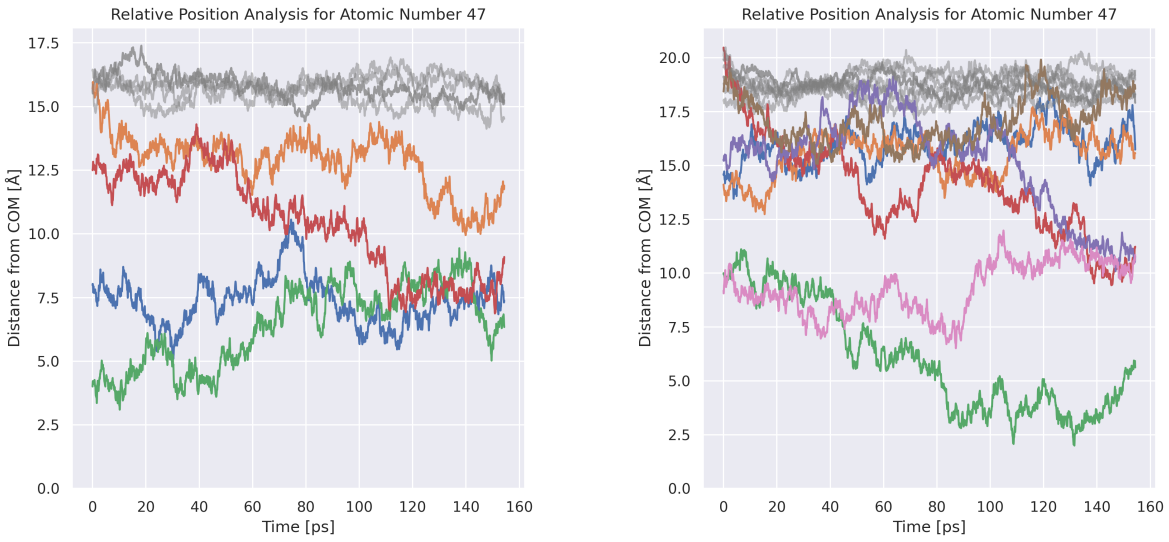


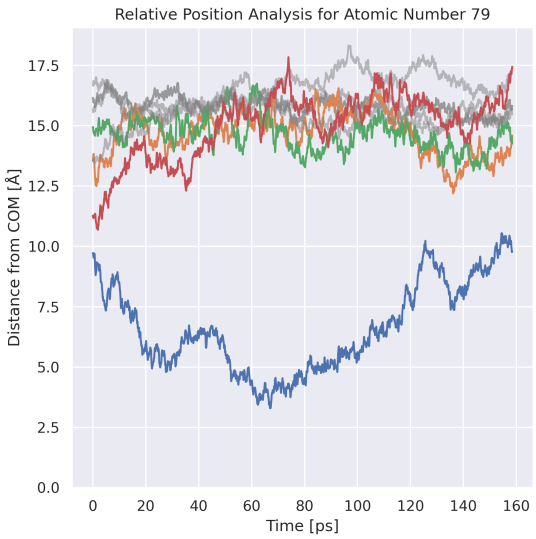
Figure A.4: PDF plot of Ga NP alloyed with Pd averaged over all NP sizes (342,653,987,1247,1568 atoms)



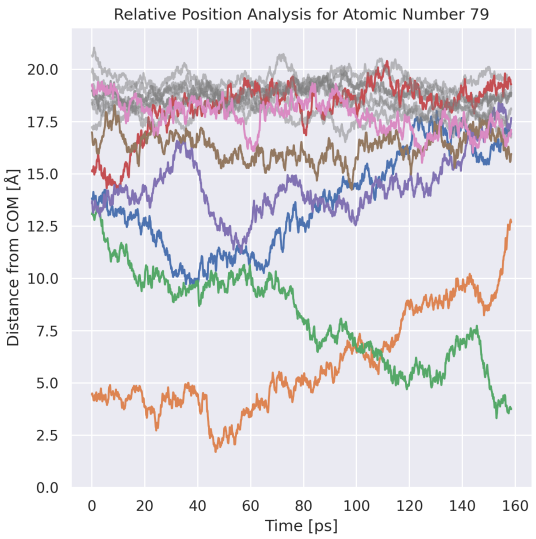
(a) 987 atoms

(b) 1568 atoms

Figure A.5: Metal solute trace plot of a Ga nanoparticle with Ag as metal solute.

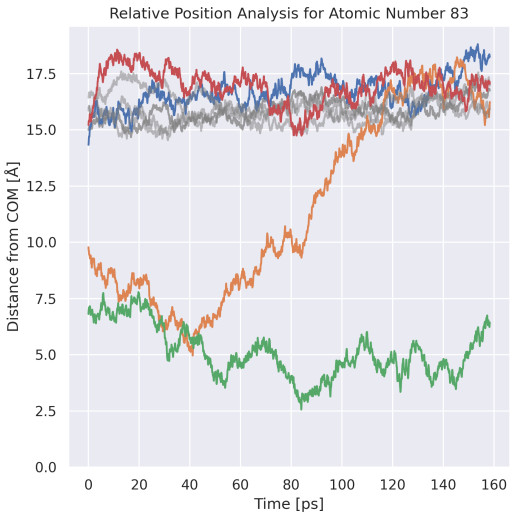


(a) 987 atoms

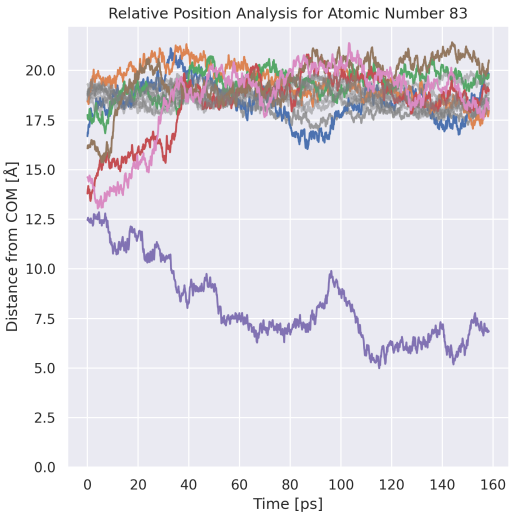


(b) 1568 atoms

Figure A.6: Metal solute trace plot of a Ga nanoparticle with Au as metal solute.

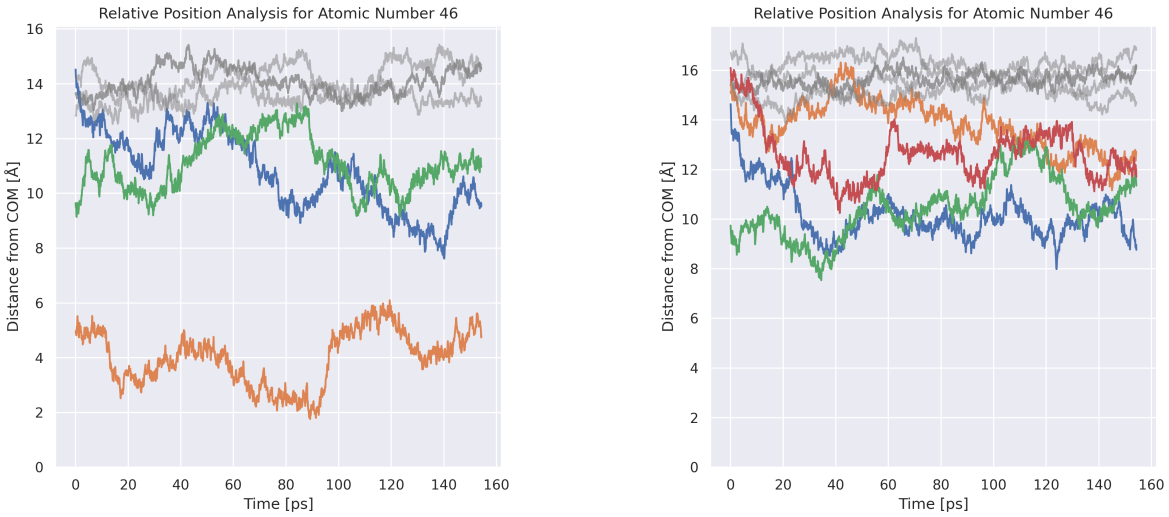


(a) 987 atoms



(b) 1568 atoms

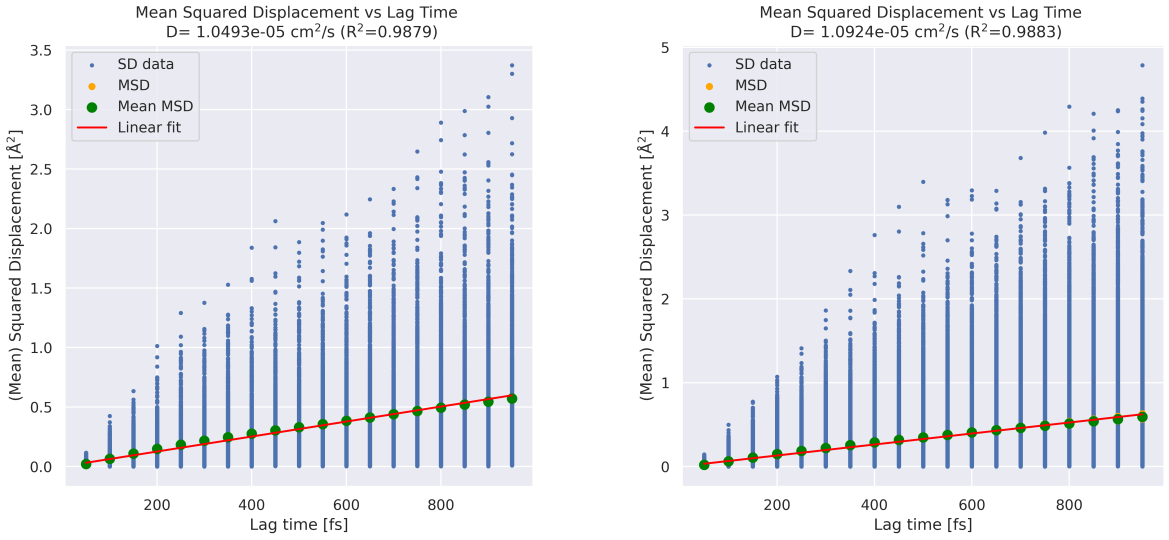
Figure A.7: Metal solute trace plot of a Ga nanoparticle with Bi as metal solute.



(a) 653 atoms

(b) 987 atoms

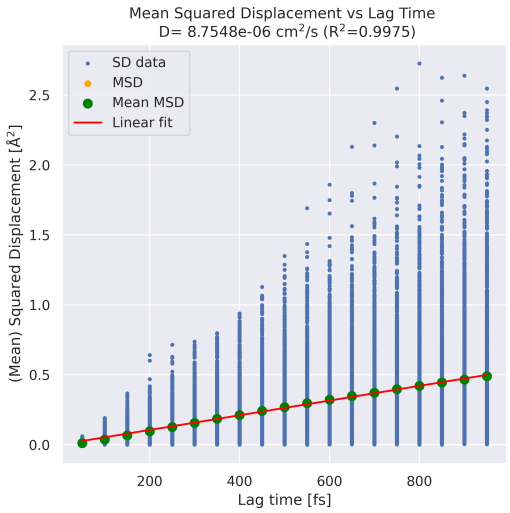
Figure A.8: Metal solute trace plot of a Ga nanoparticle with Pd as metal solute.



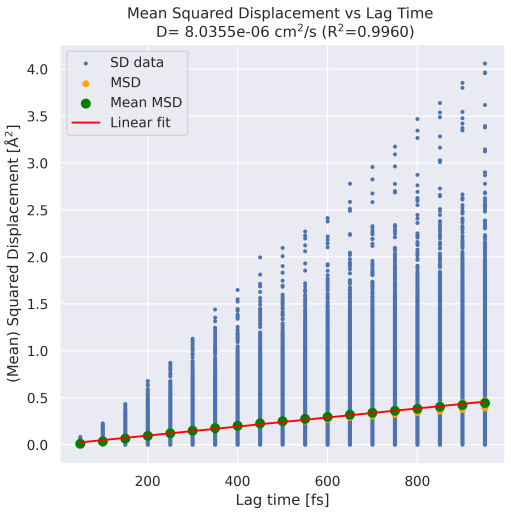
(a) 342 atoms

(b) 1568 atoms

Figure A.9: MSD plot of the Ag metal solute to find the diffusion constant. Where MSD is the mean over all different time origins at a certain lag time and the mean MSD is the mean over all metal solute atoms for each lag time.

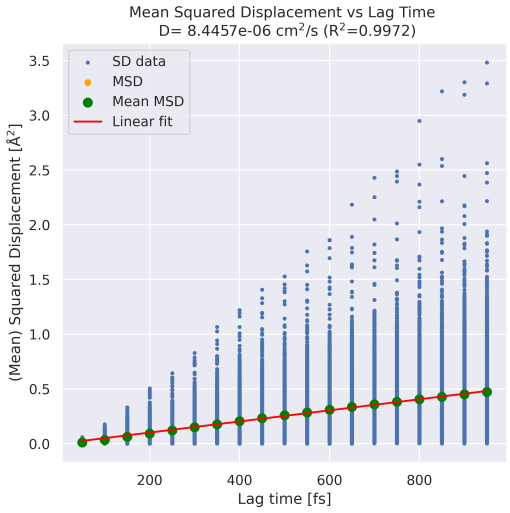


(a) 342 atoms

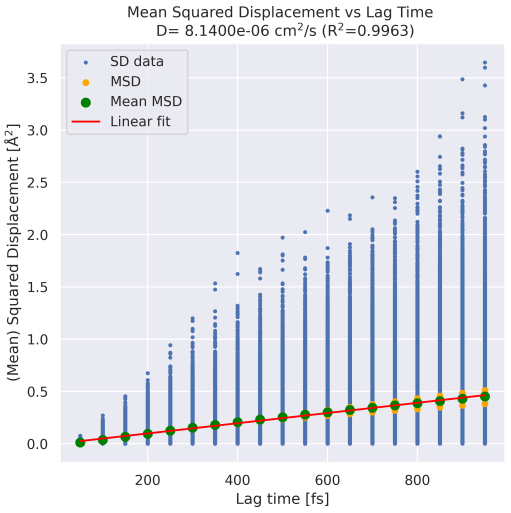


(b) 1568 atoms

Figure A.10: MSD plot of the Au metal solute to find the diffusion constant. Where MSD is the mean over all different time origins at a certain lag time and the mean MSD is the mean over all metal solute atoms for each lag time.

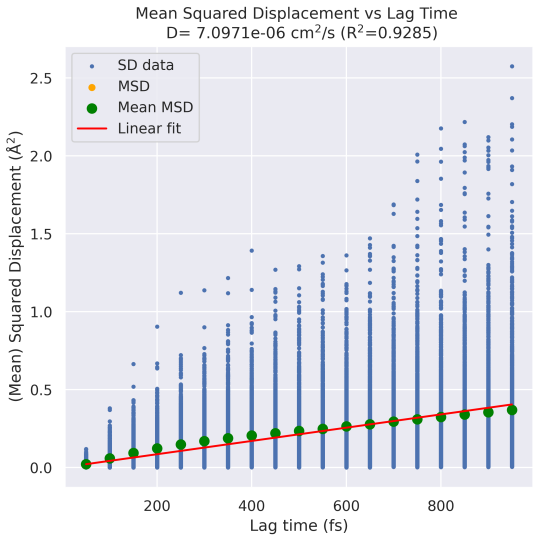


(a) 342 atoms

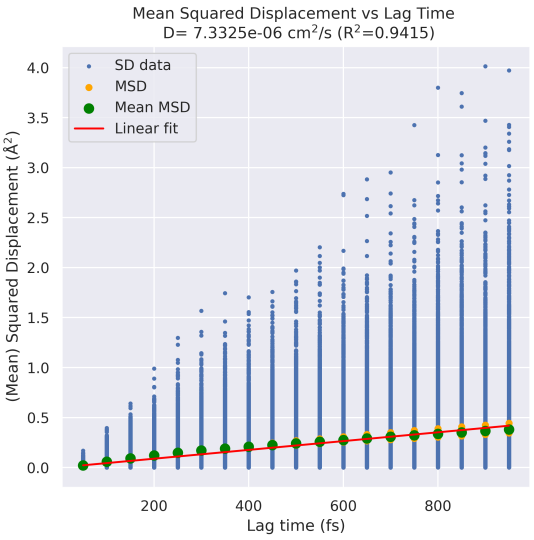


(b) 1568 atoms

Figure A.11: MSD plot of the Bi metal solute to find the diffusion constant. Where MSD is the mean over all different time origins at a certain lag time and the mean MSD is the mean over all metal solute atoms for each lag time.

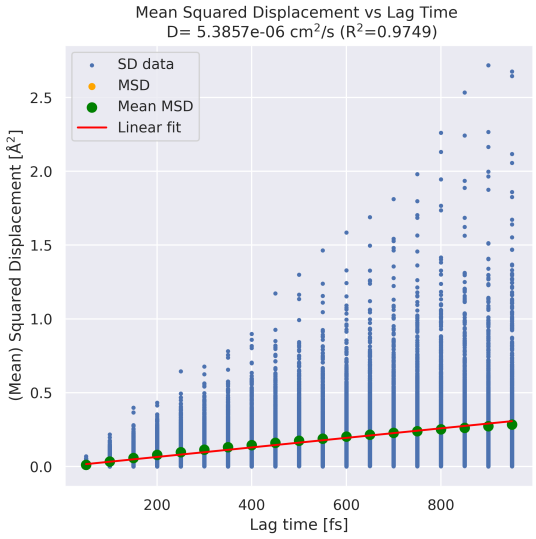


(a) 342 atoms

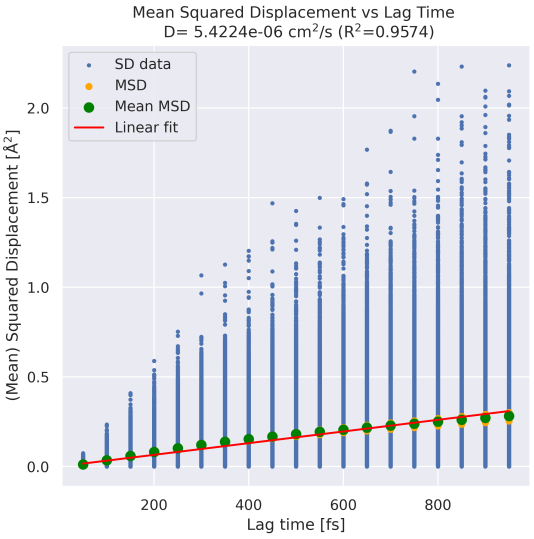


(b) 1568 atoms

Figure A.12: MSD plot of the Pd metal solute to find the diffusion constant. Where MSD is the mean over all different time origins at a certain lag time and the mean MSD is the mean over all metal solute atoms for each lag time.



(a) 342 atoms



(b) 1568 atoms

Figure A.13: MSD plot of the Pt metal solute to find the diffusion constant. Where MSD is the mean over all different time origins at a certain lag time and the mean MSD is the mean over all metal solute atoms for each lag time.

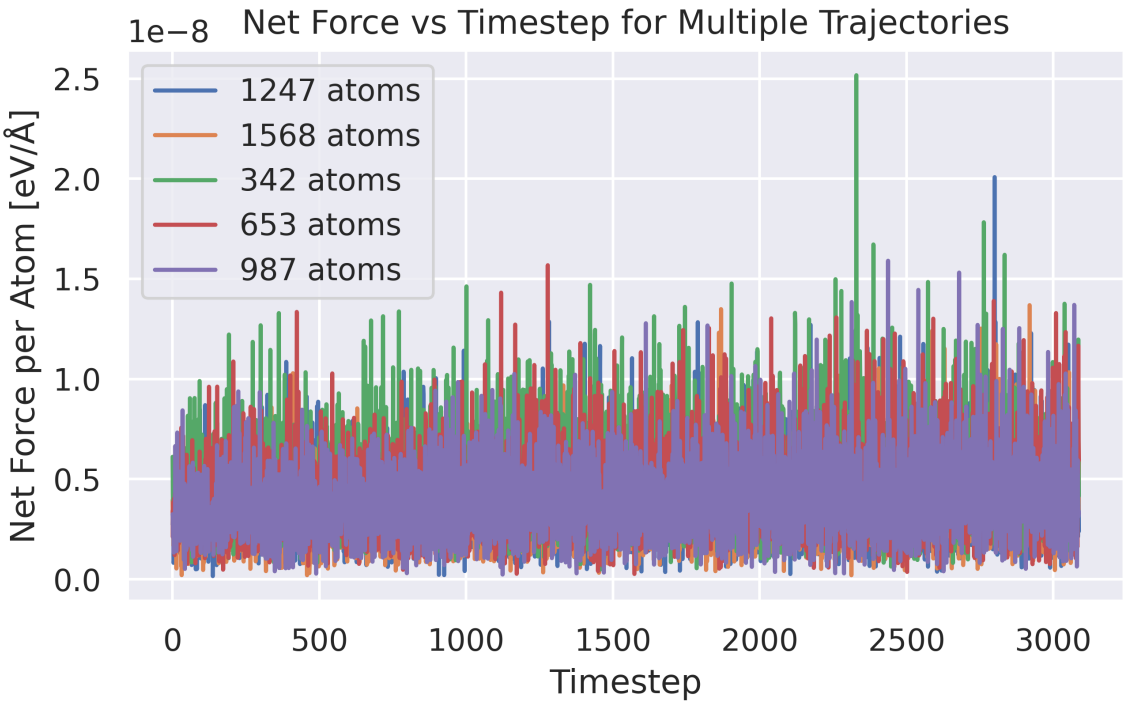


Figure A.14: Net force magnitude over recorded steps (dt=1fs, saved every 50fs) for gallium nanoparticle including Ag as solute.

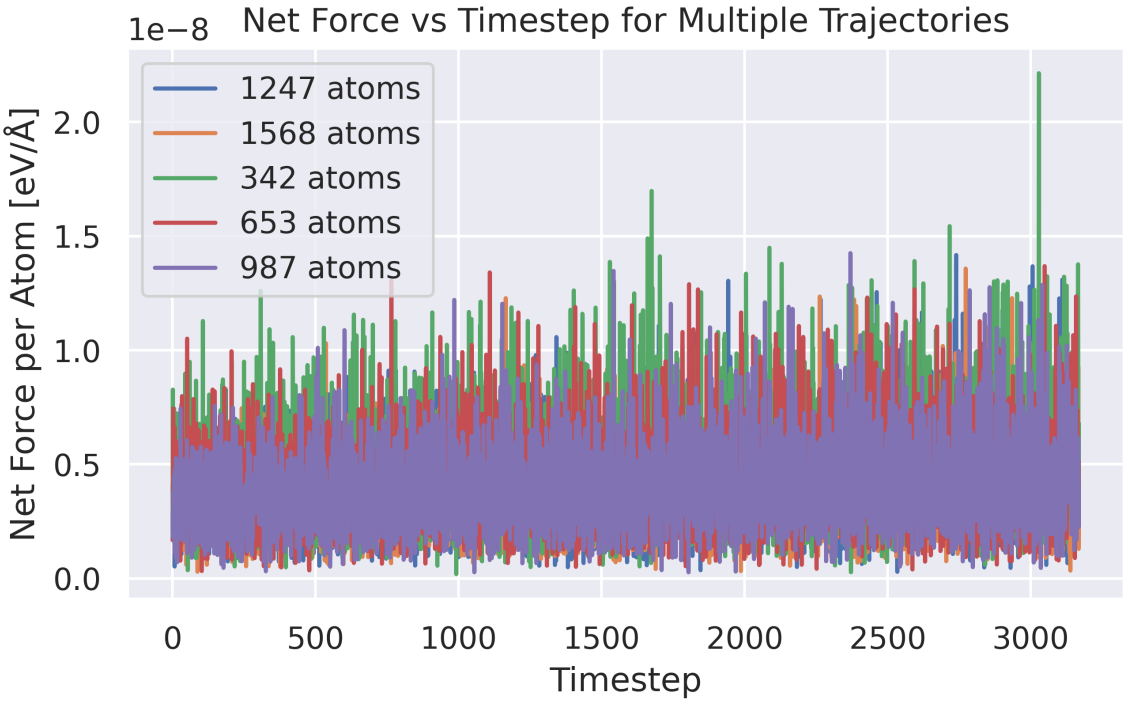


Figure A.15: Net force magnitude over recorded steps (dt=1fs, saved every 50fs) for gallium nanoparticle including Au as solute.

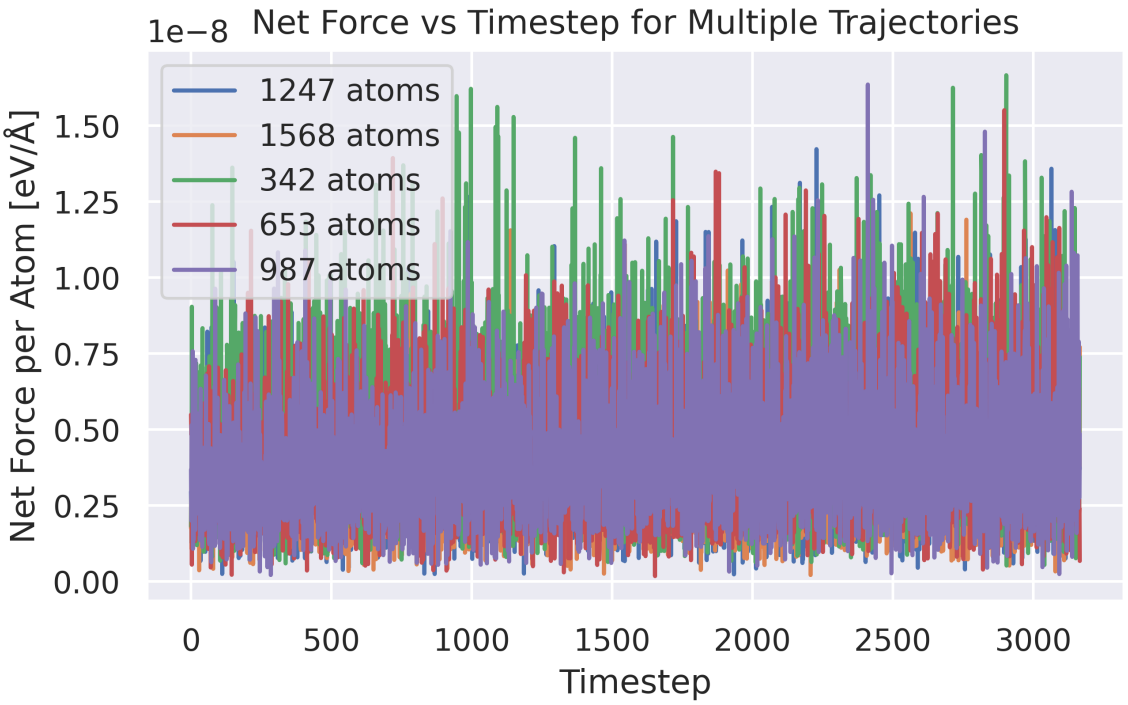


Figure A.16: Net force magnitude over recorded steps (dt=1fs, saved every 50fs) for gallium nanoparticle including Bi as solute.

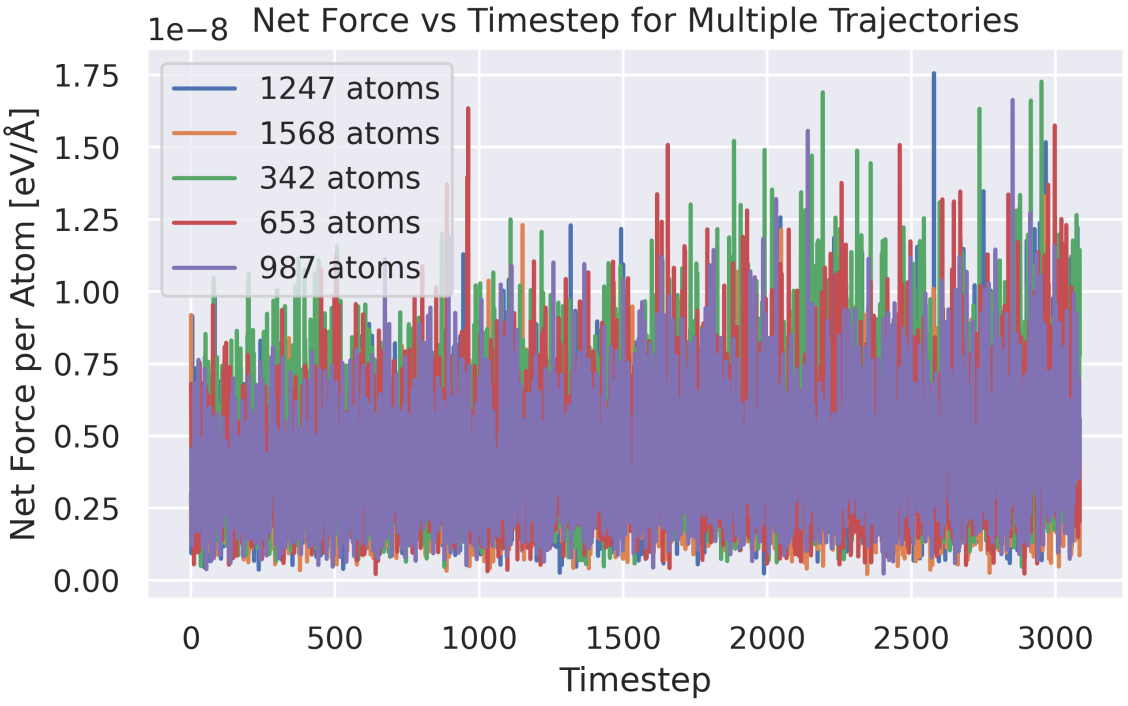


Figure A.17: Net force magnitude over recorded steps (dt=1fs, saved every 50fs) for gallium nanoparticle including Pd as solute.

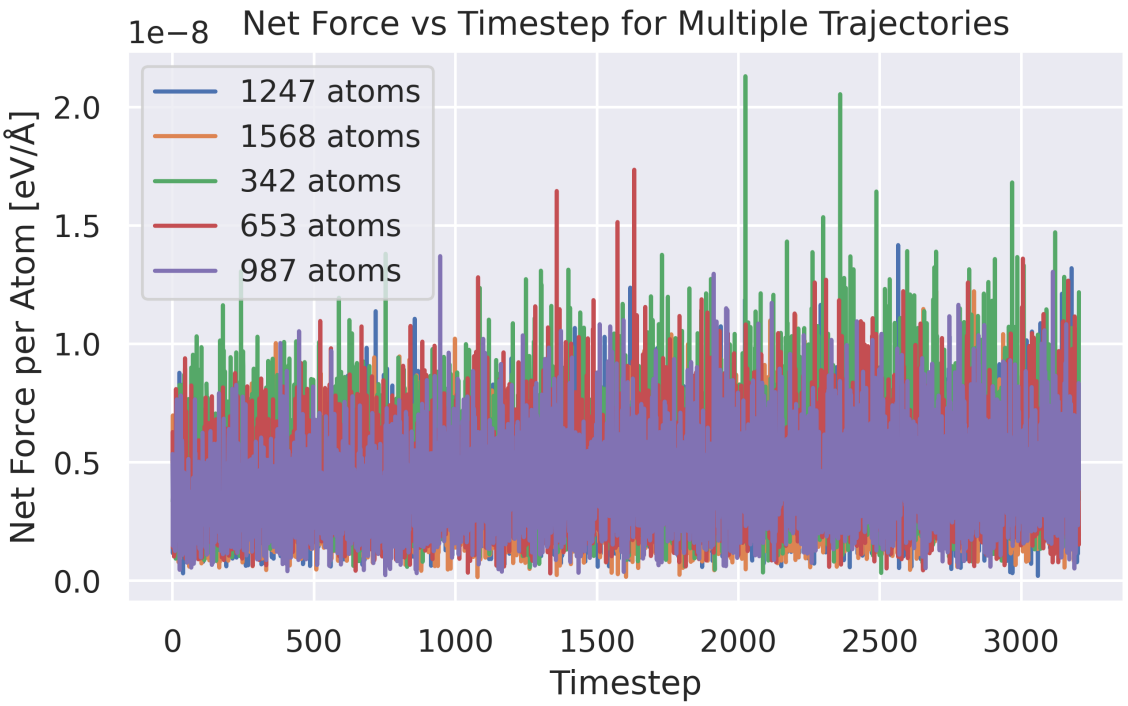


Figure A.18: Net force magnitude over recorded steps (dt=1fs, saved every 50fs) for gallium nanoparticle including Pt as solute.

B

CP2K error investigation

In this chapter we explore issues encountered while using CP2K for ground truth data.

B.1. Method - CP2K

ASE has a CP2K calculator that connects to the CP2K shell in order to run calculations. Using this setup a input file for CP2K could be automatically generated. The SCF convergence was set to 10^{-6} or 10^{-7} in order to provide good continuous data for the ML model to train on. The ADDED_MOS keyword was set to 100 and the electronic temperature was set at 350 or 500 K to stabilize convergence in this metallic system. Gallium has an odd amount of electrons resulting in a net spin in the system so unrestricted Kohn Sham/spin-polarized DFT was done. The basis set was left on the standard DZVP-MOLOPT-SR-GTH and the potential was automatically selected based on XC functional but reverted to GTH-PBE when no special potential was available. The cutoff was left at the default 400 Rydberg.

Performing a force field accuracy benchmark as well as training an ML model requires high accuracy DFT data. As discussed in the methods section tight DFT convergence criteria and other parameters were set to ensure this. Unfortunately this was not enough for CP2K. This resulted in low accuracy force field benchmarks on the foundational models, plateauing ML training and almost random models causing unstable MD.

B.2. CP2K accuracy benchmark

All foundational models showed similar behavior to Figure B.1 on AIMD CP2K data. If we take the DFT data as ground truth we see that the MLIP models show damped forces in a range between approximately -0.8 and 0.8 ev per Angstrom. No correlation between the DFT force and the force field force can be seen.

To investigate if this problem was contained to gallium or to other metallic elements as well, nanoparticles of Indium and Tin were constructed using the same workflow as the original gallium nanoparticles. This gave the following benchmark plot in Figure B.2. Due to the fact that this was a test a small dataset was created. In this case we see a way smaller force range for both the indium and tin nanoparticles, but in general it seems significantly better. Unfortunately due to the small amount of data no real conclusions can be drawn.

Similarly, it was investigated if the nanoparticle structure caused problems for CP2K by repeating the workflow modified for bulk. This gave Figure B.3 and the same behavior is observed.

Bulk copper and gold were investigated as well. Again, the AIMD was run relatively short as this was only a test. Figure B.4 shows this system is already more well-behaved but anomalies remain.

Finally another approach to generating force data was tried, namely rattling the atoms in a nanoparticle. Figure B.3 shows that the model finally aligns somewhat with the force data generated by CP2K.

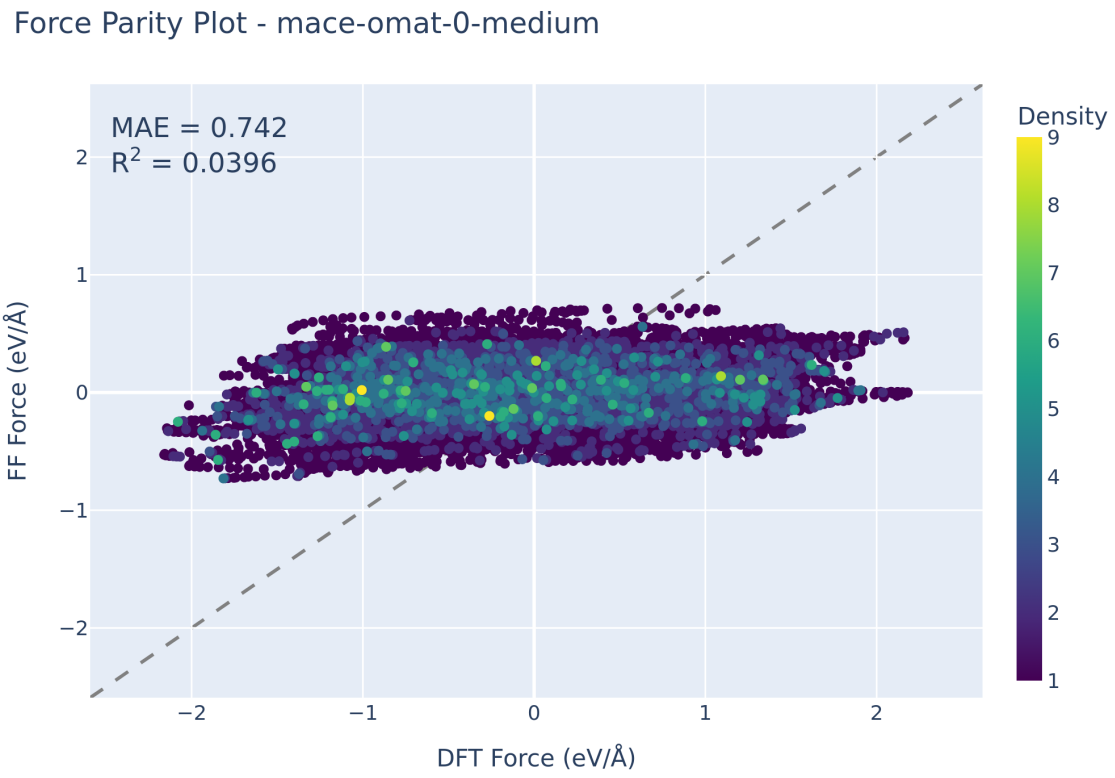


Figure B.1: Force field benchmark (DFT vs mace-omat-medium) on CP2K AIMD data of a gallium nanoparticle.

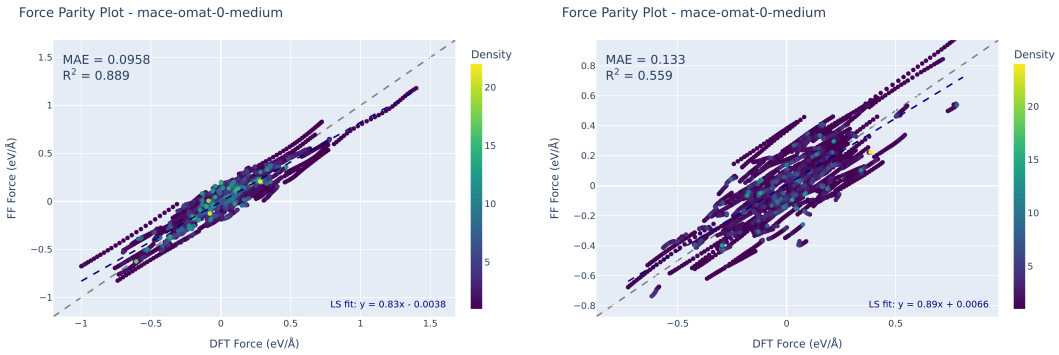


Figure B.2: Force field benchmark (DFT vs MACE-omat-medium) on CP2K AIMD data of a Indium nanoparticle (left) and Tin nanoparticle (right)

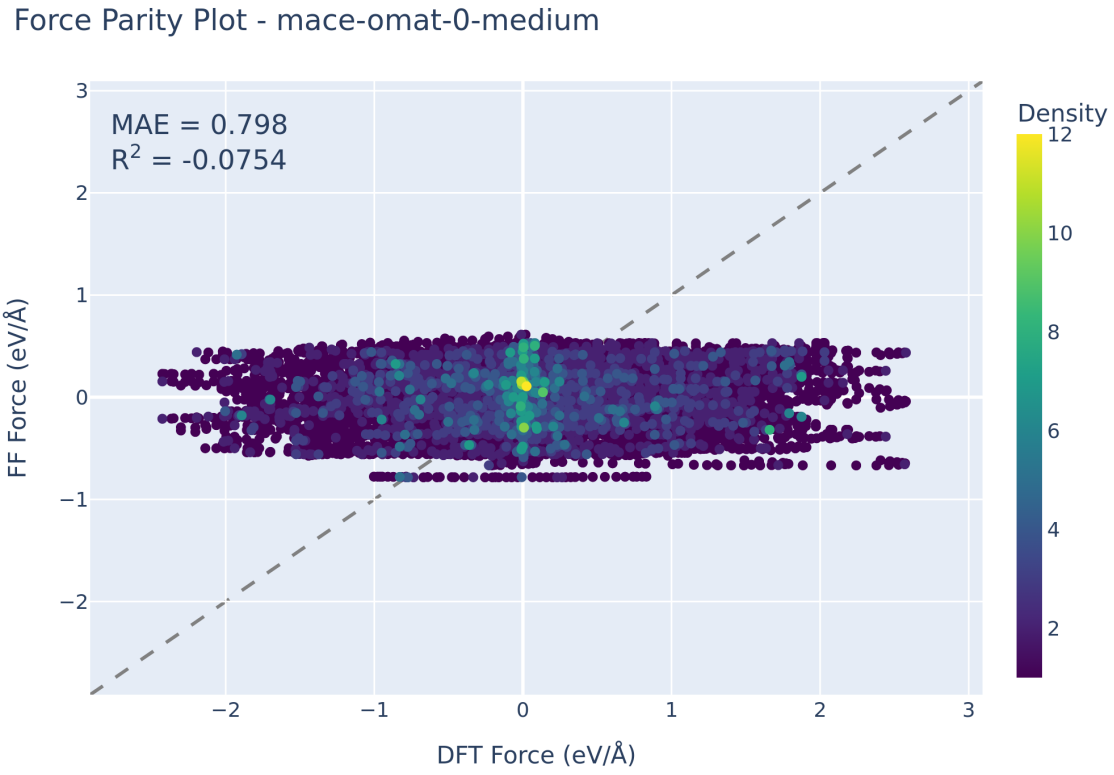


Figure B.3: Force field benchmark (DFT vs MACE-omat-medium) on CP2K AIMD data of a bulk gallium system

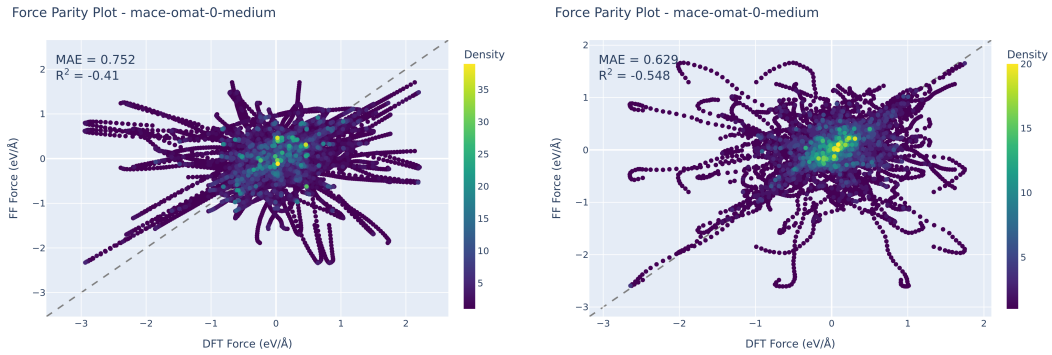


Figure B.4: Force field benchmark (DFT vs MACE-omat-medium) on CP2K AIMD data of a gold bulk system (left) and copper bulk system (right).

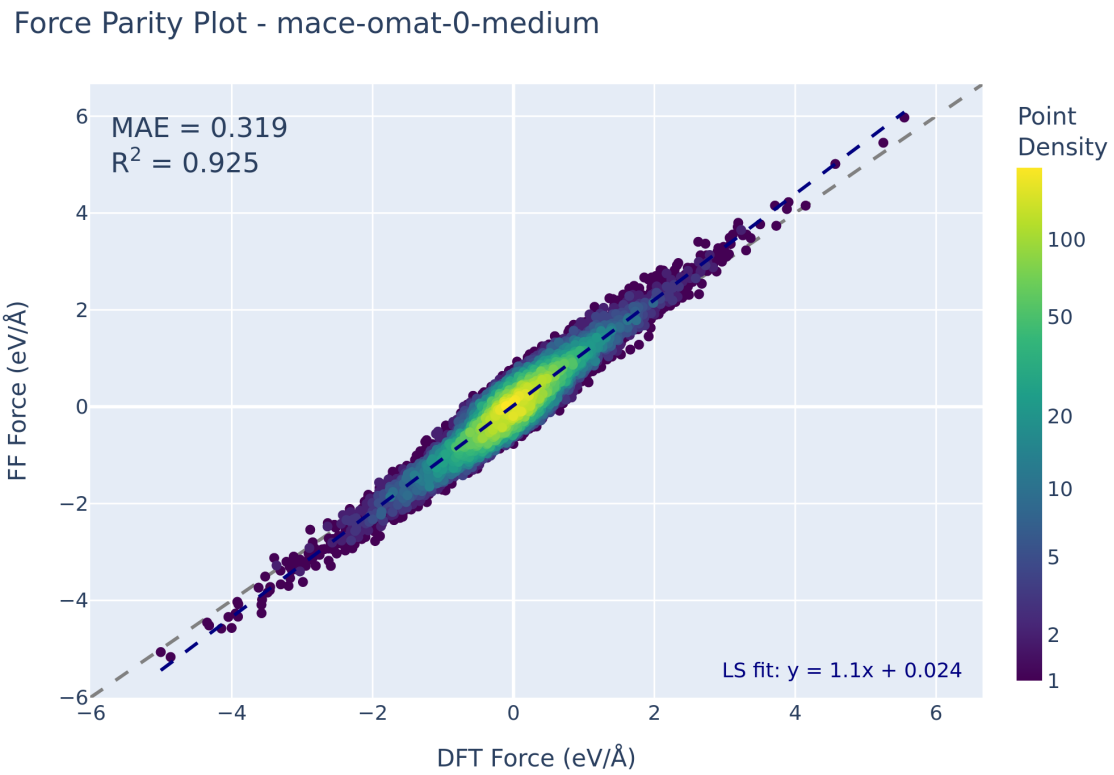


Figure B.5: Force field benchmark (DFT vs MACE-omat-medium) on CP2K rattle data of a gallium nanoparticle.

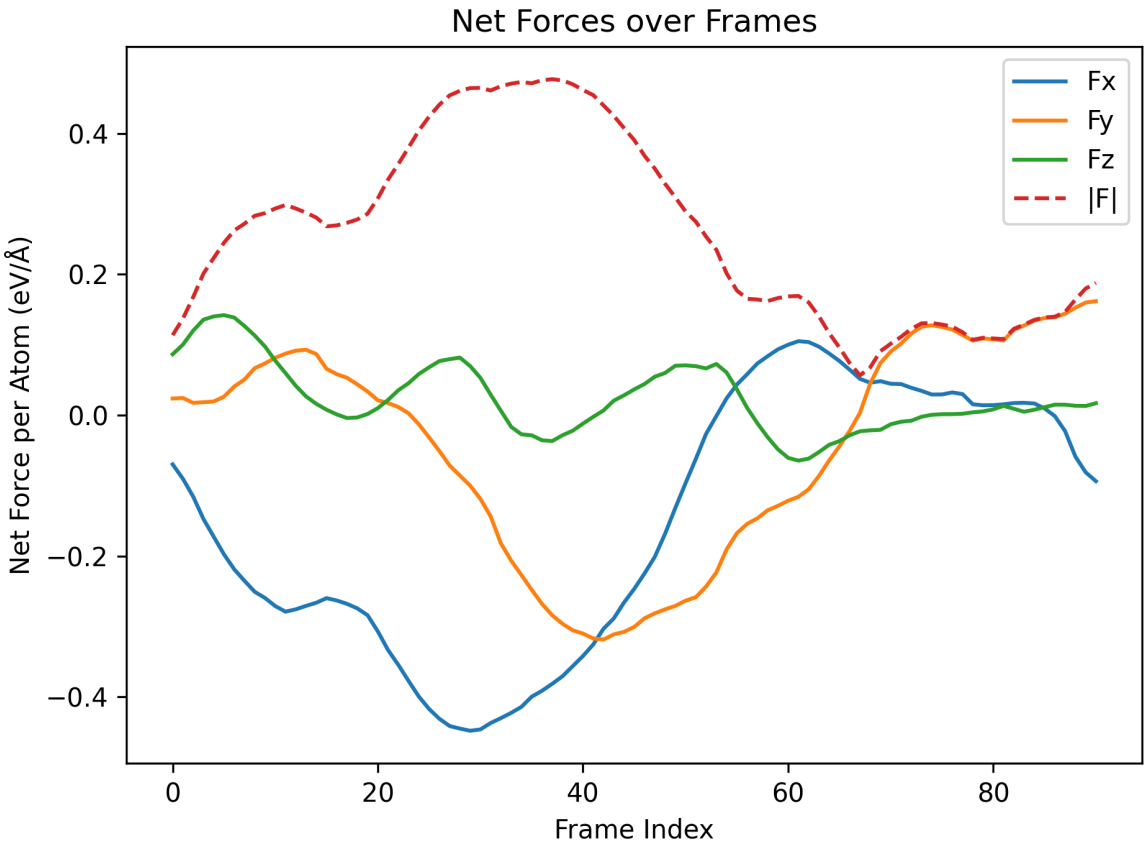


Figure B.6: Net forces per atom of CP2K AIMD run on a gallium nanoparticle.

B.3. Underlying cause

A paper by Kuryla et al. was published that looked into force consistency in molecular datasets. This paper found that some datasets have unacceptable levels of net forces. They investigated an array of DFT engines to assess their ability to generate valid force data that could be used for training of MLIP models. CP2K showed very high net forces even if the convergence criteria was set very high. Because of this result they advised against the use of CP2K for these purposes. In the same paper they investigate VASP and it showed a lot more promise regarding force consistency. This in combination with the widespread use of VASP was the reason all further DFT calculations were done using VASP.

The force consistency of all previously generated DFT data was checked using CP2K. This showed a clear and consistent trend between the net force and the alignment of the MLIP model and DFT data.

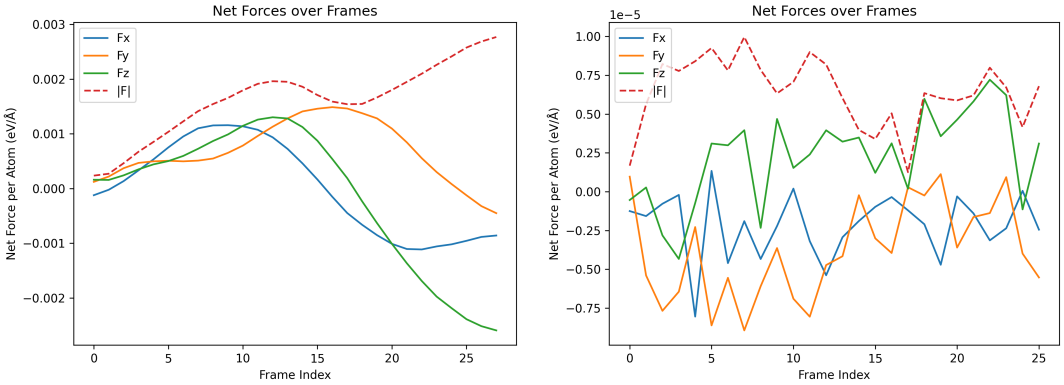


Figure B.7: Net forces of CP2K AIMD data run on a indium nanoparticle (left) and a tin nanoparticle (right).

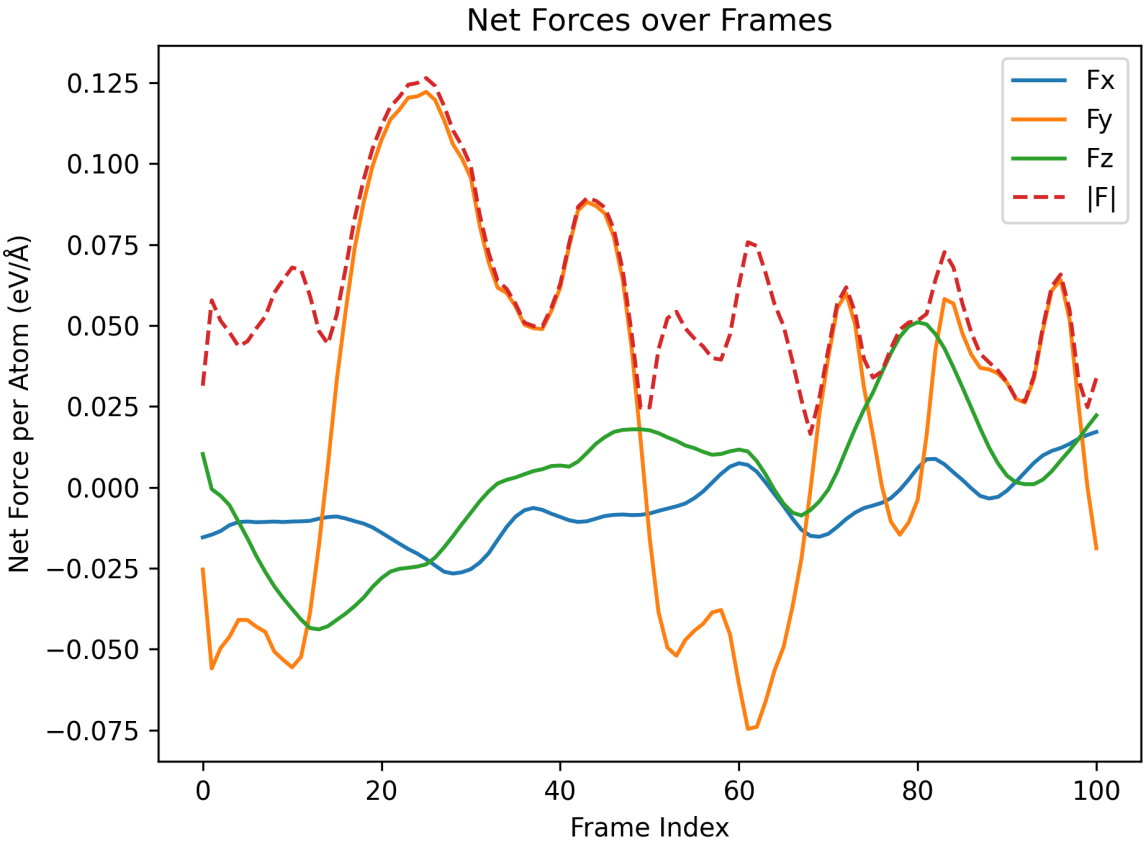


Figure B.8: Net forces of CP2K AIMD run on a bulk gallium system.

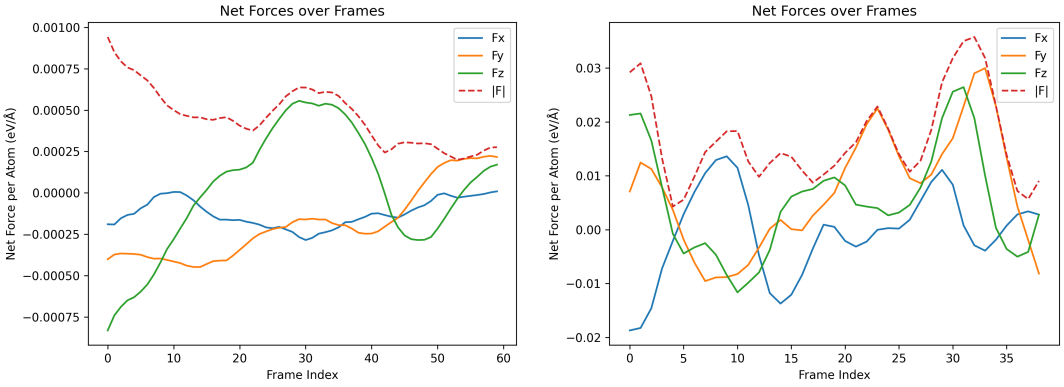


Figure B.9: Net forces of CP2K AIMD data run on a bulk gold system (left) and a bulk copper system (right).

C

Project information

C.1. Acknowledgment

I would like to thank Dr. Rossi for his invaluable support in this project, his guidance was a great help. I also want to thank Prof. Pidko for making this project possible. Finally I also want to thank Dr. Vasileiadis and Dr. Cutz for accepting the role of 2nd and 3rd examiner; this project could not have been completed without their support.

The authors acknowledge the use of computational resources of the DelftBlue supercomputer, provided by Delft High Performance Computing Center.

C.2. Declaration of AI use

In this section the use of AI will be reflected upon. Although its generally advised to show AI use for each section of the report, this will not be done in this reflection because it does not significantly change between the different sections in the report and because AI usage for this project can be better categorized in the three sections below: literature study, methods and writing. In literature study it is about how traditional and AI tools were used in the uncovering of papers and general information gathering of this project. The method section shows the use of AI tools in practical concerns of this project such as coding. In the writing section a statement is made about the use of AI tools during the writing process of this report.

C.2.1. Literature study/Theory

For literature search I initially used standard techniques such as ScienceDirect, google scholar, searching the web, etc. to search for relevant papers. When I had found some papers, combined with papers already read as preparation for my MEP which were suggested by Dr. Rossi. This created a basis of papers to start from. From this basis I could read other papers that were used as citations in my basis. Later to expand my literature search I used Semantic scholar and Elicit (as suggested by the AI for literature book by Library Education Support & Centre for Languages and Academic Skills of TU Delft) to obtain relevant references which would broaden my understanding of various topics relevant to the theory section and understanding in general. I found that compared to native searching it was better at providing relevant papers. With that said the usefulness of all tools is limited to the user and itself meaning the relative usefulness is confounded with my ability to search for papers and the use of these new AI tools. In general it is probable that the use of AI tools is easier or conforms more to already existing skills allowing for better results.

C.2.2. Methods

For writing code AI tools such as GitHub Copilot and ChatGPT were used for four purposes. The first purpose was to search for relevant Python packages and functions that could help in the project. The second was as advanced documentation, when a function signature or syntax was unknown or not clearly documented AI tools were consulted for additional explanation. Thirdly the debugging or

reviewing of code, due to the rising complexity over time as a project grows bugs can crop up, and these bugs can be found using some AI tools. Similarly AI tools can review code and give feedback on possible anti-patterns and bugs. Finally GitHub Copilot's inline suggestions were used to code more efficiently. AI tools can provide good advise over the use of well-known packages and patterns. The use of inline suggestions is very variable; boilerplate code used in plotting and easy tasks is done very well, but architectural choices and complex numeric manipulations using Numpy or PyTorch are relatively bad due to the need to extrapolate from your own code or small context windows. These tools provide a good stepping stone or suggestion but when the goal is learning additional steps must be taken.

C.2.3. Writing

Non-AI tools such as Grammarly were used to spell-check the final report. No AI-tools were used in the writing of this Master Thesis report.

C.2.4. Reflection of AI-use

AI tools were very helpful for this project. They helped with literature search and coding. Generally all tools performed decently though not perfectly. AI tools that helped in literature search did give some papers that were helpful but their inability to access papers not publicly accessible can be a significant hindrance thus it was used to expand on an existing base of papers. AI tools helping with coding helped to gain direction and code suggestions when it was not complex, but when complex systems were created knowledge was still required for the specific application in this project.

C.3. Data management

All code can be found in the public GitHub repository called ”Atomic_intelligence_gallium_SCALMs”. All data generated in this project is stored on the drive managed by the ISE group of TU Delft.

D

README

D.1. Unraveling the atomic intelligence of liquid metal catalysts

Student: David Ullersma
Degree: Chemical Engineering (CE)
University: TU Delft
Resposible supervisor: Evgeny Pidko (faculty: Applied Science)
Daily supervisor: Kevin Rossi (faculty: Mechanical Engineering)

This master thesis project (MEP) explores the dynamics of Ga-rich liquid metal catalysts using state of the art MLIPs.

Scripts were made to facilitate the dynamics investigation.

Different MLIP models have clashing requirements. This means that it is advised to uses different venvs for each one. To reduce the size of the venvs it is advised to use uv. To see the requirements of the venvs used in this project see the requirements dir. In this dir you can find an additional file called ExtraRequirements, here additional external libaries and model configuration is explained (Be aware: GPU acceleration libaries can fail silently!).

To check the arguments of a script you can use the *-h* flag or *–help* to see the documentation.

D.1.1. benchmark

Benchmarking is done on two levels:

- Computational efficiency
- Accuracy

Benchmark force field accuracy

The script *benchmark_FF_parent.py* runs a benchmark on each of the models in the model_config yaml using the provided (ground truth data: DFT generated) ASE readable file. Then if requested it plots a series of plots to analyze the model accuracy and analyze the error correlation and distribution.

Arguments:

- trajectory_files: one or more ase readable file to run the benchmark on
- working_dir: the directory in which the bench is run (to separate between benchmarks)
- model_config_path: the path to the model config yaml (below is an example of the yaml structure)
- plot: plot the benchmark
- copy_trajectory: copy the trajectory file on which the benchmark is run to the working dir for reference

- force_rewrite_flat_hdf5: if a flat hdf5 file (for plotting) is already cached it will not automatically recreate it when a new model is added to the benchmark dataset. Turning this on will force the code to recreate the flat hdf5 file to include the new models
- committee_model_names: all model names to make into a committee of models (WIP)

The model config yaml follows the following structure:

```
model_architecture:
    models:
        specific_model:
            model_path: 'path/to/specific_model.model'
    venv_path: 'path/to/venv/python3'
```

for example:

```
MACE:
    models:
        mace-omat-0-medium:
            model_path:
            ↪  /home/david/MEP/MEP_DU_KR_26/input_data/input_data/MACE/foundational_models/mace-omat-0-medium.
    venv_path: '/home/david/MEP/MEP_DU_KR_26/.venv_devMACE/bin/python3'
ORB:
    models:
        orb-v3-conservative-inf-omat: # currently implemented with the pretrained package
        orb-v3-direct-inf-omat:
    venv_path: '/home/david/MEP/MEP_DU_KR_26/.venv_ORB/bin/python3'
```

features include:

- compact binary hdf5 format: in service of large datasets and interoperability hdf5 files are used to store the force field benchmarking data. The hdf5 files are structured so that an ASE Atoms object can be recreated if needed. Although this format is compact plotting on atom and frame/structure attributes requires conversion of the data. *data_processing_utils.py* provides functions in order to facilitate conversion between formats.
- torchsim inference: torchsim supported models run all static reevalutations of the provided systems in batches to speedup
- ASE calculator interface: in case torchsim fails or is unavailible it can fallback to using a ASE calculator, but this is/can be slower due to the non-batched inference of the model (depending on GPU VRAM).
- custom models: the model config framework makes custom (MACE) models possible. The current framework is build with the possibility of other types of custom models.
- virtual environment separation (via subprocesses): Each model architecture can have its own virtual environment (in the model config) to prevent clashing dependencies between model architectures.
- plotting: If the script is asked to plot the benchmarked data it will (currently) provide the following plots if able: force and energy parity plot for each model using pymatviz which implements scatterplot binning to reduce the size and computational effort to use this plot, force parity plot for each model using pymatviz but with corrected statistics based on the underlying dataset (pymatviz computes statistics over the binned data), custom force parity plot with additional boxplot for error distribution, error correlation plots of error versus DFT and ML force size, error distribution plots for each model. WARNING large datasets can crash this script and when energy data is not corrected it can fail the energy parity (WIP) plot OLS.

Returns: 4 dirs of plots of each model:

- error_analysis: plots correlating absolute error to DFT force and ML force (WIP) and error distribution plots.
- force_plotly: custom plotly parity plots with box dist and force error gradient (WIPs)

- force_pymatviz: raw pymatviz parity plots (stats are computed on the binned values: favors low density areas in the parity plot).
- force_pymatviz_stats: modified pymatviz parity plots with stats based on the underlying data.

Benchmark computational efficiency (WIP)
benchmark_comp_parent.py does MD sim using ASE of a preset number of steps on the given input structures (via the input structure list or input dir: *.xyz glob file search) an even spread of system sizes is recommended. When it runs the MD it measures the CPU, Memory, VRAM and GPU utilization and writes it to a csv file using a sampler thread that samples the different metrics. WARNING: CPU and memory utilization **is** measured based on PID, but GPU and VRAM utilization **is not** based on PID. It is recommended to use a non classical model (that does not use the GPU) as reference for example a EAM model using the KIM repository.

Arguments:

- input_directory: the directory to search for input files to benchmark (searching *.xyz files)
- input_strucutre_list: one or more structures can be provided for benchmarking
- total_steps: amount of MD steps to run for benchmark. Decrease to speed up the benchmark, but be warned that when decreased the initial setup tasks such as graph creation can skew the numbers.
- time_step_fs: the time step size used for MD in femto seconds.
- temperature_K: the temperature using for MD in Kelvin.
- model: specify a specific model if only one is required instead of all available

Returns: A dir for each model. In each dir the trajectories generated to benchmark the model and 2 csv files: one with the averaged results and one with the complete set of results.

D.1.2. ML training

In this directory config files, setup scripts, final models and other files related to the training of the ML models can be found.

D.1.3. dynamics

The main interface is *run_dynamics_parent.py*.

Arguments:

- structure_files: The structure file(s) to run MD on (ASE readable format).
- working_dir: The working directory to run the MD in.
- model_config_path: The path for the config yaml file for the models to evaluate. (default:“./model_config.yaml”)
- n_steps: The number of MD steps to run for each structure.
- time_step_fs: The MD time step in femtoseconds.
- temperature_K: The MD temperature in Kelvin.
- data_file_name: The name of the HDF5 data file to store the MD and analysis results in. (default: “MD_dynamics_results.h5”)
- plot: Whether to plot the results.
- RDF_analysis: Whether to run full RDF analysis (can be heavy when evaluating large systems).
- RDF_partial_analysis: List of atomic number pairs (e.g. ‘78-31’ for Pt-Ga) to run partial RDF analysis on.
- relative_position_analysis: Chemical element atomic number to run relative position analysis on (e.g. 78 for Pt).
- net_force_analysis: Whether to run net force over time analysis.
- diffusion_constant_analysis: Atomic number to run diffusion constant analysis on (e.g. 78 for Pt).

- models: List of models to be evaluated (must be included in model config file). If not provided, all models in the config file will be evaluated.
- consolidate_h5md_files: Whether to consolidate .h5md files into a single file per model after MD simulations and which approach (h5repack is better but requires dependency) otherwise relative hdf5 links are used.
- remove_h5md_files: Whether to remove the .h5md files generated during MD simulations.
- elements_to_consider_surface: List of chemical element symbols (e.g. 'Pt', 'Ga') to consider as surface atoms for relative position analysis if any. If empty, all atoms are considered as surface atoms for relative position analysis.

This main file uses the following files to do a MD simulation and analyze the data:

- *pytroch_diffusion_const.py*: does the diffusion constant analysis using a custom pytorch function. It computes the squared deviations for all metal solutes for all combinations of lag times and time origins with a set lag time max. To reduce the amount of SD to be calculated the function provides the option to subsample the amount of time origins and lag times discretized by the time between recorded frames.
- *pytorch_rdf.py*: implements functions to compute any RDF/PDF from trajectory data. The *compute_mean_rdf_pytorch* function is the highest level of abstraction: Can be used with static labels (for example: atomic numbers) via 1D mask and dynamic labels (for example when clustering) via 2D mask. mask_a is the main mask which is used for normalization (meaning 'Ga-Pt' is unequal to 'Pt-Ga' due to normalization), both masks can be used to get partial RDF/PDF's. WARNING: can use a lot of memory to compute large systems/trajectories, manual subselection of frames may be necessary.

Returns: Depending on the arguments plots of the traj, but always a h5 file which is all h5md torch-sim files concatenated and if not set to remove all h5md torch-sim files will be saved as well (Recommended).

D.1.4. geometry

three main scripts reside in this directory:

- *add_organic_molecules_to_NP.py*
- *create_Ga_NP.py*
- *structural_cluster.py*

add_organic_molecules_to_NP.py is used to place organic molecules around an approximately spherical nanoparticle.

Arguments:

- organic_molecule_paths: Paths to organic molecules (ASE readable)
- num_organic_molecules: Number of organic molecules to add (distributed over the organic molecules provided) (default: 10)
- input_nanoparticle: Path to input nanoparticle (ASE readable).
- output_nanoparticle: Path to output nanoparticle with organic molecules (ASE writable)
- distance_from_radius: Distance from nanoparticle radius to place organic molecule in Angstroms (default: 2.0)
- min_distance: Minimum distance enforced between organic atoms and NP atoms to avoid overlap in Angstroms (default: 2.0)
- max_attempts:Maximum attempts to place organic molecule without overlap (default: 10)
- distance_increment: Distance increment to move organic molecule if overlap occurs in Angstroms (default: 0.5)
- random_orientations: Apply random orientations to organic molecules before placement

- n_atoms_average_for_radius: Number of furthest atoms to average for nanoparticle radius calculation (default: 10)

Returns: NP in xyz file format with organic molecules placed around it.

create_Ga_NP.py uses wulff construction to construct a valid solid Ga NP and then uses MACE to melt it using MD. Noble metals can be added before or after melting MD to get the noble metals in equilibrium or not.

Arguments:

- sizes: Target size of the nanoparticle in number of atoms.
- mace_model_path: Path to the MACE model file.
- add_catalyst: Whether to add catalyst atoms to the nanoparticle.
- catalyst_elements: Element symbol(s) of the catalyst to add. If multiple, will create separate nanoparticles for each element.
- catalyst_fraction: Fraction of Ga atoms to replace with catalyst atoms.
- order_adding_catalyst: Whether to add catalyst before or after Wulff construction.

Returns: sizes * 1 or catalyst_elements different NP structures in xyz format, trajectory is written in .h5md format as created by torch-sim or .traj when ASE is used (by default: torch-sim).

geom_gen.py includes functions used by *create_Ga_NP.py* but this file can also be called as a script to construct and save a Ga alloy

Arguments:

- input_paths: Paths to the input structure file
- fraction_cat: Fraction of Ga atoms to substitute with catalyst atoms
- cat_symbol: Chemical symbol of the catalyst atom
- output_paths: Paths to save the output structure files if None, will save to {input_path}_with_{cat_symbol}_cat.xyz

Returns: A random gallium alloy nanoparticle for each input path.

structural_cluster.py uses MACE descriptor to cluster giving insight into the different environments of a trajectory (for example surface and bulk).

Arguments:

- h5md_path: Path to the H5MD file (as created by torch_sim)
- output_dir: Directory to save output files ('./MACE_cluster')
- model_path: Path to the MACE model file
- step: Step size for frame selection from trajectory (gives every nth frame) (default: 10)
- use_cache: Whether to use cached descriptors and pipeline if available
- set_seed: Random state seed for reproducibility (WARNING: reduces performance due to disabled parallelism)
- n_cluster: Number of clusters for BisectingKMeans
- cluster_dim: Dimensionality for clustering
- min_dist_umap: Minimum distance parameter for UMAP
- n_neighbors_umap: Number of neighbors parameter for UMAP

Returns: 3D labeled structure, 3d umap projection, COM distance label histogram and pairwise label PDF plot for PCA and UMAP dimensionality reduction for clustering.

D.1.5. HPC

This directory includes scripts used on a HPC system and example SLURM submit scripts for these scripts:

VASP_nanoparticle_rattler_v2.py rattles a given NP geometry and recomputes it with VASP in order to get ground truth data.

Arguments:

- input_structure_path: (relative/absolute) path to the input structure file.
- output_dir: Directory to save output files.
- rattle_stdev: Standard deviation for the rattle displacement (in Angstroms).
- num_rattles: Number of rattled structures to generate.
- alloy_element: Element symbol of the alloying element.
- alloy_fraction: Fraction of atoms to replace with the alloying element (between 0 and 1).
- pre_converge: Whether to pre-converge the structure before rattling (reduce max force to below 0.5 in max 20 steps using BFGS algorithm), automatically done when alloying.

Returns: xyz file with the rattled and VASP computed structures including force data.

VASP_nanoparticle_rattler_organic_mol.py is a combination of the organic molecule placer and the VASP rattler script and therefore exposes similar arguments to the two mentioned scripts.

data_set_packager.py packages xyz files created by the rattle script into a dataset that can be processed further to use as training data for training/finetuning a MACE model. Splits on files to prevent data leakage.

Arguments:

- input_dir: (relative/absolute) path to the input dataset directory.
- file_pattern: File pattern to match dataset files (e.g., “*.xyz”).
- train_val_test_split: Train/Validation/Test split percentages (e.g., “80-10-10”).
- extra_files: Additional files to add to the training set.
- amount: number of datasets to package.
- output_dir: Output directory for the packaged dataset.
- force_threshold: Maximum allowed force (in eV/A) for structures to be included in the dataset.
- elements_in_test_valid_set: List of elements that must be present in the test set structures.

Returns: n directories with randomly packaged datasets

DFT_SP_energy_vasp.py calculated the E0 for a certain element as used for training a MACE model.

Arguments:

- element: Element symbol of the atom to simulate (default: Ga).
- output_dir: (optional) output directory. If not provided, uses current working directory.

Returns: xyz file with the element and its energy. Uses spin polarized DFT to most accurately find the E0 energy.

benchmarkDFT_FF_singlepoint_vasp.py uses a ASE calculator to interface with vasp to recalculate a ground truth trajectory. This for example can be used to check the influence of convergence criteria or XC functionals. (WIP: includes hardcoded variables)

Arguments:

- input_structure_path: (relative/absolute) path to the input structure file.
- functional: XC functional to use in CP2K (default: PBE_VASP).
- SpinPolarized: use Spin polarization in recompute (default: False).

- basis_set: Basis set to use in vasp (default: N/A).
- SCF_convergence: SCF convergence threshold (default: 1.0E-7).
- SpinPolarized_DFT: Spin polarization in DFT reference data (default: False).
- csv_path: Path to the CSV file to save benchmark results (default: predefined path).

Returns: CSV with benchmark data including energy, force, basis_set, etc.

D.1.6. Misc

Four files exist outside the above mentioned directories these files include data processing, dynamics and MD utils and calculator setup that provides the interface to use multiple different calculators such as ASE and torch-sim. Most functions in these scripts are deprecated and are no longer used in the main scripts of this repo only *calculator_setup.py* and *data_processing_utils.py* are still used.

data_processing_utils.py contains three functions callable using a CLI typer interface:

torchsim_h5md_to_xyz converts a torchsim h5md file to a xyz for compatibility with other software packages such as OVITO.

Arguments:

- torchsim_h5md_path: Path to the torchsim H5MD trajectory file.
- output_xyz_path: Path to save the output XYZ file.

Returns: One xyz file with the trajectory

concat_ts_h5md can concatenate multiple h5md files into one similar to the structure used by *run_dynamics_parent.py*, this can be used if a HPC job fails or exits earlier causing the concat file to be empty.

Arguments:

- output_path: Path to the concat file.
- input_paths: List of paths to h5md files (tip: use UNIX wildcards e.g. path/to/files/*.h5md)
- model_architecture: Model architecture as set in the config YAML e.g. MACE, ORB.
- specific_model: The specific name of the model as defined in the config YAML
- structure_file_names: Names of the structure files for the concat file
- replace_suffix_to: if the structure file names are not defined it automatically tries to reconstruct them from the provided input paths and sets these suffixes for the structure files. Can only be done with the specific file naming pattern used in *run_dynamics_parent.py* (‘{specific model_name}_MD_{structure_file_n

Returns: one file with all the trajectory data

split_concat_ts_h5md splits the concat file into the small h5md torchsim files that can for example be used in clustering.

Arguments:

- input_path: Path to concat file.
- output_dir: Path to directory to put all torchsim h5md files in.
- model_architecture: Model architecture as set in the config YAML e.g. MACE, ORB.
- specific_model: The specific name of the model as defined in the config YAML

The folder with miscellaneous scripts contain mostly old but significant scripts that were used in this project. One is of particular importance:

trace_summary_plot.py creates the solute trace summary plot as seen in the report.

Arguments:

- input_dirs: List of input directories containing torchsim files (.h5md) must have the same size NPs for each input_dir

- atomic_symbol_trace: List of atomic symbols to trace (e.g. ‘Au’, ‘Pt’) in the same order as input_dirs
- output_dir: Directory to save the output plots
- final_fraction: Fraction of the trajectory to consider from the end (e.g. 0.5 means last half of frames)
- max_com_distance: Maximum center of mass distance to consider for histogram range
- time_interval_fs: Time interval between frames in femtoseconds
- element_symbols_surface: List of atomic symbols to consider as surface atoms (e.g. ‘Ga’). If empty, all atoms are considered.