DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Evaluating Machine Learned Interatomic Potentials for Practical Simulations

Richard Strunk

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Evaluating Machine Learned Interatomic Potentials for Practical Simulations

Author:	Richard Strunk
Supervisor:	Prof. Dr. Daniel Cremers
Advisor:	Karnik Ram
Submission Date:	04.11.2025

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 04.11.2025 Richard Strunk

Acknowledgments

I'd like to express my highest gratitude towards Karnik Ram for accepting and guiding me through this project over the last months. Available even during weekends to answer questions, provide feedback, and course-correct my exploratory experiments, his guidance was detrimental to this work and to my personal growth as a researcher. I'd also like to sincerely thank my friends and family for their unwavering support and encouragement throughout my studies, especially during the more challenging times.

Abstract

In this work we explore the capabilities and limitations of machine learning interatomic potentials (MLIPs) for molecular dynamics simulations. We review the architectures and design choices behind state-of-the-art MLIP models, and evaluate their performance on a series of custom benchmarks designed to probe their strengths and weaknesses in various scenarios. We find that while MLIPs are significantly faster than traditional ab initio methods and scale well with system size, they exhibit unique flaws and instabilities that limit their practical applicability.

Contents

1. Introduction

1.1. Problem Statement

Driven by our curiosity about the nanoscopic world, simulating atoms – and by extension molecules – at the quantum level is a highly active area of research going back a century [1, 2]. From materials science to drug discovery, biophysics, fusion research, and catalysis: accurate tools capable of simulating millions of atoms for nanoseconds within a practical timeframe could have profoundly positive implications from academia and industry to society at large [3, 4, 5, 6, 7, 8]. Among the particularly interesting simulation candidates are Metal-Organic Frameworks (MOFs), capable among other things of actively absorbing carbon dioxide for carbon capture to storing hydrogen [9, 10]. Traditionally, two options exist for what is called Molecular Dynamics (MD): Use **prohibitively expensive** ab-initio (from first principles) methods for quantum-level accuracy, or settle for **significant inaccuracy** by using inexpensive empirical potentials [11, 1]. In recent years Machine-Learning Interatomic Potentials (MLIPs) have emerged, striving to offer the speed of empirical potentials while maintaining the accuracy of ab-initio methods like DFT, essentially providing the best of both worlds [12, 13, 14]. However, their practical applicability is still an open question of ongoing research in the field [15, 16, 17]. The general black-box nature of all Machine Learning (ML) models, with the complex architectures, specialized training procedures and datasets of MLIPs, paired with their slow integration into commonly used MD software programs pose a significant barrier for researchers and practitioners to use, understand and adopt this promising new technology [17].

1.2. Research Questions & Objectives

This thesis aims to tear down the barriers of blind skepticism and unfounded enthusiasm surrounding MLIPs by not only providing a comprehensive comparative analysis of existing models and their architectures, but also by evaluating their applicability in a practical workflow. What are the strengths and weaknesses of different MLIP models? How accurate and reliable are they when compared to DFT ab-initio references? Can they conserve energy? How well can MLIPs handle vibrational phonon modes and derive thermodynamic properties? How fast are they in practice, and how do they scale with system size and hardware? And last but not least: Can they be used to simulate the adsorption of carbon dioxide molecules in MOFs accurately and efficiently and pave the way for High-Throughput Screening (HTS) of novel materials for carbon capture?

1.3. Contributions

First, we will provide a comprehensive overview of key concepts and techniques used in the field of MD and MLIPs, building a fundamental understanding for readers of all backgrounds (Chapter 2). Next, we will offer a diachronic view of the field, highlighting landmark models and the current State Of The Art (Chapter 3). We will investigate different design choices and architectures used in these models, offering insights into the theoretical concepts key to their success. Then, we will design and implement a software suite to evaluate different MLIPs on a handful of carefully created benchmarks (Chapter 4) that probe different aspects of their performance. Furthermore, we attempt to make *all* ASE MLIPs universally available to LAMMPS, one of the most popular MD software packages, removing the need for custom integrations and paving the way for practical use in real-world workflows. Finally, our benchmark results and case study findings will be presented, evaluated and discussed in detail (Chapter 5).

1.4. Scope and Limitations

This thesis aims to provide a comparative analysis and evaluation of existing models, offering a comprehensive overview of the field and aims to verify if MLIPs can be used for MOF MD simulations. We also aim to offer insights into the underlying theoretical concepts and architectures that power these models, so that readers can better understand how these models work and why they work. We were able to implement an Atomic Simulation Environment (ASE)-LAMMPS bridge to use ASE models in LAMMPS, which allows researchers to use MLIPs in a practical workflow.

1.5. Thesis Structure

This thesis is composed of six chapters, starting from the problem statement going over to background information and then slowly working towards detailed implementation and evaluation.

Chapter 1: Introduction

The introduction provides an overview of the problem statement for readers of every background. It clearly defines the research questions, the methodology being used, outlines the scope and limitations of this work, and gives an overview of the structure of the thesis.

Chapter 2: Fundamentals

The fundamentals chapter gives the reader a glimpse into Quantum Mechanics and Density Functional Theory (DFT), as well as offering necessary depth in various techniques and concepts used by these models, including reverse diffusion processes, symmetries and graph neural networks.

Chapter 3: Related Work

The related work chapter offers a diachronic view of the field, starting from pioneering works all the way to the current State Of The Art. It highlights trends and advancements, and offers a glimpse into the various MLIP architectures.

Chapter 4: Implementation

The implementation chapter details the design and development of the benchmark suite. It discusses the choice of datasets, evaluation metrics, and the integration with the ASE framework. It also describes the MOF-74 case study setup and outlines a few failed attempts at modifying existing models to improve their accuracy and speed.

Chapter 5: Evaluation

Here we offer results of the case study detailed in the implementation, analyzing the performance of different MLIPs in terms of accuracy, speed and practicality. The results are discussed in the context of the research questions and objectives outlined in the introduction.

Chapter 6: Conclusion

Finally, the conclusion chapter summarizes the key findings of the thesis, reflects on the research questions posed in the introduction, and discusses the implications of the results. It also outlines potential directions for future research in the field of MLIPs.

2. Fundamentals

2.1. The Schrödinger Equation

The Schrödinger equation is a linear partial differential equation describing the wave function of any non-relativistic quantum-mechanical system. The wave function of a quantum-mechanical system contains all its observable properties, like its energy, momentum, or position; and fully describes the system's quantum state. The equation, named after Erwin Schrödinger who formulated it in 1925, can be viewed similar to Newton's law of motion, $F = ma$, just for quantum systems. And much like Newton's law, the equation breaks down under near light-speed conditions. The time-dependent Schrödinger equation is given by

$$\hat{H}\Psi(\mathbf{r},t) = \hat{E}\Psi(\mathbf{r},t) = i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t) \tag{2.1}$$

where $\hbar$ is the reduced Planck constant, $\Psi(\mathbf{r},t)$ is the wave function of the system, $\hat{H}$ is the Hamiltonian operator, and $\hat{E}$ is the energy operator. Any time-dependent Schrödinger equation can be transformed into a time-independent equation provided the Hamiltonian operator $\hat{H}$ is time-independent. This can be achieved by decoupling the spatial and temporal parts of the wave function

$$\Psi(\mathbf{r},t) = \psi(\mathbf{r})e^{-iEt/\hbar} \tag{2.2}$$

where $\psi(\mathbf{r})$ is the spatial part of the wave function and E is the energy eigenvalue. Plugging this into equation 2.1 yields the time-independent Schrödinger equation

$$\hat{H}\psi(\mathbf{r}) = E\psi(\mathbf{r}) \tag{2.3}$$

which is an eigenvalue problem where the Hamiltonian operator $\hat{H}$ acts on the spatial wave function $\psi(\mathbf{r})$, yielding the energy eigenvalue E and the corresponding eigenfunction $\psi(\mathbf{r})$. Provided the Hamiltonian operator $\hat{H}$ of a quantum system is known, solving the time-independent Schrödinger equation yields the allowed energy levels and corresponding wave functions of the system. Analytical solutions to the time-independent Schrödinger equation exist only for the simplest systems, like the hydrogen atom (see subsection 2.1.1), the quantum harmonic oscillator, or the particle in a box. More complex systems require numerical methods to approximate the solutions (see subsection 2.1.2).

2.1.1. Wave Equations of the Hydrogen Atom

We look at a single hydrogen atom as a simple example for an analytical solution to the time-independent Schrödinger equation. The problem is described by the Hamiltonian operator,

composed of the kinetic and potential energy operators of the electrons in the Coulomb field of the proton. This problem can be seen as roughly analogous to the classical two-body problem of the Moon orbiting the Earth, where one body is significantly heavier than the other, so the effects of one body on the other can be neglected. In this case, the proton is significantly heavier than the electron, and therefore negligible – which is the basis of the Born-Oppenheimer (BO) approximation. Plugging this Hamiltonian operator into the Schrödinger equation yields

$$\left[-\frac{\hbar^2}{2\mu}\nabla^2 - \frac{\mathrm{e}^2}{4\pi\epsilon_0 r}\right]\psi(\mathbf{r}) = E\psi(\mathbf{r}) \tag{2.4}$$

where $r = |\hat{r}|$[1] is the distance of the electron to the proton, $\mu \approx m_e$ is the reduced mass of the electron, e is the elementary charge, and ϵ_0 is the vacuum permittivity.

A fitting wave function for the hydrogen atom

$$\psi_{n,l,m}(r,\theta,\phi) = R_{n,l}(r)Y_l^m(\theta,\phi) \tag{2.5}$$

derived by separation of angular and radii-dependent terms, where n is the principal quantum number, l is the azimuthal quantum number, and m is the magnetic quantum number, $a_0 = \frac{4\pi\epsilon_0\hbar^2}{\mu \mathrm{e}^2} \approx 0.529\,\text{Å}$ is the Bohr radius, $Y_l^m(\theta,\phi)$ are the spherical harmonics (see 2.52), and $R_{n,l}(r)$ is the radial function (see 2.7), where L_{n-l-1}^{2l+1} are the associated Laguerre polynomials. This separation between the angular and radial parts allows omitting the angular part to obtain a 1D wave function in 2.6, simplifying the problem to one dimension. For intuitive visualization, the probability density of the electron in a hydrogen atom for different quantum numbers n and l is visualized in figure 2.1, where $r = 0$ is at the center, with boundary conditions $u_{nl}(0) = 0$ and $\lim_{r\to\infty} u_{nl}(r) = 0$.

$$u_{nl}(r) = rR_{n,l}(r) \tag{2.6}$$

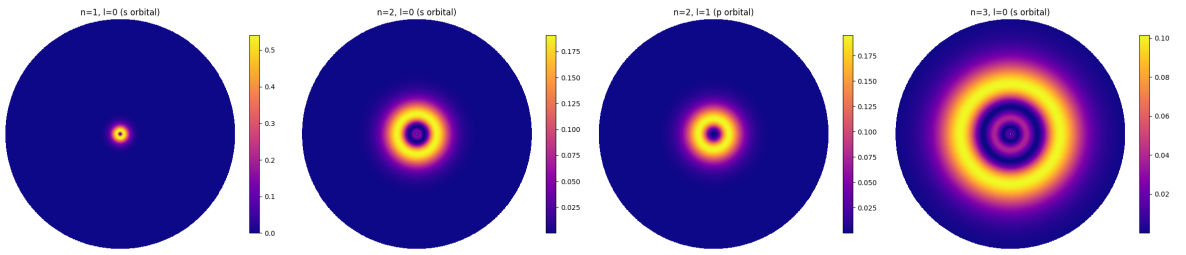

Figure 2.1.: Probability density $|u_{nl}|^2$ plotted for different electron quantum numbers n and l.

The number of possible quantum states of an electron per energy level is given by $2n^2$, and per Pauli's exclusion principle, there can never be two fermions (e.g., electrons) in the same

[1] due to trivial rotational symmetry

quantum state, forcing the wave function to be antisymmetric with respect to the exchange of two electrons [18].

$$R_{n,l}(r) = \sqrt{\left(\frac{2}{na_0}\right)^3 \frac{(n-l-1)!}{2n(n+l)!}} e^{-\frac{r}{na_0}} \left(\frac{2r}{na_0}\right)^l L_{n-l-1}^{2l+1}\left(\frac{2r}{na_0}\right) \tag{2.7}$$

2.1.2. Many-body Problem

Calculations involving more than one electron, let alone multiple atoms, get complicated fairly quickly. Just like in the single electron example above, we start by describing the system with a Hamiltonian operator $\hat{H}$, that includes kinetic and potential energy terms for all particles in the system. However, now that there are multiple electrons and now also nuclei interacting, the Hamiltonian operator becomes significantly more complex with Electron-Electron repulsion, Nucleus-Nucleus repulsion, and Electron-Nucleus attraction terms. Denoting the position, mass, and atomic number of a nucleus I in our system with R_I, M_I, and Z_I respectively, and the position of electron i by r_i with mass m_e, the Hamiltonian is now given by

$$\hat{H} = -\sum_I \frac{\hbar^2}{2M_I}\nabla_{R_I}^2 - \sum_i \frac{\hbar^2}{2m_e}\nabla_{r_i}^2 + \sum_{I<J} \frac{Z_I Z_J e^2}{|R_I - R_J|} + \sum_{i<j} \frac{e^2}{|r_i - r_j|} - \sum_{I,i} \frac{Z_I e^2}{|R_I - r_i|} \tag{2.8}$$

With analytical solutions out of reach, we're left with iterative numerical methods, leaving no other choice than to discretize the wave function. For a system with N electrons and M nuclei, the wave function depends on $3(N+M)$ spatial coordinates, the position of each particle in 3D space. The wave function can either be discretized in "real space" or into coefficients of a chosen basis set (e.g., Laguerre polynomials, B-splines, Gaussian-type orbitals, etc.).

$$\frac{d^2}{d\mathbf{r}^2}\Psi(\mathbf{r},t) \approx \frac{\Psi(\mathbf{r}+h,t) - 2\Psi(\mathbf{r},t) + \Psi(\mathbf{r}-h,t)}{h^2} \quad \text{or} \quad \Psi(\mathbf{r},t) = \sum_i c_i(t)\phi_i(\mathbf{r}) \tag{2.9}$$

Discretizing space is expensive: L sampling points or basis function coefficients per dimension result in $L^{3(N+M)}$ grid points over the entire space. Simulating a single neutral ${}_{92}\mathrm{U}$ atom requires the discretization of a 279-dimensional space – even with a modest $L = 10$ sampling points per dimension, that's 10^{279} grid points, unfathomably large.

For space discretization the Hamiltonian operator $\hat{H}$ is projected to a matrix $H = T + \cdots + V$, where for example the kinetic energy operator

$$\hat{T} = -\sum_i \frac{\hbar^2}{2m_e}\nabla_{r_i}^2 \quad \text{becomes} \quad T = -\sum_i \frac{\hbar^2}{2m_e}\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -2 \end{bmatrix} \tag{2.10}$$

using a second-order finite-difference scheme with grid spacing h, allowing the formulation of the matrix eigenvalue problem

$$\hat{H}\Psi(\mathbf{r},t) = \hat{E}\Psi(\mathbf{r},t) \quad \Longrightarrow \quad HU = EU \quad \Longrightarrow \quad H = UEU^\dagger \tag{2.11}$$

which using matrix diagonalization yields the energy eigenvalues E and eigenvectors U approximating the wave function $\Psi(\mathbf{r}, t)$. An example python script to solve the Hamiltonian eigenvalue problem for a hydrogen atom using finite differences is provided in Listing A.2 in Appendix A. Generally, diagonalizing a dense matrix of size $n \times n$ scales $O(n^3)$, resulting in an overall compute complexity of $O((L^{3(N+M)})^3)$ or $O(L^{9(N+M)})$ compactly written. Due to long-range electrostatic interactions between all particles in the system, the Hamiltonian matrix is often not sparse enough to elicit sparse-matrix diagonalization schemes. In practice, basis functions are employed to reduce the dimensionality of the problem, which allow symmetry exploitation, fewer required grid points, and often better convergence properties [19]. However, this still scales extremely poorly with the number of particles in the system and requires computing the integrals

$$H_{ij} = \langle \phi_i | \hat{H} | \phi_j \rangle = \int \phi_i^*(\mathbf{r}) \hat{H} \phi_j(\mathbf{r}) d\mathbf{r} \tag{2.12}$$

for all basis function pairs ϕ_i and ϕ_j.

Hartree-Fock Method The Hartree-Fock method removes the electron-electron interaction term from the Hamiltonian, and approximates its effects with an average mean field potential, which fails to capture electron correlation effects like van der Waals (vdW) forces. It approximates the many-body wave function $\Psi(\mathbf{r}, t)$ with a single Slater determinant of one-electron wave functions (orbitals) $\psi_i(\mathbf{r}, t)$,

$$\Psi(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N, t) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(\mathbf{r}_1, t) & \psi_2(\mathbf{r}_1, t) & \cdots & \psi_N(\mathbf{r}_1, t) \\ \psi_1(\mathbf{r}_2, t) & \psi_2(\mathbf{r}_2, t) & \cdots & \psi_N(\mathbf{r}_2, t) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{r}_N, t) & \psi_2(\mathbf{r}_N, t) & \cdots & \psi_N(\mathbf{r}_N, t) \end{vmatrix} \tag{2.13}$$

that inherently satisfies the antisymmetry requirement

$$\Psi(\ldots, \mathbf{r}_i, \ldots, \mathbf{r}_j, \ldots, t) = -\Psi(\ldots, \mathbf{r}_j, \ldots, \mathbf{r}_i, \ldots, t) \tag{2.14}$$

of fermions per Pauli's exclusion principle. Hartree-Fock scales as $O(F^4)$ for F basis functions, making it feasible for small systems with up to a few hundred basis functions.

Coupled Cluster Method Coupled Cluster methods build on the Hartree-Fock method, using an *exponential ansatz*

$$|\Psi_{CC}\rangle = e^{\hat{T}} |\Phi_0\rangle \tag{2.15}$$

to include electron correlation effects $\hat{T}$ on top of a reference wave function $|\Phi_0\rangle$ (e.g., the Hartree-Fock wave function). This operator $\hat{T}$ is decomposed into excitation operators $\hat{T}_1, \hat{T}_2, \ldots, \hat{T}_n$ that describe single, double, ..., n-tuple excitations of electrons from occupied to unoccupied orbitals. CCSD(T) includes single, double, and triple excitations and scales as $O(F^{4+3})$ for F basis functions, and is considered the gold standard for solving the Schrödinger equation for small molecules.

However, the method is highly sensitive to the number of basis functions and still scales too poorly for larger systems (>500 atoms).

2.2. Density Functional Theory

DFT is a quantum mechanical method used to derive the ground state of many-electron systems, the lowest possible energy a quantum system can have. Instead of trying to solve the many-body Schrödinger equation directly, DFT aims to simplify the problem using the electron density

$$\rho(\mathbf{r}) = N \int |\Psi(\mathbf{r}, \mathbf{r}_2, \ldots, \mathbf{r}_N)|^2 d\mathbf{r}_2 \ldots d\mathbf{r}_N \tag{2.16}$$

which represents the probability of finding an electron at a specific point in space $\mathbf{r}$. This electron density is normalized to the total number of electrons

$$N = \int \rho(\mathbf{r}) d\mathbf{r} \tag{2.17}$$

in the system. So instead of solving the entire wave function $\Psi(\mathbf{r}, t)$ depending on $3N$ spatial variables of our N electrons, we only need to solve for a general electron density $\rho(\mathbf{r})$ for three spatial dimensions, since electrons are fundamental particles and therefore indistinguishable from each other. This electron density is the central quantity in DFT, used as input for various functionals to derive systemic properties like the total energy, forces, et cetera.

Observables Given an observation operator $\hat{O}$ and a wave function Ψ, one can calculate the expected value $\langle \hat{O} \rangle$ as

$$\langle \hat{O} \rangle(t) = \langle \Psi | \hat{O} | \Psi \rangle(t) = \int \Psi^*(\mathbf{r}, t) \hat{O} \Psi(\mathbf{r}, t) d\mathbf{r} \tag{2.18}$$

which can be derived from the continuous definition of the expected value

$$\mathbb{E}(X) = \int x f(x) dx \tag{2.19}$$

of a random variable X with probability density function $f(x)$. In our case $f(x) = |\Psi(\mathbf{r}, t)|^2 = \Psi^*(\mathbf{r}, t)\Psi(\mathbf{r}, t)$ is the probability density, where the complex conjugate ensures real values.

Functionals Generally, a functional is a function that takes another function as input and produces a scalar output. Given a function $y = f(x) = x^2$, a functional $K[f]$ could be defined as $K[f] = \int_{-\pi}^{\pi} f(x) dx$ resulting in $K[y] = \frac{2\pi^3}{3}$.

If there would exist a functional

$$\Psi[\rho] : \rho(\mathbf{r}) \mapsto \Psi(\mathbf{r}, t) \tag{2.20}$$

that uniquely maps the electron density $\rho(\mathbf{r})$ of a system to its ground-state wave function $\Psi(\mathbf{r}, t)$, Equation 2.18 could be rewritten as

$$\langle \hat{O}[\rho] \rangle(t) = \langle \Psi[\rho] | \hat{O} | \Psi[\rho] \rangle(t) \tag{2.21}$$

to obtain the expected value $\langle \hat{O}[\rho] \rangle(t)$ as a functional only of the electron density $\rho(\mathbf{r})$. And since the energy of a system is such an observable property, it would permit obtaining the expected ground-state energy

$$E_0 = E[\rho_0] = \langle \Psi[\rho_0] | \hat{H} | \Psi[\rho_0] \rangle = \langle \Psi[\rho_0] | \hat{T} + \hat{V} + \hat{U} | \Psi[\rho_0] \rangle \tag{2.22}$$

of the system as a functional of the electron density ρ_0 alone, where $\hat{H} = \hat{T} + \hat{V} + \hat{U}$ is the Hamiltonian operator, composed of the kinetic energy operator $\hat{T}$, the external potential operator $\hat{V}$ (e.g., the Coulomb potential of the nuclei), and the electron-electron interaction operator $\hat{U}$. $T[\rho]$ and $U[\rho]$ are called universal functionals, they are the same in all electron systems, while $V[\rho]$ is system-specific depending on the *nuclei*. For a given density $\rho(\mathbf{r})$, the external potential $V[\rho]$ is given by

$$V[\rho] = \int \rho(\mathbf{r}) V_{ext}(\mathbf{r}) d^3\mathbf{r} \tag{2.23}$$

2.2.1. Hohenberg-Kohn Theorems

Uniqueness theorem Given the ground-state electron density ρ_0, the external potential $V(\mathbf{r})$ is uniquely determined, including all ground-state properties of the system [20]. Although $V_{ext}(\mathbf{r})$ can give rise to multiple ground-states with different densities, two different potentials can never give rise to the same ground-state density [21].

Also, no two different external potentials $V_{ext}(\mathbf{r})$ can give rise to the same ground-state electron density $\rho(\mathbf{r})$ [20], meaning that there exists a one-to-one mapping between the ground-state electron density $\rho(\mathbf{r})$ and the external potential $V_{ext}(\mathbf{r})$ [20, 2], although the exact mapping is unknown.

Variational principle There exists a universal energy functional $E[\rho]$

$$E_0 = \min_{\rho} E[\rho], \quad \text{where } E[\rho] = T[\rho] + U[\rho] + V[\rho] = F[\rho] + \int V_{ext}(\mathbf{r}) \rho(\mathbf{r}) \, d\mathbf{r} \tag{2.24}$$

such that the ground-state energy E_0 is minimized by a correct ground-state electron density ρ_0, where $F[\rho] = T[\rho] + U[\rho]$ [20, 2]. In the spirit of gradient descent, we define a functional derivative $D(\mathbf{r})$ for a density variation $\delta\rho(\mathbf{r})$ and functional $F[\rho]$ as:

$$\lim_{\epsilon \to 0} \frac{F[\rho + \epsilon\delta\rho] - F[\rho]}{\epsilon} = \int D(\mathbf{r}) \delta\rho(\mathbf{r}) d\mathbf{r}, \quad \text{where } D(\mathbf{r}) = \frac{\delta F[\rho]}{\delta \rho(\mathbf{r})} \tag{2.25}$$

It follows that the correct minimizing density ρ_0 must satisfy:

$$\frac{\delta F[\rho]}{\delta \rho(\mathbf{r})} = -V_{ext}(\mathbf{r}) \tag{2.26}$$

as described by Giuliani and Vignale [21]. This correct minimizing density must be found iteratively with a self-consistent field (SCF) procedure, constraint to orthonormality via $\int \psi_i^*(\mathbf{r})\psi_j(\mathbf{r})d\mathbf{r} = \delta_{ij}$ for the single-particle wave functions ψ_i and ψ_j where δ_{ij} is the Kronecker delta.

2.2.2. Kohn-Sham Reformulation

The problem is that neither $T[\rho]$ nor $U[\rho]$ are known explicitly. To solve this, Kohn and Sham [22] proposed reformulating the problem by introducing a fictitious system of non-interacting electrons with Φ_i Kohn-Sham orbitals that has the same ground-state electron density $\rho(\mathbf{r})$ as the real, interacting system under the constraint that $\rho(\mathbf{r}) = \sum_i^N |\Phi_i(\mathbf{r})|^2$. They decompose the universal functional $F[\rho]$ into three parts:

$$F[\rho] = T_s[\rho] + E_H[\rho] + E_{xc}[\rho] \tag{2.27}$$

where $T_s[\rho]$ is the kinetic energy of the non-interacting system, $E_H[\rho]$ is the classical electrostatic (Hartree) energy, and $E_{xc}[\rho]$ – the Exchange-Correlation (XC) functional – describes *everything else*. This gives rise to the Kohn-Sham equations

$$\left[-\frac{\hbar^2}{2m_e}\nabla^2 + V_{\text{eff}}(\mathbf{r})\right]\Phi_i(\mathbf{r}) = \epsilon_i\Phi_i(\mathbf{r}) \tag{2.28}$$

for the Kohn-Sham orbitals $\Phi_i(\mathbf{r})$, where ϵ_i are the Lagrange multipliers ensuring orthonormality of said orbitals, and $V_{\text{eff}}(\mathbf{r})$ is the effective potential that encodes all interactions in the system:

$$V_{\text{eff}}(\mathbf{r}) = \frac{\delta}{\delta\rho(\mathbf{r})}\left(E_H[\rho] + E_{xc}[\rho] + \int V_{ext}(\mathbf{r}')\rho(\mathbf{r}')d\mathbf{r}'\right) = V_H(\mathbf{r}) + V_{xc}(\mathbf{r}) + V_{ext}(\mathbf{r}) \tag{2.29}$$

We know that the Hartree potential

$$V_H(\mathbf{r}) = \int \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|}d\mathbf{r}' \tag{2.30}$$

and the external potential

$$V_{ext}(\mathbf{r}) = -\sum_I \frac{Z_I e^2}{|\mathbf{r} - R_I|} \tag{2.31}$$

are given by the Coulomb potential of the nuclei. But the XC potential

$$V_{xc}(\mathbf{r}) = \frac{\delta E_{xc}[\rho]}{\delta\rho(r)} \tag{2.32}$$

is unknown and must be approximated either with Local Density Approximation (LDA) or with Generalized Gradient Approximation (GGA) methods

$$E_{xc}^{LDA}[\rho] = \int \rho(r)\epsilon_{xc}(\rho(r))dr \tag{2.33}$$

$$E_{xc}^{GGA}[\rho] = \int f(\rho(r), \nabla\rho(r))dr \tag{2.34}$$

where $\epsilon_{xc}(\rho(\mathbf{r}))$ is the XC energy density of a homogeneous electron gas with density $\rho(\mathbf{r})$, and $f(\rho(\mathbf{r}), \nabla\rho(\mathbf{r}))$ is a function of the local electron density and its gradient [23]. LDA overbinds electrons, while GGAs like the Perdew–Burke–Ernzerhof exchange–correlation functional (PBE) tend to underbind them [21, 11, 24], which led to the use of hybrid GGA methods like B3LYP [25]. But these solutions are far from perfect: Both LDA and GGA operate with a local view of the electron density, ignoring medium range interactions like van der Waals forces [11] or electrostatic interactions in charged systems [26]. In practice this means that standalone DFT XC functions can not accurately model basic H_2O clusters or protein-ligand interactions without additional dispersion corrections like DFT-D3 [26, 27].

2.3. Graph Neural Networks

2.3.1. Graphs

Graphs are highly flexible, easy to extend, and applicable to a wide range of real-world applications. Transportation systems like road and rail networks, atomic systems, and social networks can all be modeled by graphs.

We define a graph G as a tuple $G = (V, E)$. V is the set of nodes (vertices) and $E = V \times V$ is the set of edges, $n_e = |E|$ is the number of edges, and $n_v = |V|$ is the number of nodes. We denote the adjacency matrix of G as $A \in \mathbb{R}^{n_e \times n_e}$ where $A_{ij} = 1$ if $(v_i, v_j) \in E$, otherwise $A_{ij} = 0$. Nodes and edges have feature vectors that correspond to certain features. For example, when modeling roads as edges between junction nodes, each road edge could have a length feature, and each junction a delay time feature. On the other hand, to model atomic systems, for example, each atom is represented as a node, and we connect nodes closer than 6 Å (Six Angstroms) with an edge.
The core assumption made by most graph-based methods is that nodes closer together (as defined by the minimum hop count) are more informative to each other than nodes that are further away. This heavy inductive bias should be taken into consideration before modeling any system with a graph.

2.3.2. Message Passing

The most adopted form of Graph Neural Networks (GNNs) is message passing. Here, the model iteratively updates hidden node representations by letting the nodes exchange information with their immediate neighbors. This happens multiple times in so-called message-passing rounds, resulting in continuously wider message propagation. Interestingly, Message Passing Neural Networks (MPNNs) were specifically designed for MLIPs, as described in [28],

Each round k consists of three steps: The first is message collection, where the hidden node vectors v_i^k are used to compute a message between the two using a Multi-Layer Perceptron

(MLP) function ϕ^k as seen in equation 2.35.

$$m_{ij}^k = \phi^k(v_i^k, v_j^k) \tag{2.35}$$

The second step involves taking all incoming messages of a node, aggregating them with an operator $\bigoplus$ as seen in equation 2.36.

$$M_i^k = \bigoplus_{j \in \mathcal{N}(i)} m_{ji}^k \tag{2.36}$$

It is important to note that the aggregation operator must be permutation-invariant, meaning that the order of the neighbors must not matter since the neighbors are an unordered set. Common choices for the aggregation operator are summation, mean, or max pooling. A common issue is the variance-drift problem as outlined by Glorot and Bengio [29], where the variance of the aggregated messages increases with the number of neighbors. If X and Y are iid.[2] random variables, then $Var(X+Y) = Var(X) + Var(Y)$. This means that nodes with many neighbors will have a higher variance in their aggregated messages than nodes with few neighbors, and therefore nodes with more neighbors have a greater overall influence. To normalize the variance under addition, we multiply the sum by $(\sum_{X \in \ominus} Var(X))^{-\frac{1}{2}}$.

The third and last step, the update step, uses the output of a MLP function φ^k to update the new hidden node value as seen in equation 2.37.

$$v_i^{k+1} = \varphi^k(v_i^k, M_i^k) \tag{2.37}$$

Graph Attention Networks (GATs) aim to exploit the fact that some nodes may be more influential to neighbors than other nodes. For example, some bonds may be more or less important in the application of atom graphs, or some connections in social networks may be stronger than others. Message passing treats all neighbors equally, which can lead to information dilution from less important neighbors. Before message-passing, we use a standard MLP to compute an attention or affinity score

$$a_{ij}^k = \text{MLIP}_{\text{attn}}^k(v_i^k, v_j^k) \tag{2.38}$$

$$\tag{2.39}$$

between two nodes i and j. These scores need to be then normalized with either sigmoid for a soft selection of neighbors or softmax for a hard one. Finally, the update step is modified to include the attention scores, as seen in equation 2.40.

$$M_i^k = \bigoplus_{j \in \mathcal{N}(i)} \sigma(\alpha_{ji}^k) m_{ji}^k \tag{2.40}$$

$$\tag{2.41}$$

[2] independent and identically distributed

2.4. On Symmetries

Symmetries are a fundamental property of nature, and their existence is a cornerstone of modern physics. Not only do they directly postulate fundamental conservation laws of our universe [30], they also allow us to improve our machine learning models.

2.4.1. Equivariance and Invariance

A function $f : V \to W$ between sets with group actions $T_g : V \to V$ and $S_g : W \to W$ for a group G is called invariant to a group element $g \in G$ if applying the group action to the input does not change the output. This can be expressed mathematically, as seen in Equation 2.42. For example, the distance between two points is invariant to the rotation and translation of the points, specifically to the SE(3) group, which includes the SO(3) group of rotations in 3D space.

$$f(T_g x) = f(x) \tag{2.42}$$

A function $f : V \to W$ is called equivariant to a group element $g \in G$ if applying the group action to the input is equivalent to applying a (potentially different) group action to the output. This can be expressed mathematically, as seen in Equation 2.43. For example, the vector between two points is equivariant to the SE(3) group.

$$f(T_g x) = S_g f(x) \tag{2.43}$$

Irreducible Representations In representation theory, Irreducible Representations (Irreps) describe how our data transforms under group actions [31]. Specifically, we are interested in the Irreps of SO(3), which describe rotations in 3D space. The Irreps of the SO(3) group are described by vector spaces $V^{(l)}$ labeled by angular momentum quantum numbers $l \in \mathbb{N}_0$ and have dimension $2l + 1$. They describe how our data transforms under rotations in SO(3).

Scalars ($a \in V^{(0)}$) are rotation invariant quantities like energy, mass or distance. They won't change when we rotate or translate our coordinate system.

Vectors ($v \in V^{(1)}$) have both a magnitude and a direction. They often represent velocities or forces. Rotating the coordinate system while affecting the vector components, does not change the physical quantity the vector represents. They have three scalar components, typically denoted as (x, y, z).

Quadrupole ($T \in V^{(2)}$) can be represented as symmetric, traceless 3×3 matrices. They can represent quantities like the moment of inertia or the stress tensor in a material and have five independent components.

Tensors with rank $l \geq 2$ are called *higher-order tensors* and transform in more complex ways under rotations.

Tensor Products Now that we have defined different tensor types, we need to understand how to combine them while preserving their transformation properties, especially when we face

tensors of different types. We define the tensor product $\otimes$ as a bilinear operation that combines two tensors of types l_1 and l_2 into a new tensor of type l, preserving equivariance to rotations in SO(3). The result of a tensor product operation of two Irreps $V^{(l_1)}$ and $V^{(l_2)}$ is not irreducible and needs to be decomposed into a sum of Irreps $V^{(l)}$, with types l from $|l_1 - l_2|$ to $l_1 + l_2$, as illustrated for the tensor product of two vectors in Equation 2.44.

$$V^{(l_1)} \otimes V^{(l_2)} \approxeq \bigoplus_{l=|l_1-l_2|}^{l_1+l_2} V^{(l)} \tag{2.44}$$

Given *two scalars* $a, b \in V^{(0)}$, their equivariance-preserving tensor product is simply multiplication $a \otimes b = ab \in V^{(0)}$, since according to Equation 2.45 the tensor product of two scalars is still a scalar.

$$V^{(0)} \otimes V^{(0)} \approxeq V^{(0)} \tag{2.45}$$

The same applies for *scalars* $a \in V^{(0)}$ with *vectors* $v \in V^{(1)}$, where the tensor product is scalar multiplication $a \otimes v = av \in V^{(1)}$, described in Equation 2.46. Scaling a vector does not change its direction, so the transformation properties remain the same.

$$V^{(0)} \otimes V^{(1)} \approxeq V^{(1)} \tag{2.46}$$

We demonstrate that the tensor product of a scalar and a vector is indeed equivariant to rotation matrices $R \in SO(3)$ in Equation 2.47. The rotation matrix R is *orthogonal* ($R^T R = I$) and has a determinant of 1 ($\det(R) = 1$), hence *special*. We define a Wigner D[3] matrix $D^{(l)}(R)$ describing how a tensor of type l transforms under the rotation R and define a scalar $a \in \mathbb{R}$ and a vector $v \in \mathbb{R}^3$.

$$a \otimes Rv = D^{(1)}(R)(a \otimes v) \tag{2.47}$$

Unsurprisingly in this simple example, the Wigner D-matrix for scalars is simply $D^{(0)}(R) = 1$ and for vectors $D^{(1)}(R) = R$, so $D_R = R$. Additional python code for this concrete example can be found in Listing A.1.

But what about the tensor product of *two vectors* $v_1, v_2 \in V^{(1)}$? The tensor product $v_1 \otimes v_2$ is just the outer product of the two vectors. This outer product results in a reducible cartesian tensor $M \in \mathbb{R}^{3\times3}$ which according to the decomposition structure 2.44 can be split up into three Irreps: a scalar (the dot product, trace), a vector (the cross product, antisymmetric part), and a quadrupole (the symmetric traceless part).

$$V^{(1)} \otimes V^{(1)} \approxeq V^{(0)} \oplus V^{(1)} \oplus V^{(2)} \tag{2.48}$$

[3] "Darstellung", German for representation [32]

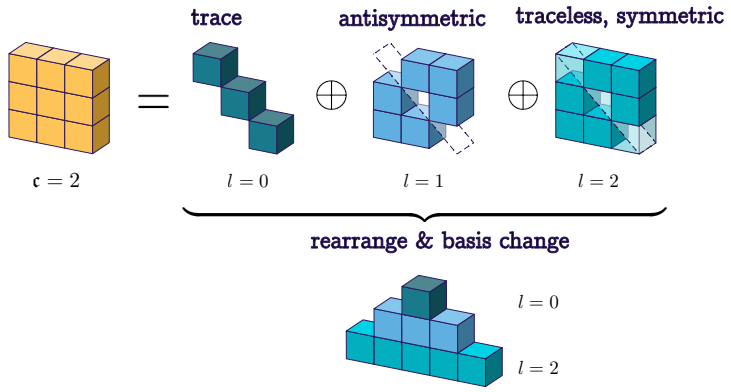

Figure 2.2.: Decomposition of the tensor product of two vectors into a scalar, vector, and quadrupole [33].

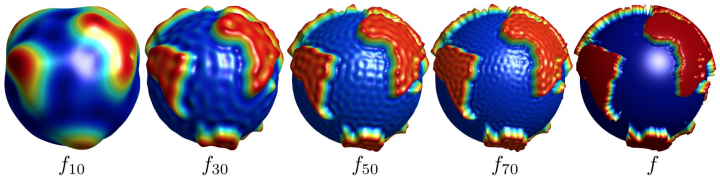

Figure 2.3.: Spherical harmonics are the equivalent of Fourier series on the surface of a sphere [34].

Spherical Harmonics To encode a signal on the surface of a sphere, spherical harmonics can be used. They approximate – with increasing accuracy – any square-integrable function on the sphere by decomposing it into basis functions of increasing order l: $Y^l : \mathbb{S}^2 \to D^l = \left(Y^l_{-l}(\theta,\phi), Y^l_{-l+1}(\theta,\phi), \ldots, Y^l_{l-1}(\theta,\phi), Y^l_l(\theta,\phi)\right)$. They are a family of $2l+1$ polynomial basis functions where D^l is the set of l-Irreps. They are basis functions for Irreps of the SO(3) group and are closely related to the Fourier Transform, where a signal is decomposed into complex coefficients of sine and cosine functions [31].

We can project to a sphere using spherical coordinates $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi)$ as seen in Equation 2.49, where v^l are the complex coefficients of the Spherical Harmonics $Y^l(\theta, \phi)$.

$$f(\theta,\phi) = \sum_{l=0}^{\infty} v^l Y^l(\theta,\phi) \tag{2.49}$$

The order l describes not only the frequency of oscillations but also the transformation properties under rotation.

Their basis functions are given in Equation 2.52 and the first real-valued Spherical Harmonics are visualized in Figure 2.4, where $l \in \mathbb{N}_0$ describes the angular quantum number and $m \in \mathbb{Z}$ with $-l \leq m \leq l$ describes the number of oscillations around the z-axis. They form a complete orthonormal set of functions on the sphere and integrate to one over it, as seen in Equation 2.50 and Equation 2.51.

$$\int_0^{2\pi}\int_0^{\pi} Y_m^l(\theta,\phi)Y_{m'}^{l'*}(\theta,\phi)\sin(\theta)d\theta d\phi = \delta_{ll'}\delta_{mm'} \tag{2.50}$$

$$\int_0^{2\pi}\int_0^{\pi} |Y_m^l(\theta,\phi)|^2\sin(\theta)d\theta d\phi = 1 \tag{2.51}$$

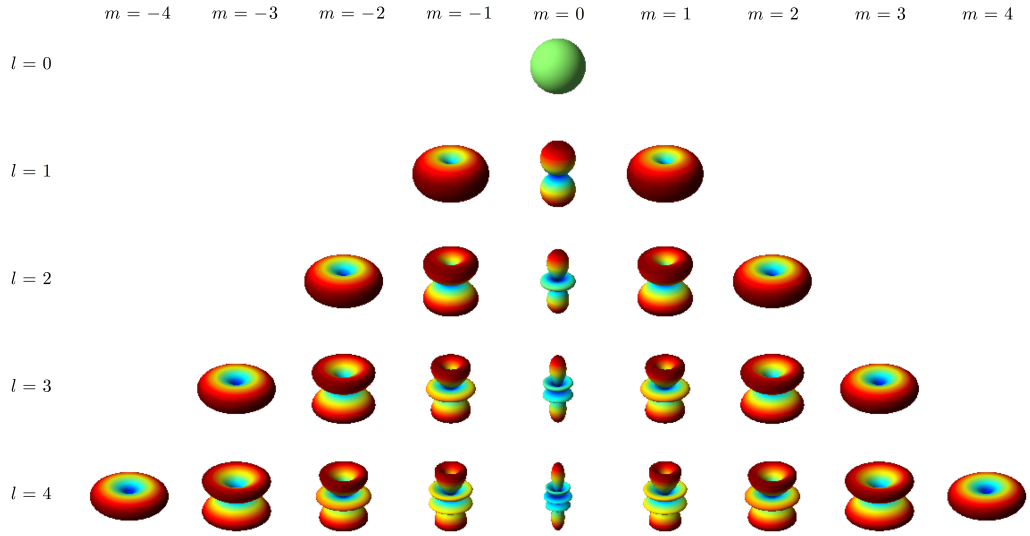

Figure 2.4.: Spherical harmonics Y_l^m for different quantum numbers l and m [35].

$$\hat{Y}_m^l(\theta,\phi) = (-1)^m\sqrt{\frac{(2l+1)}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos(\theta))e^{im\phi} \tag{2.52}$$

They are naturally equivariant to the SO(3) group with respect to Irreps of the same order, as a rotation of a sphere results in an equivalent rotation of the signal on the sphere. Given a rotation matrix $R \in SO(3)$, a Spherical Harmonics family Y^l of degree l transforms according to the Wigner D-matrix $D^{(l)}(R)$ as seen in Equation 2.53.

$$Y^l(R\vec{x}) = D^{(l)}(R)Y^l(\vec{x}) \tag{2.53}$$

It's not strictly necessary to use Spherical Harmonics to achieve equivariance, but they have well-defined transformation rules under rotations and allow us to merge information from different directions in a principled way.

2.4.2. Symmetries in Machine Learning

Standard MLPs are notoriously sensitive to rotations and translations of input data. They simply "don't know" what it means to be a vector and what it means for one to be morphed, rotated or translated. The easiest way to teach a model these or more general symmetries is by letting it discover them entirely by itself. The theory is that *more data* with an *easier*

architecture beats *less data* with a *more complex architecture*. And this concept turns out to be true more often than not, since complex architectures can be error-prone, incomplete and expensive to train. For example training a neural network to shoot a ball at a target at fixed length: while in essence it would only need to aim slightly higher than the target direction, we would need to train it for all vertical orientations, and then we can only hope that it can effectively interpolate between the datapoints. At the other end of the spectrum one finds equivariance built directly into the architecture of the model. This means that the input vector can be rotated arbitrarily in the horizontal direction, and the output vector rotates accordingly up to a tiny numerical error. Now just one single training example is enough to teach the model for all possible rotations, drastically reducing the amount of data needed to train it. Let's make a back of the napkin calculation of how much less training data needs to be seen when compared to a model with 3D equivariance: A datapoint x is rotated along three euler angles α, β, γ with a resolution of 10 steps each, resulting in $10^3 = 1000$ different orientations to represent the same underlying data. So assuming the non-equivariant model can perfectly interpolate between seen data, it would take it 1000 times more data to learn the same relationship as an equivariant model. But fully equivariant models often have the downside that they are complex, hard to understand and implement, and they take significantly longer to train. As a result, models are often built to be equivariant to only a subset of the full symmetry group. These *partial equivariant* models are often the result of a compromise between data availability, model complexity, and training time. One prime example are Convolutional Neural Networks (CNNs) for image classification tasks, that are effectively equivariant to 2D translations of the image, but not to rotations, scaling and mirroring.

2.5. Reverse Diffusion Processes

Generative modeling is the task of learning to generate new data points from an unknown data distribution, given only a handful of unlabelled data points. There are two approaches to this task: learning a probabilistic model in one step or learning to approximate a vector field that iteratively refines a random sample to increase the likelihood of a given sample. Examples of the first way to generate new samples from an approximated solution in a one-shot manner are Variational Autoencoders (VAEs) introduced by Kingma and Welling [36], Generative Adversarial Networks (GANs) from Goodfellow et al. [37], and language models like GPT [38, 39, 40], Claude, and others. Two examples of the second approach are both reverse diffusion processes, Denoising Diffusion Probabilistic Models (DDPM) and Score Matching with Langevin dynamics (SMLD), although they are almost identical. In the context of force-field prediction, recent work showed that these two general approaches are indeed *equivalent* under the condition that the predicted structures are relaxed. The key idea behind diffusion models are *Langevin dynamics*, which also finds use in MD simulations, specifically Langevin Monte Carlo (LMC). Any iid. probability density function p can be simulated by taking a uniformly distributed sample $x_0 \sim U(-1, 1)$ and iteratively updating it using a perturbation term W

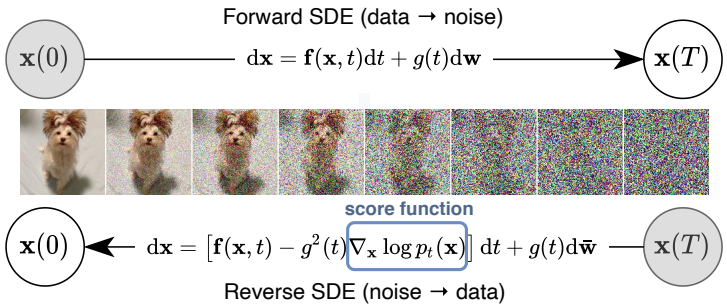

Figure 2.5.: Solving a reverse-time SDE yields a score-based generative model [41].

sampled using the *Wiener process* (2.54) and an *Itô diffusion* update rule (2.55) using a *score function* $\nabla \log p(x)$ with a fixed timestep ξ. This process *converges to the desired distribution* p in the limit $\lim_{t\to\infty} x_t \sim p(x)$.

$$W_{t+\Delta t} - W_t \sim \mathcal{N}(0, I) \tag{2.54}$$

$$x_{t+1} = x_t + \xi \nabla \log p(x) + \sqrt{2ß} W_t \tag{2.55}$$

So the problem of approximating a probability density function p is now reduced to approximating its score function $\nabla \log p(x)$. A fixed Markov chain (as seen in figure 2.5) of length T is first defined, in which a given data point is gradually corrupted, making it less likely under p.

A neural network is then trained to reverse this corruption, learning to increase the likelihood of a data point under p at step t. This results in the network learning a (hopefully low-dimensional manifold) vector field where each input is mapped to a vector that points towards higher likelihood regions of the data distribution, very much like a gradient ascent step, as seen in figure 2.6.

Generally, the forward process can be described by the Stochastic Differential Equation (SDE) 2.56, where $f(x, t)$ is the drift coefficient, $g(t)$ is the diffusion coefficient, and W_t is a standard Wiener process (2.54) at time t.

$$dx = f(x, t)dt + g(t)dW_t \tag{2.56}$$

The reverse function is governed by the reverse-time SDE 2.57, where we approximate the score function $\nabla \log p_t(x)$ with a neural network.

$$dx = [f(x, t) - g(t)^2 \nabla \log p_t(x)]dt + g(t)d\bar{W}_t \tag{2.57}$$

For DDPM, $f(x, t) = -\frac{1}{2}\beta(t)x$ and $g(t) = \sqrt{\beta(t)}$, where $\beta(t)$ is a pre-calculated noise schedule.

To train a neural network, either the KL divergence 2.58 between the true distribution p at time t and the predicted distribution q at time t as gaussians with mean μ and variance σ, or

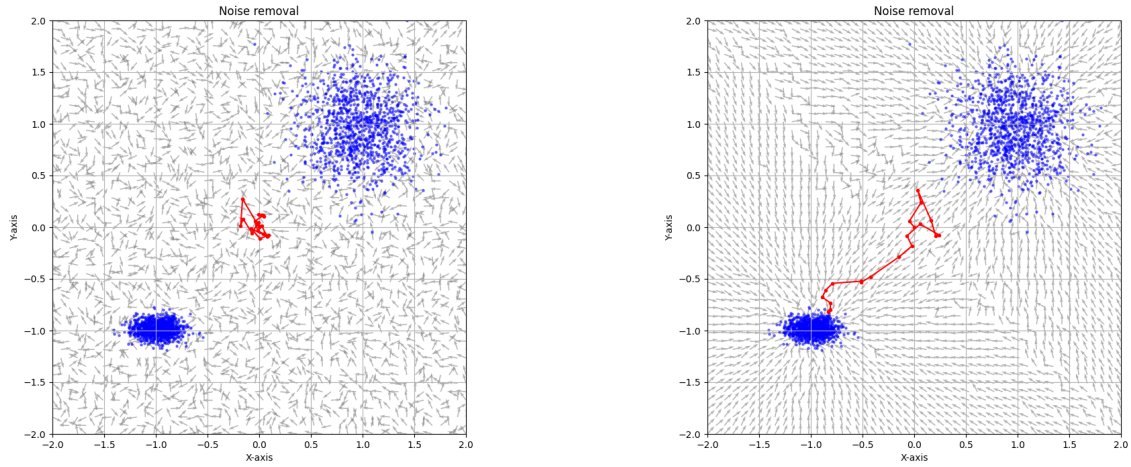

Figure 2.6.: Diffusion vector space at epoch 0 (left) and 10 (right). Blue scatter points indicate *unlabelled data*, gray arrows illustrate a denoising network's average normalized *noise removal direction*, and red is a *sample trajectory* starting from (0, 0).

the direct score-matching loss 2.59 to predict the noise added to the data point at time t can be used [41].

$$D_{KL}(\mathcal{N}(\mu_\theta, \sigma_\theta)||\mathcal{N}(\mu_q, \sigma_q)) = \log \frac{\sigma_p}{\sigma_q} + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2} \tag{2.58}$$

$$\mathbb{E}_{x\sim p}\left[||s_\theta(x) - \nabla \log p(x)||_2^2\right] \tag{2.59}$$

Interestingly, predicting the noise directly via score matching loss is indeed equivalent to minimizing the KL divergence in the Evidence Lower Bound (ELBO) for a fixed variance schedule. For simplicity, the score matching loss is often preferred over the more complex ELBO loss.

3. Related work

3.1. Landmark MLIPs

Even though DFT methods heavily improve speed and scaling properties for Ab-inhibito Molecular Dynamics (AIMD) compared to State Of The Art (SOTA) Coupled Cluster (CC) methods, they're still too slow for large scale systems, usually the more interesting ones. One key problem is the fact that DFT approximates the entire wave function, in essence solving the entire system completely, every single time step, which is very costly and as we have seen still scales poorly with system size. But for MD we only care about the forces acting on the atoms, which are derived from the wave function, not the wave function itself. DFT is therefore doing a lot of unnecessary work, but so far no direct method can predict forces as accurately as the wave function can derive them, which is why it's still being used. So if there was a way to compute the forces from atomic positions directly, we could skip the highly expensive wave function calculation entirely. MLIPs are trained on a dataset of DFT calculations to learn this mapping as accurately as possible, providing the best of the two worlds: the accuracy of DFT with the speed of cheap empirical potentials.

Explicit MLIPs For N atoms, their position $R \in \mathbb{R}^{N\times 3}$ and species $Z \in \mathbb{Z}^N$ are taken as input, and the mapping to energy $E \in \mathbb{R}$, forces $F \in \mathbb{R}^{N\times 3}$ and stress $\sigma \in \mathbb{R}^{N\times 6}$ is learned directly. This is considered the straightforward and intuitive approach, but no energy conservation guarantees are offered by it.

$$F_i, E, \sigma_{\alpha\beta} = f_\theta(r_1, r_2, \ldots, r_n) \tag{3.1}$$

They are also referred to as direct-force predictors.

Implicit MLIPs For implicit MLIPs, atomic positions $R \in \mathbb{R}^{N\times 3}$ and species $Z \in \mathbb{Z}^N$ are taken as input, and a mapping is learned to the potential energy $E \in \mathbb{R}$ of the system alone:

$$E = f_\theta(r_1, r_2, \ldots, r_n) \tag{3.2}$$

The forces F_i, the stress σ, and the strain tensors $\epsilon_{\alpha\beta}$ are then computed by taking the negative gradient of the network with respect to the atomic positions r_i and the strain tensors $\epsilon_{\alpha\beta}$ respectively.

$$\mathbf{F}_i = -\nabla_{\mathbf{r}_i} E \quad \text{and} \quad \sigma_{\alpha\beta} = \frac{1}{V}\frac{\partial E}{\partial \epsilon_{\alpha\beta}} \tag{3.3}$$

They are also called conservative MLIPs. Chmiela, Tkatchenko, Sauceda, et al. [42] were the first to describe this idea in their Gradient-domain Machine Learning (GDML) approach. The big problem with this approach is the second backpropagation step, which is very costly, especially for training where you now need a gradient of a gradient.

3.1.1. SchNet (2018)

SchNet [13] is one of the pioneering GNN-based MLIPs. Introduced by Schütt et al., it was the first popular implicit MLIP model, a concept that all major models still more or less employ today. Its release initiated a paradigm shift in the field, resulting in the gradual replacement of handcrafted methods like Gaussian Approximation Potentials (GAP) [43] by end-to-end fully learnable architectures.

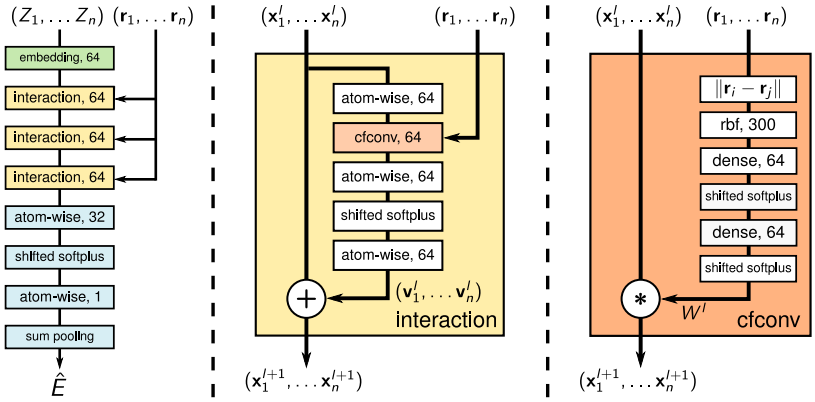

Figure 3.1.: SchNet architecture overview. CFConvs make the surface smooth; the use of distances makes the model invariant to rotations [13].

Invariance to E(3) SchNet is invariant to rotations, translations, and atomic indexing, which made it a significant step forward compared to previous models. Translation equivariance and atomic indexing symmetry are the easiest to achieve, as they often come naturally when working with graphs; which is why, going forward, we will only focus on rotational equivariance.

They achieve rotational equivariance by using the distance $d_{ij} = ||r_i - r_j||_2$ between two atoms i and j as input to the model, an operation *invariant* to rotations. These distances are then transformed into coefficients of Radial Basis Functions (RBF),

$$e_k(d_{ij}) = \exp(-\gamma(d_{ij} - \mu_k)^2) \quad \text{with} \quad k \in [1, K] \tag{3.4}$$

a fix for the high variance of the distance variable slightly resembling Fourier Features [44], which are then used further downstream for the update step.

The update step is a simple convolution with ResNet [45] skip-connections over neighboring

atoms,

$$v_i^{(k+1)} = v_i^{(k)} + (v^{(k)} * W^k) = v_i^{(k)} + \sum_{j \neq i} W^k(d_{ij}) v_j^{(k)} \tag{3.5}$$

weighted by a learned function $W(d_{ij})$ of the inter-atomic distances. But newer models no longer take distances alone as model input, due to the fact that they underrepresent atomic systems especially when the model has fewer interaction layers. Only measuring distance can lead to having different atomic systems that result in the same prediction, even though they exert different forces and energies.

Continuous Filter Convolutions A major challenge for applying convolutions in MLIPs is the irregular and sparse arrangement of neighboring atoms. This contrasts with CNNs for images, where pixels form a regular grid and the discretization is fine enough to make reasonable predictions. For MLIPs, a discrete convolution kernel analogous to those in CNNs results in a poorly discretized, discontinuous Potential Energy Surface (PES). SchNet addresses this problem by introducing Continuous-Filter Convolution (CFConv) layers. Instead of using a discrete kernel matrix, this method learns a *continuous kernel function*. First, the distance between two atoms d_{ij} is expanded using a set of RBF as shown in 3.4. Then, the resulting vector is processed by a small MLP to generate a filter weight $W(d_{ij})$:

$$W(d_{ij}) = \text{MLP}(e_1(d_{ij}), e_2(d_{ij}), \ldots, e_K(d_{ij})) \tag{3.6}$$

This technique of generating filter weights from interatomic distances results in a smooth PES, as illustrated in Figure 3.2.

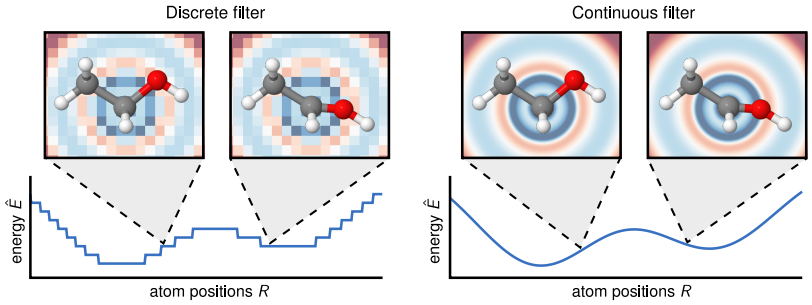

Figure 3.2.: Discrete convolution kernels result in a low-resolution PES, while continuous convolution filters lead to a smooth one [13].

3.1.2. PaiNN (2021)

The Polarizable atom interaction Neural Network (PaiNN) [14] model from Schütt et al. is a direct evolution of SchNet and builds on ideas from Tensor Field Networks (2018) [46]. It employs a message-passing approach, which is now the standard for MLIPs. CFConv layers are

used for message and node updates, with the RBFs from DimeNet (2020) [47] multiplied by an additional cosine cutoff function to ensure smoothness at the cutoff distance.

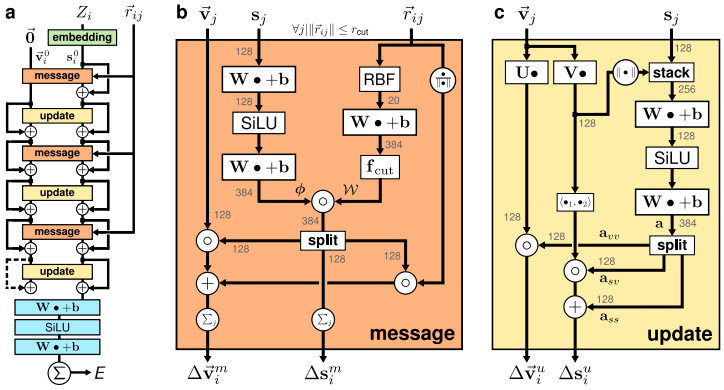

Figure 3.3.: PaiNN architecture overview. It is combining different equivariant operations and their results, allowing the network to mix and match accordingly [14].

From Invariance to Equivariance Even though they still employ invariant distance-based CFConvs from Schutt et al. [13], they multiply their output with vector-based features. They constrain the update and message update functions to a limited set of equivariant operations, expanding SchNets idea of rotational invariance to rotational equivariance. This set includes potentially non-linear functions on scalars, vector scaling, vector products, scalar products, and linear combinations of equivariant vectors. These operations preserve directional information [14] and thus make both functions equivariant to rotations. As demonstrated in Figure 3.4, equivariance is significantly more data efficient than invariance. This is especially important for our purposes where unique training samples are costly to obtain.

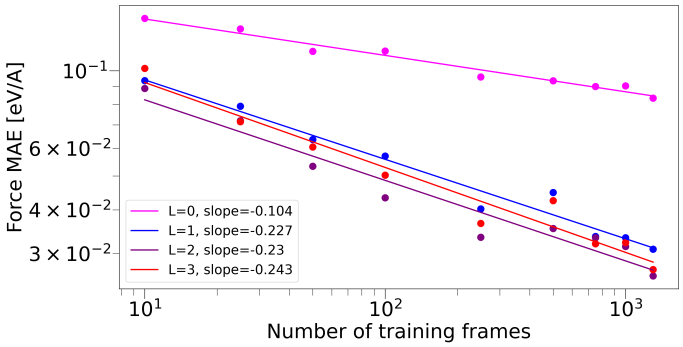

Figure 3.4.: Different force MAEs. $L = 0$ means the model is rotation-invariant, $L > 0$ describes different granularities of rotational equivariance [48].

3.1.3. NequIP (2022)

Neural Equivariant Interatomic Potentials (NequIP) [48] from Batzner et al. were the first to fully embrace the work from Tensor Field Networks (2018) [46] efficiently. Instead of using just a few equivariant operations, they employ full Special Orthogonal Group in 3D (SO(3)) tensor products using the E(3) equivariant neural networks (e3nn) [49, 46, 50, 51, 52] python library. Nvidia has since released *cuEquivariance* [53], essentially a GPU-accelerated version of *e3nn*.

The key idea is to project the directional information onto spherical harmonics $Y_m^{(l)}(\hat{r}_{ij})$ (see paragraph 2.4.1), which are rotationally equivariant. These spherical harmonics are then multiplied with a learnable radial function $R(r_{ij})$

$$S_m^{(l)}(\hat{r}_{ij}) = R(r_{ij})Y_m^{(l)}(\hat{r}_{ij})$$

to form a convolutional filter similar to SchNet's CFConv layers.

In their experiments they found that using $l = 0$ for the spherical harmonics (i.e. rotational invariance) yielded significantly worse results than using $l = 1$ or $l = 2$, but increasing l further didn't improve results much more, as can be seen in Figure 3.4.

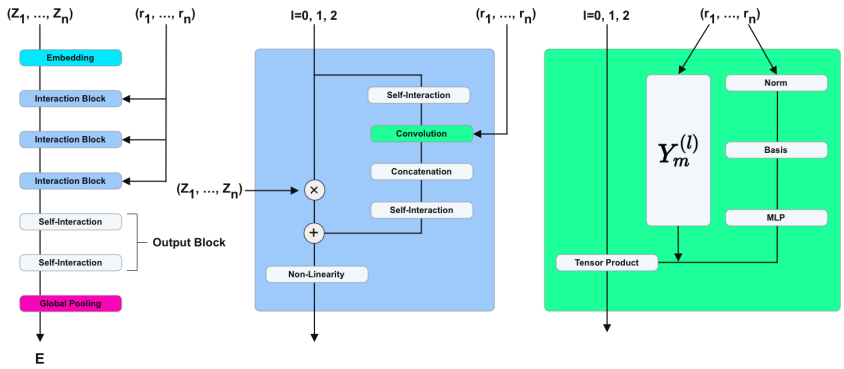

Figure 3.5.: NequIP architecture overview. The use of spherical harmonics allows the model to be fully equivariant [48].

The Clebsch-Gordan tensor product of two SO(3) representations is again a representation of SO(3), which allows them to mix and match features of different orders l freely - a significant improvement over the use of only a limited set of equivariant operations.

3.1.4. Orb v2 (2024)

The Orb models from Neumann, Gin, Rhodes, et al. [54] were the first to demonstrate competitive accuracy without an equivariant or invariant architecture. They use a denoising diffusion objective similar to DDPMs for pretraining on equilibrium structures, and then fine-tune on forces and energies as described by Zaidi, Schaarschmidt, Martens, et al. [55]. This makes the model not only more data efficient, but also results in significantly faster inference

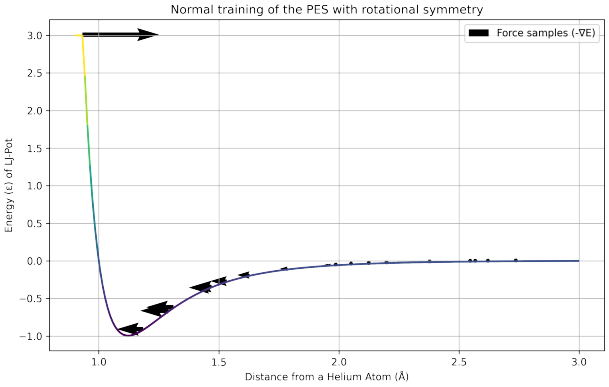

Figure 3.6.: Using symmetries simplifies the PES learning task significantly. Simplified to only depend on the atomic distances, depiction is a scetch.

times, as it doesn't use a computationally expensive equivariant architecture. Harcombe and Duignan [56] reported that their diffusion models needed 50% less training data to reach the same accuracy as their non-diffusion models that also don't respect symmetries in their experiments. Diffusion pretrained models offer different advantages and disadvantages compared to traditional Neural Network Potential (NNP)s over each other, one is not strictly better than the other. Even though symmetry-agnostic diffusion models are more data efficient and are arguably more accurate for relaxation tasks, normal NNPs typically yield better force Root Mean Square Error (RMSE) values. This is likely due to the fact that diffusion models predict a direction with *unit* magnitude, while NNPs are also trained on the force magnitude, as illustrated in Figure 3.7. As such, the Force RMSE is no longer a good metric for diffusion models, which is why it is not used in the Matbench Discovery Benchmark [57]. Rhodes, Vandenhaute, Šimkus, et al. [58] argue that as long as the model can accurately relax structures, the exact forces are irrelevant. [1]. Orb-v2 is one of the fastest SOTA MLIP models (Figure 5.14), primarily due to its highly simple architecture and relatively small amount of parameters. It avoids loops and is therefore highly vectorizable, allowing for highly efficient batching and parallelization on modern hardware and uses torch-scatter [59] for message aggregation, which supports compilation via TorchScript [60]. They provide models pretrained on the Materials Project Trajectories (MPtrj) dataset [61].

3.1.5. eSEN (2025)

The Equivariant Smooth Energy Network (eSEN) model from X. Fu, Wood, Barroso-Luque, et al. [62] builds upon several earlier works from the authors, including eSCN [63] and SCN [64]. For simplicity, all papers from Zitnick are treated as a single conceptual contribution to eSEN, as they are tightly coupled. On the Matbench Discovery benchmark [57], eSEN is, as of August

[1] https://youtu.be/WPOjPButuE0?t=2541

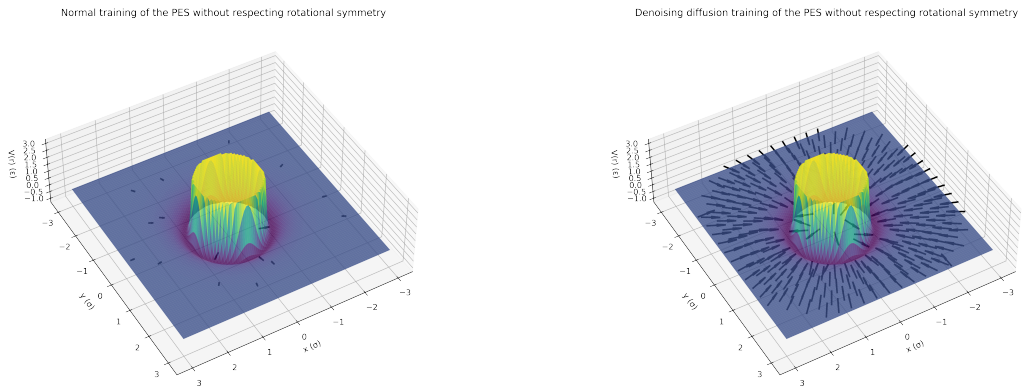

Figure 3.7.: Learning a specific PES without and with diffusion, not respecting symmetries. Left: Normal training on forces only. The model gets the exact force magnitude, but only sees a tiny PES subset. Right: Diffusion training to maximize data likelihood. The model sees a significantly larger PES subset, but only learns an averaged force direction with unit magnitude.

2025, the most accurate model across almost all categories, though the more recent Universal Models for Atoms (UMA) family [15] has not yet been put there. It is an implicit MLIP that employs a novel Special Orthogonal Group in 2D (SO(2)) equivariant architecture, which avoids e3nn tensor products [49], while still being equivariant to rotations. The first key idea is to remove all but one degree of freedom in the message passing step. Given two atoms i and j with their positions r_i and r_j, plus their embedding Irreps with corresponding Wigner-D matrix $D_{R_{ij}}$ performs a rotation R_{ij} that aligns the inter-atomic vector $r_{ij} = r_j - r_i$ with the z-axis. Generally, with one atom fixed at the center of the coordinate system, the other atom has three degrees of freedom to the first atom left, prompting a full SO(3) equivariant architecture. Zitnick, Das, Kolluru, et al. [64] rotate the coordinate system such that atom j lies on the positive z-axis, removing two more degrees of freedom. Then, the eSEN performs the message passing

$$m_{ij} = D_{R_{ij}^{-1}} f_{\text{message}}(D_{R_{ij}} h_i, D_{R_{ij}} h_j, |r_{ij}|, \dots)$$

rotating the result back to the original reference frame. The message network still sees the spherical channels displayed relative to its frame of reference and can therefore still make out relative information of its neighbors but only has to learn this with one varying degree of freedom, the azimuthal angle θ around the z-axis, which f_{message} needs to smoothly convolve over. Spherical harmonics become now become independent of the azimuthal angle θ and are therefore reduced to circular harmonics, which only depend on the polar angle ϕ (see paragraph 2.4.1) as shown in Figure 3.9.

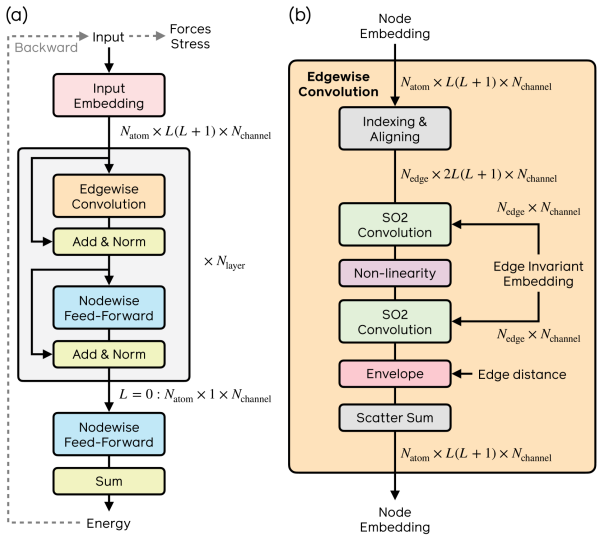

Figure 3.8.: eSEN architecture overview: Faster SO(2) convolutions emerge naturally when only one degree of freedom remains [62].

The second key idea is to make use of the fact that a convolution in the spatial domain is equivalent to a multiplication in the frequency domain [65]:

$$(f * g)(x) = \int f(y)g(x-y)dy \quad \Longleftrightarrow \quad \mathcal{F}(f * g)(k) = \mathcal{F}(f)(k) \cdot \mathcal{F}(g)(k)$$

And because just like fourier coefficients the spherical harmonics are such frequency components, the convolution of coefficients $x_{l,m,c}$ with filters $h_{l,m,c,l',c'}$ can be expressed as

$$y_{l,0,c} = \sum_{l'c'} h_{l,0,c,l',c'} x_{l',0,c'}$$

where l is the order, m the degree, and c the channel index. But as the coefficients might be complex,

$$\begin{pmatrix} y_{l,m,c} \\ y_{l,-m,c} \end{pmatrix} = \sum_{l'c'} \begin{pmatrix} h_{l,m,c,l',c'} & -h_{l,-m,c,l',c'} \\ h_{l,-m,c,l',c'} & h_{l,m,c,l',c'} \end{pmatrix} \begin{pmatrix} x_{l',m,c'} \\ x_{l',-m,c'} \end{pmatrix}$$

is the general formula they use and provide.

Even though in speed ahead of the previous e3nn architectures with their notoriously expensive tensor products, eSEN trails behind the completely non-equivariant Orb models in speed, at least in theory. Not only due to its implicit architecture, which requires a second backpropagation step to compute forces and stress from the energy, but primarily because every message passing step requires a custom Wigner-D matrix for rotations into and out of the local frame of reference, a costly endeavor.

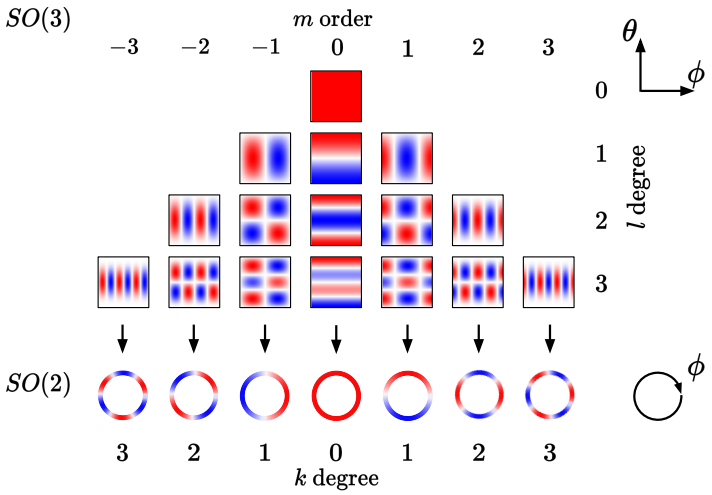

Figure 3.9.: Spherical harmonics $Y_m^{(l)}(\theta, \phi)$ reduce to circular harmonics $C_m^{(l)}(\phi)$ when θ is removed [63].

3.1.6. Orb-v3 (2025)

The Orb-v3 models from Rhodes, Vandenhaute, Šimkus, et al. [58] improve on the previous Orb-v2 models. Keeping the same symmetry-agnostic diffusion pretraining approach, they heavily improve upon the previous version in both speed and accuracy. They provide both a

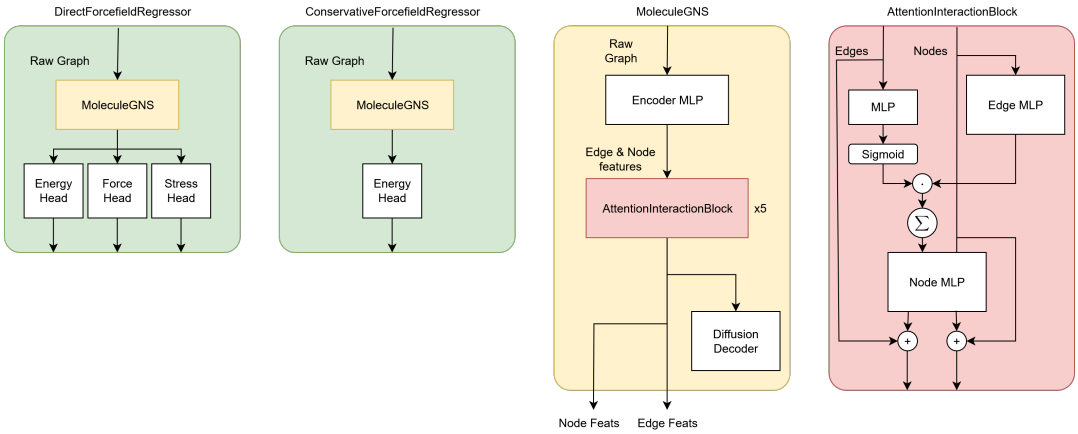

Figure 3.10.: Orb v3 architecture overview.

conservative and direct-force version of the model, just like they did for Orb-v2. The explicit, direct-force version of Orb-v3 is distilled from the implicit, conservative model [58]. Novel to v3 is a clever regularization technique that encourages the network to learn equivariance during training, with minimal computational overhead during inference. They craft a fixed SO(3) rotation matrix R that is matrix-multiplied with the atomic position matrix before passing

them to the model. They then use $||\nabla_R E||_2^2$ as an additional loss term, encouraging the model to ignore rotations of the input to the overall energy prediction.

```
generator = torch.zeros_like(cell, requires_grad=True)
rotation = rotation_from_generator(generator)
positions = (rotation @ positions[:, :, None])[..., 0]

inputs = [positions, displacement, generator]
grads = torch.autograd.grad(
  outputs=[energy], # (n_graphs,)
  inputs=inputs, # (n_nodes, 3)
  grad_outputs=[torch.ones_like(energy)],
  # ...
)

rotational_grad = grads[2]
rotational_grad_rms = torch.linalg.norm(
  rotational_grad,
  dim=(1, 2),
).mean()
loss += rotational_grad_rms * rot_grad_weight
```

Listing 3.1: Orbs v3 equivariance regularization loss. The model is encouraged to ignore small rotations of the input to the energy

Additionally, they include angular spherical harmonic embeddings as features and use 8 Bessel bases instead of the Gaussian RBF expansion [13] from Orb v2. Even though gaussian RBF are C^∞ continuous, they are ill-suited for the task. Bessel functions are oscillatory and better suited for encoding wave-like functions:

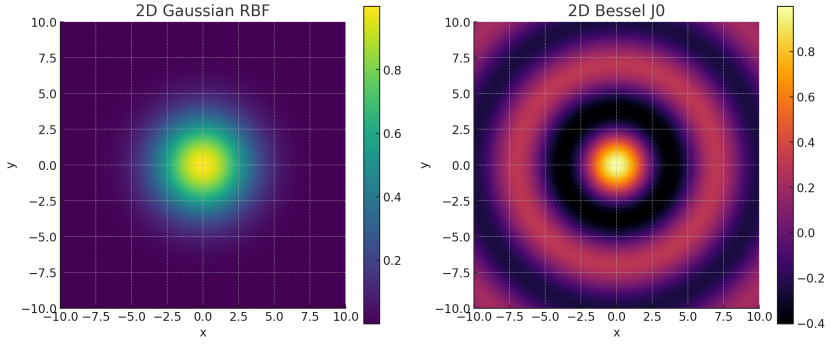

Figure 3.11.: Gaussian radial basis functions (left) are smooth, but not well suited to represent oscillatory functions like waves. Bessel functions (right) are oscillatory and better suited for this task.

The overall simple architecture avoids any expensive equivariant operations and allows a streamlined vectorized implementation. Additionally, Orb-v3 includes an online-trained confidence head similar to AlphaFold [66] that predicts the expected error of the model, allowing the user to filter out potentially bad predictions.

3.1.7. UMA (2025)

The UMA models from Wood, Dzamba, X. Fu, et al. [15] are the latest addition to the family of MLIPs. They improve the eSEN family, by training on a vastly larger and more diverse dataset

and by using a clever architecture change to achieve speedups making it almost competitive with the conservative Orb-v3 models. They introduce Mixture of Linear Experts (MoLE) layers,

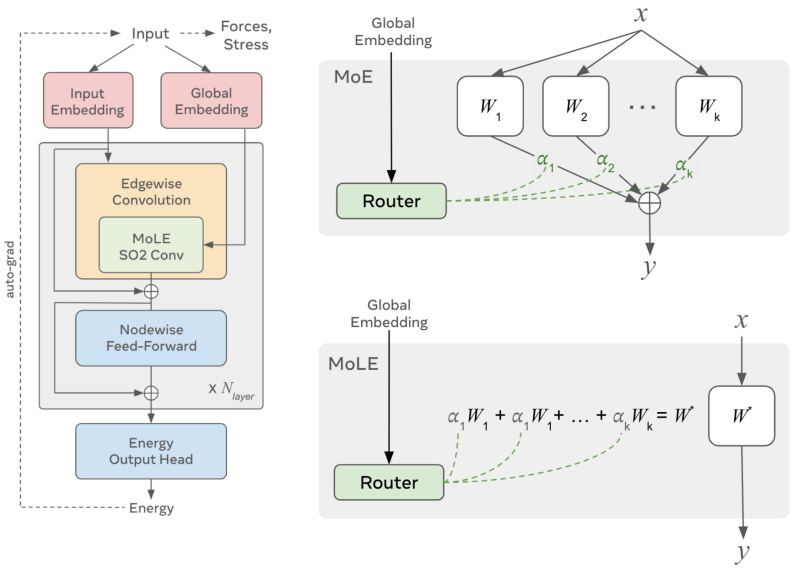

Figure 3.12.: UMA architecture overview: MoLE use fewer active parameters [15].

essentially a mixture of experts [67] variant that can be pre-computed with a global embedding of the atomic environment and global properties like net charge or spin. This allows the model to adapt its architecture to the specific task at hand, reducing the amount of active parameters significantly. While this works quite well for static MD simulations, any changes in the atomic environment requires re-computing the networks parameters which are still the largest in any MLIP to date. Just like eSEN, they use a conservative implicit architecture, requiring a second backpropagation step to compute the forces and the stress from the energy derivative.

4. Implementation

4.1. Orb Modification Attempts

The initial self-assigned goal of this thesis was the modification of an established SOTA MLIP architecture [58] to investigate effects of different design choices on model accuracy and performance. For example, could sparsely-gated Mixture of Experts (MoE) layers [68] be used to speed up inference in an existing model? Ultimately Orb-v3 was chosen, what seemed like a good trade-off between training cost, inference speed and accuracy. Although eSEN [62] had both a training script available and was more accurate, its training was significantly more expensive, and inference took orders of magnitude longer. But there is a caveat: most modifications require a complete retraining of the model from scratch, an expensive and highly time-consuming task. To complicate things further, the published source-code of Orb did not contain the DDPM-based pretraining code, which had to be manually re-implemented from scratch based on works from Zaidi, Schaarschmidt, Martens, et al. [55] and some leftover hints in the code-base.

Although a working pretraining script could be implemented successfully, the training of just a single epoch of MpTraj, a dataset used for training Orb-v3, the diffusion pretraining took around 12 hours on an NVIDIA RTX 4080 and about 10 hours on a single NVIDIA H100 node. This made it practically impossible to try out different model modifications in a reasonable timeframe, which made it often worse than it was before, both in accuracy and speed. The implementation of sparsely-gated MoE layers [68] was attempted multiple times, where heavy roadblocks were encountered. The first major issue was Orb's highly convoluted and over-engineered code-base. Making changes that don't break *something* were both quite challenging in general and time-consuming. The second major issue was the fact that Orb does not make use of a batch dimension, it just merges the different batch graphs into to a single input graph.

This has huge and far-reaching implications, especially for MoE layers which use the batch information for subtle finetuning of the gating network to avoid overloading single experts. While in theory a batch dimension could be specifically resolved to these MoE layers, it would most definitely require a custom Compute Unified Device Architecture (CUDA) [69] kernel for the actual expert computation to be efficient, which is out of scope for this thesis. For all these reasons, we moved on from the initial idea of modifying Orb-v3.

4.2. MD Frameworks

4.2.1. ASE

ASE is a MD framework tool-set for `Python` with a strong focus on simplicity and ease-of-use [70]. Even though most researchers prefer GROMACS [71, 72] or LAMMPS [73] for maturity, speed and their community, ASE's simplicity and the fact that it is written in `Python` makes it the preferred choice for ML researchers, as most models are also written in `Python` [59, 74]. The key strength of ASE is the *calculator* interface, a universal interface that takes as input an `ase.Atoms` object containing the atomic positions, species, cell vectors and boundary conditions of the system, and outputs the computed potential energy and forces acting on each atom. Optionally, these calculator interfaces can choose to provide other properties like stress for constant number of particles, pressure and temperature (NPT) simulations or charges for electrostatic interactions. This allows ASE to integrate potentials and dynamics from the vast majority of MD frameworks, simply by implementing this interface [75] for their software. Additionally, ASE comes with a DFT calculator interface out of the box: GPAW [76]. This allows for easy and frictionless switching between different potentials from empirical to DFT, allowing for easy benchmarking and evaluation of different MLIP models.

4.2.2. LAMMPS

The Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is a highly scalable and parallelizable MD simulation framework leveraging empirical and classical potentials [73]. It uses domain decomposition to distribute the workload of spatial domains to different workers [77]. This allows systems with millions of atoms to be simulated efficiently, even on longer time scales, making it a frequent choice for researchers in material science. LAMMPS is primarily maintained by Axel Kohlmeyer and a team of developers at Sandia National Labs [78], who were quite helpful when we reached out.

Packages A key strength of LAMMPS is its matured ecosystem of around 60 different *packages* [79], that extend the core functionality to meet a wide variety of demands. These packages can optionally be compiled along with the core software and provide ad-hoc functionality to the user. They rarely interact with each other and can often be used independently, making for a highly modular software. Examples include `KSPACE` with a long range electrostatic solver like Particle-Particle Particle-Mesh (PPPM) [80], the `KOKKOS` package for compilation of established styles[1] to CUDA [69] kernels [81, 82], and the `ML-IAP` package for interfacing mlip, as described in 4.2.4.

Usage To run a LAMMPS simulation, an input script needs to be provided by the end-user that specifies the initial atom configuration, hyperparameters like simulation duration, the

[1] property computations

timestep size, the integrator, what to output and the potential(s) used. An example of a generic input script can be found in Listing A.3 in the Appendix A. For our purposes, we will use the `python` interface and installation of LAMMPS, allowing us to run LAMMPS simulations directly from `Python` scripts.

4.2.3. Integrating MLIPs into ASE

The overwhelming majority of MLIP models provide a ready-to-use interface for ASE in the form of a calculator [75] class [58, 48, 15, 83]. Because ASE is written in `Python`, and it's abstaining from performing complicated domain decomposition, the integration of MLIP models is very straightforward, see Listing A.4 in Appendix A for an example. After linking the *calculator* described in subsection 4.2.1, ASE calls the `calculate` method and fetches the computed properties as required.

4.2.4. Integrating MLIPs into LAMMPS

As of writing this thesis, integrating MLIP models into LAMMPS is still a complicated and tedious task. There are two big roadblocks one will encounter for all integration pathways:

Programming language gap. Even though the overwhelming majority of LAMMPS is written in `C++`, the overwhelming majority of ML models are written in Python, and not all of them use the same framework: Some researchers use PyTorch [59] for its unmatched flexibility and ease of use, while others might prefer to use JAX [74] from `Google` for its superior performance. So either one ports the entire model into `C++` using `torch.jit.compile` [59], `jax.jit` [74] or by herself; or she creates a bridge between the two languages using Cython [84] and Numpy [85] as `C++` array interface for `Python` code.

The first option, Python to `C++` compilation, is definitely faster and easier to use, provided it does actually compile. For example, when trying to port the *Orb model* [58] using TorchScript [59], it failed to compile almost all the model's submodules due to unsupported python operations like `*kwargs` arguments and `Optional[]` types, which are non-trivial to remove in a large codebase in a limited time-frame. Also, this particular model family makes use of an adaptive graph construction scheme, which also seems to clash with TorchScript.

A `C++`-Python bridge is naturally slower, but framework-agnostic and significantly more versatile, given one has time and expertise to write the complicated `cython` interface, which is often non-trivial and must be done with the utmost care to avoid memory leaks and segmentation faults.

Different force philosophies. LAMMPS is primarily built around the idea of pair styles or pair potentials [79]. Pair potentials don't immediately compute the per-atom force F_i at-once, they compute intermediate forces

$$F_{ij} = f(r_{ij}, \dots) \tag{4.1}$$

between two atoms i and j, usually as a function of their distance r_{ij} first. Then they sum up all pairwise forces per atom, resulting in the overall force on an atom i [86]:

$$F_i = \sum_j F_{ij} \quad \text{and} \quad F_j = \sum_i -F_{ij} \tag{4.2}$$

While this approach is mostly reasonable for most empirical potentials, MLIPs are designed to output the overall per-atom forces F_i directly or as a gradient of the potential energy U with respect to the atomic positions $\vec{r}_i$: And that is not just a different design philosophy, it is the very reason why MLIPs are as accurate as they are. They should ideally operate holistically in the atoms' neighborhood,

$$F_i = f(\{r_{ij}, r_{ik}, \dots\}, \dots) \quad \text{with } j, k \in \mathcal{N}(i) \tag{4.3}$$

instead of focussing just on pairwise interactions that fail to capture important many-body effects like angular dependencies (e.g. in H_2O). To bridge this gap, two options make themselves available: In the philosophy of GDML from Chmiela, Tkatchenko, Sauceda, et al. [42], one can task implicit (conservative) models to output pairwise forces [87] as a gradient of the potential energy U with respect to the inter-atomic distance vector $\vec{r}_{ij}$:

$$F_{ij} = -\frac{\partial U}{\partial \vec{r}_{ij}} \cdot \frac{r_{ij}}{||r_{ij}||} \tag{4.4}$$

which allows one to apply them in the pairwise force philosophy of LAMMPS. Only a handful of models offer support for this type of readout, as it adds computational overhead and integrational complexity, and they only do this to support LAMMPS [87]. Not only is conformity to pair forces tedious and must be redone for every model, it can result in *significant* performance degradation. Packages in LAMMPS generally expect pairwise potentials to be fairly inexpensive, and will not shy away from calling pair potentials as often as they like, in the extreme case once per atom per step [88], which will – provided the model lacks a smart caching strategy – result in a slowdown in the worst-case by a factor of N, where N is the number of atoms in the system. The other possibility is to modify LAMMPS's core to allow for direct per-atom forces, which is not trivial and requires at least some knowledge of `C++` and LAMMPS's internal workings.

Option 1: New LAMMPS package

The most complicated, time-consuming and redundant way of integrating MLIPs into LAMMPS is the creation of a new package that provides a custom `pair_style`. This would require writing a new `pair_style` class in `C++` that interfaces with our MLIP models, either via `C++`-Python bridge or by porting the model to `C++` as discussed in paragraph 4.2.4. While this is by far the most flexible option with the highest level of control, it requires the most interation with LAMMPS's core codebase, is the most time-consuming and might need additional maintenance in the future. And some popular models like MACE [89] were directly ported into `C++` and

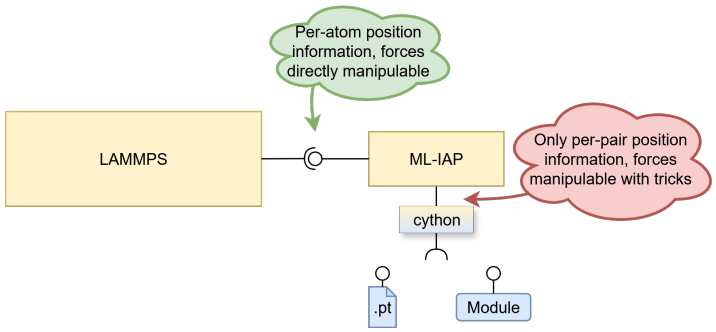

Figure 4.1.: Integrating MLIPs via ML-IAP into LAMMPS. Blue means written in Python, yellow means written in C++.

given a custom package [90], which shows that in theory this option is at least possible. And even though it definitely can be the faster option in terms of MD throughput on the CPU, porting a model requires a lot of work from experts from the LAMMPS software team and the respective MLIP framework; a process which is not always possible, not guaranteed to offer the best performance, definitely not generalizable to other models and often not available to run on the GPU. All-in-all, this option didn't seem viable, realistic or time-efficient enough for our purposes.

Option 2: The ML-IAP package

To provide a universal interface for MLIP models, the Machine Learning Inter Atomic Potentials (ML-IAP) package was added to LAMMPS in 2020. It uses a `cython` [84] `C++` to `Python` bridge to interface with `Python` code, allowing for on-the-fly integration of models without the need for a port. To integrate a brand-new MLIP model into LAMMPS when acting through python, one can either specify a loaded python module inheriting from the abstract `MLIAPUnifiedInterface` class performing the model invocation, or specify a `.pt` file containing such conforming model serialized by `torch.save`. This means that at the time of writing, one can not load a JAX model from a file, although support for it is underway [91]. Again, this operation does not compile the model into `C++`; instead it just serializes the model weights and architecture into a Python `pickle` [92] file. A schematic overview of the standard, indented ML-IAP integration is shown in Figure 4.1. A big shortcoming of the provided `cython` bridge is, as shown in Listing A.5 in Appendix A, the absence of absolute atomic positions. One is just given the pairwise distance vectors, which is not often not enough. Not only do almost all models expect to receive absolute atomic positions, some model families like Orb [58, 54] actually require them for efficient graph construction. The `ML-IAP` package typically expects the model to output the pairwise forces F_{ij} instead of the overall per-atom forces F_i, as discussed in paragraph 4.2.4. This means that one either folds and rewrites the model to conformity like NequIP [48, 93] did [87], or the package is modified, necessitating a complicated compilation process.

Option 3: Metatomic

In search for integration alternatives, Dr. Guillaume Fraux from the EPFL, the Federal Polytechnic School of Lausanne (fr.: École polytechnique fédérale de Lausanne), brought his `Metatomic` [17] project to our attention. It aims to provide a universal MLIP model interface for not only LAMMPS, but also ASE, GROMACS [71, 72], i-PI [94] and many other frameworks, meaning that once a model is compatible with `Metatomic`, it can be used seamlessly in all supported frameworks without any additional work.

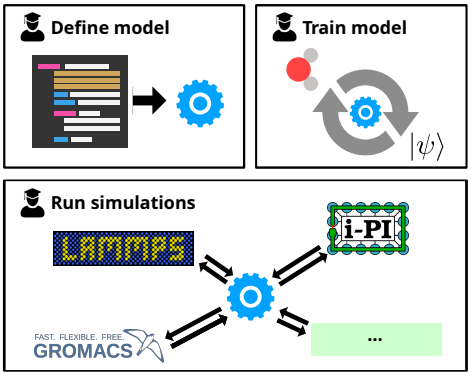

Figure 4.2.: Not only does Metatomic integrate into all frameworks seamlessly, it also allows for model training after porting. [17]

Discussing the Options

Option 3 would have been the most elegant, future-proof and helpful solution, so it was naturally considered first. The primary downside is a `C++` porting requirement with `torch.jit.compile` [59], neither always possible for torch models, nor applicable to `JAX` models like Nequix [16]. After a short correspondence with Dr. Fraux it became clear that his solution is not yet viable for our purposes. Option 1 was clearly a last resort in case the `ML-IAP` package would turn out unpatchable, which left us only with option 2, the `ML-IAP` package.

Modifying LAMMPS `ML-IAP` already had access to the `atoms->x` array naturally, but didn't provide it to the `cython` interface. To give the python code access, a pointer to the position array `atom->x` was added to the `MLIAPData` class, which is then wrapped by a `cython` interface class `MLIAPDataPy` which one can then use in python. For an efficient and zero-copy transfer of the `C++` array to python, numpy [85] is used as an interface. The only problem was that C++ arrays don't carry information about their length, as they are just pointers to the first element in memory. So we looked for a variable in LAMMPS that would provide us with the number of atoms in the system, which sounds easy but was anything but. `atoms->ntotal` exists, but this number varies over time, even with the same number of atoms as input, due to – which took some figuring out – the presence of *ghost atoms*. LAMMPS uses so-called

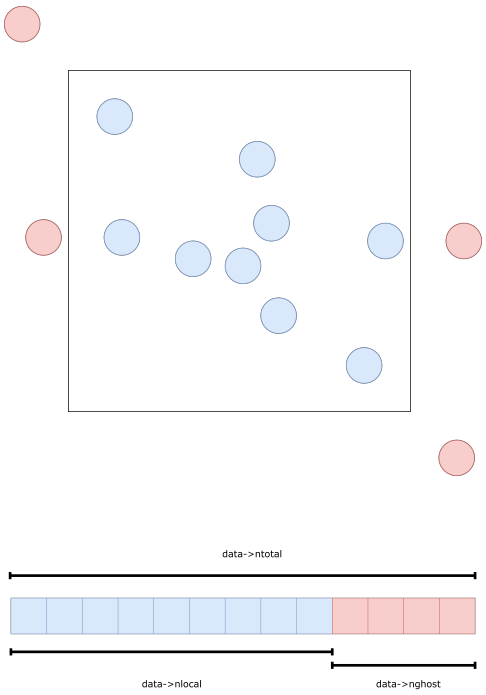

Figure 4.3.: Ghost atoms in LAMMPS: (Top) They are used to simulate periodic boundary conditions. (Bottom) Memory layout of `atom->x` and `atom->f`. Local atoms are displayed in blue, ghost atoms in red.

ghost atoms for inter-worker communication, allowing the simulation of larger systems via domain decomposition as described by Plimpton [77], so it wasn't exactly clear what type of atom can be found at what position in the `atoms->x` array. After some digging through the LAMMPS source code and some trial-and-error, two facts became clear: Both `atom->x` and `atom->f` contain the positions and forces of *all atoms*, local and ghost. Secondly, the arrays start with local atoms, directly followed by all ghost atoms as shown in Figure 4.3 b). We use all atoms, local and ghost as input to our MLIPs, because ghost atoms too can exert forces on real atoms as shown in Figure 4.3 a). We only apply the predicted forces to our local atoms, and postulate that if there exists a ghost atom, it is updated by *exactly the same* force predictor. This is important to ensure that Newton's third law holds, i.e. that every action has an equal and opposite reaction. As was learned only later, LAMMPS uses ghost atoms not only for inter-worker communication, but also to simulate periodic boundary conditions as shown in Figure 4.3 a). Models are informed that the system is non-periodic and the rest is handled by LAMMPS automatically. This also means that models that previously didn't support periodic boundary conditions, now do. Additionally, if one is not interested in the stress tensor, the original cell can simply be replaced by a large cubic cell, which we did for simplicity and to avoid any potential issues with out-of-cell atoms. We recognize that this solution requires

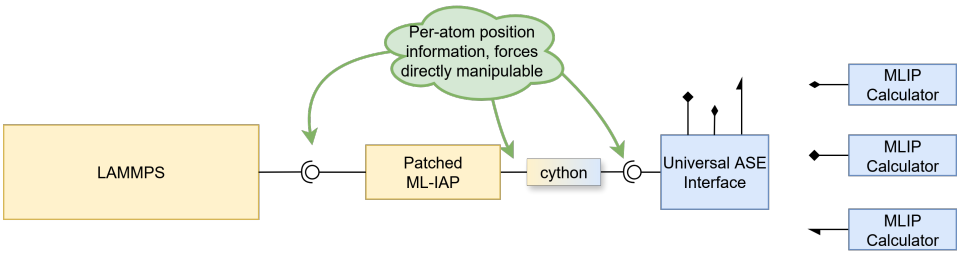

Figure 4.4.: Building a universal ASE adapter for LAMMPS ML-IAP integration. Blue means written in Python, yellow means written in C++.

the MLIP to compute more interactions than strictly necessary, resulting in an overhead of approximately 20–30%. Future work could focus on adding support for periodic boundary conditions naturally, ignoring ghost atoms. For this purpose, cell and boundary information has also been made available in the `MLIAPData` class. The full patch for LAMMPSs `ML-IAP` package can be found in Listing A.6 in Appendix A.

Compiling LAMMPS Building LAMMPS from source was a complicated and difficult task. After days of failed compilation attempts on Windows®, Debian Trixie [95] on the Hyper-V® [96] hypervisor was employed, which initially proved stable for compilation and testing tasks. We set up a virtual environment using `anaconda`, installing the lammps package and its dependencies. Then we compiled the patched source code using `cmake` and `make` and replaced the conda-installed LAMMPS binary with our own patched one. The full procedure – adapted from NequIP [97, 48] – can be found in Listing A.11 in Appendix A.

All roads lead to ASE Pretty much all SOTA models provide an ASE calculator interface, which has become the de-facto standard. There is no reason to waste time and effort writing complicated integrations for each and every model, when one can instead just write a small `UniversalASEUnifiedMLIAPInterface` class that combines both worlds seamlessly. Listing A.7 has the full implementation of such a universal interface, Listing A.8 shows how to use it to integrate any MLIP model with an ASE calculator interface into LAMMPS. Note that only the `real` and `metal` LAMMPS units are supported. ASE uses `metal` units, eV for energies and eV/Å for forces, while LAMMPS `real` units use kcal/mol for energies and kcal/(mol · Å) for forces, which we automatically convert using the conversion factor $1\text{eV} = 23.0621\text{kcal/mol}$ [98].

4.3. Test Suite

To evaluate different models, a comprehensive benchmark suite was implemented in `Python`, allowing for easy addition of new models and benchmarks.

4.3.1. Structure

When first designing the test suite, ease-of-use and modularity were the main goals. For this reason, it was decided to use ASE [99] as the main MD framework, simply because all MLIPs provide a calculator interface, and it ships with a DFT calculator: Grid-based Projector Augmented Wave (GPAW) [76]. The idea is to simply give each model a `driver`-inheriting class that outputs an ASE calculator object for a specified model variation. We then write specific test scripts in `/tests` that run the benchmarks automatically for a specified "driver" and model variation, saving the results to `/results` where they can be plotted using scripts in `/plotting`, all in a single virtual environment. The folder structure of this test suite can be seen in Listing A.9.

Dependency Conflicts As more and more models were added to the test suite, the dependencies started more and more to clash. For example Nequix [16] requires the `matscipy` [100] package for a single function (graph construction), but `matscipy` requires `numpy` [85] version 1, while all other dependencies require at least version 2. This could be fixed by using a specific fork of `matscipy` that supports `numpy` 2, which was already in the making. Then it was necessary to add `matscipy = { git = "https://github.com/Jpx3/matscipy.git", branch = "25_numpy_2"}` to the `pyproject.toml` file to tell `uv` [101] to use that fork. But this causes yet another problem, as `matscipy` now generates a different version number depending on the git commit hash, namely `1.1.1.dev.x` where x stands for the number of commits as the last tag. But `pip` now thought that `1.1.1.dev.x` is older than `1.1.1`, so it would not accept it, which required additional patching and fiddling. And this was just one example of many. eSEN and UMA use the same `fairchem-core` package, but at different versions: `1.10.0` and `2.6.0` respectively. But they have removed support for eSEN in `fairchem-core` version 2, and `fairchem-core` 1 is not compatible with `numpy` version 2, and on and on it goes. These conflicting and ever-increasing dependency constraints also resulted in `torch` [59] being restricted to version 2.6.0, which does not offer forward support for CUDA 12.9.

Virtual Environments A good alternative would be to give each model its own virtual environment. Every model would get its own folder that contains both a `requirements.txt` and a `driver.py` file that executes the given tasks with the model. For example for eSEN, one could initialize the virtual environment with `python -m venv drivers/eSEN/.venv`, install the dependencies with `drivers/eSEN/.venv/bin/pip install -r /drivers/eSEN/requirements.txt` – which also includes the `common` dependencies like `ase` and `phonopy` [102, 103], add our `common` python module to the `PYTHONPATH`, and then run the driver with `drivers/eSEN/.venv/bin/python`

`m drivers.eSEN.eSEN task inference_time`. This way, each model can have its own dependencies and versions, and the test suite can be run in a modular way. Even though extensive work on this was started, porting the entire suite to this new structure was unfortunately not possible in the given timeframe. An example folder structure of this approach is given in Listing A.10.

4.3.2. Benchmarks

Diatomic Energy Potential One of the arguably earliest ways to compare models is by evaluating the potential energy between two atoms at varying distances. This test is particularly interesting, as no model has been explicitly trained on it, making it a test of generalization capabilities – both interpolation and extrapolation.

It lets us check whether the models correctly capture basic interatomic interactions, which represent only a tiny fraction of all possible interactions in a real atomic system. The argument can be made, that if a model fails to capture these "easy" interactions correctly, it might struggle with larger out-of-distribution systems, especially sparse ones.

To simplify the test a bit, we focus only on pairs of atoms of the same species. Modelling same-species diatomic interactions accurately is a non-trivial task that demands encapsulating effects ranging from Coulomb forces, exchange interactions, to vdW forces, and pauli repulsions. So while this test is simpler in nature, it should in theory still offer valuable insights into different models.

In practice, the test consists of placing the first atom at the center of a $25\text{Å} \times 25\text{Å} \times 25\text{Å}$ cell and spreading the other atom on all the 125 points of the unit sphere with radii from 0.5 Å to 5.0 Å and then tasking the models to compute the potential energy of each configuration. The results are then averaged over all 125 points for each radius and compared to DFT reference values. To investigate the effects of this rotation, we also plot the mean and standard deviation of the energy over all 125 points for each distance. In order to compute DFT reference values, we employ the GPAW [76] ASE calculator with MPI [104] for parallelization, using the PBE (GGA) functional with a LCAO-like partial wave basis (similar to TZP) on a real-space grid with a spacing of 0.075 Å using Finite Difference (FD). This rather small spacing was necessary for the ETDM eigensolver to converge properly on more intricate wavefunctions (Li-Li for example) where coarser grids fail.

Phonons Phonon analysis are used to investigate vibrational properties of a structure or material. Given a relaxed atomic structure, each atom in the structure is slightly displaced, the forces on all atoms computed by different models, and then eigenfrequencies and eigenmodes are calculated. The X axis of a phonon band structure plot shows the Brillouin zone path of the unit cell of the structure, while the Y axis shows the phonon frequencies ω in THz, equivalent to the vibrational energy of the phonon mode and directly related to the temperature. In essence, phonon band structure plots show at what frequencies atoms in a material vibrate harmonically when "excited" along specific directions in reciprocal space on a Brillouin zone

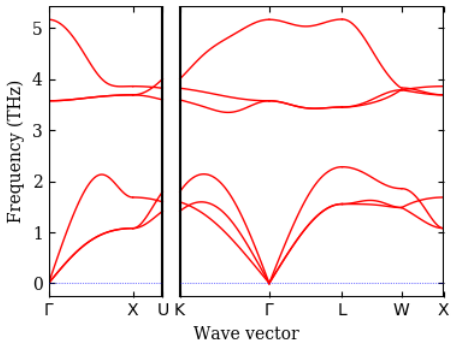

Figure 4.5.: Example phonon band structure of NaI [105].

path as shown in Figure 4.5. They serve as a fingerprint of a material's stability, uncover its ability to conduct heat and sound, show a structure's heat capacity, and can even be used to determine its phase-transition behavior [106]. We can compare phonon band structures produced by MLIP models to the DFT PBE references to see if the vibrational properties are captured correctly, which is crucial for many applications in materials science and condensed matter physics. We closely follow the procedures from X. Fu, Wood, Barroso-Luque, et al. [62] and Bandi, Jiang, and Marianetti [107], that – unlike this thesis – only benchmark their own or a very small subset of other models. We deliberately choose not to allow structure relaxation by the models before tasking phonon analysis, as is commonly done in literature [62, 107]. Instead, be make correctly identifying the structure to be in equilibrium a prerequisite, as we believe that this makes the comparison both fairer and more difficult for some models. Additionally, we purposefully refrain from performing any model-specific adjustments to improve phonon results, like for example adjusting model precision or applying strain to the structure before analysis [58], with the indent of replicating the default, out-of-the-box experience as closely as possible. The phonon analysis is performed with the `phonopy` [103, 102] python package on a randomly sampled subset of ≈ 400 structures of the `mdr-phonondb` [105] dataset with the default 0.01 Å displacement for force calculations. But not only can we compute the band structure, we can also compute thermal properties like the vibrational free energy $F_{\text{vib}}(T)$, the constant volume heat capacity $C_V(T)$ and the vibrational entropy $S_{\text{vib}}(T)$ and the phonon density of states (DOS) $g(\omega)$ as seen in Figure 4.6 While X. Fu, Wood, Barroso-Luque, et al. [62] investigate the effects of different displacement values, we stick to 0.01 Å for simplicity and computational cost. One full passthrough takes 4–8 hours depending on the model, so all models took a few days to complete. The results can be found in subsection 5.1.1 and subsection 5.1.2.

Heating Carbon-Dioxide Gas We want to investigate how well MLIPs can run carbon-dioxide gas simulations in a NVE ensemble depending on temperature. For this we both investigate energy conservation and leverage diffusion coefficients as a measure of how well the

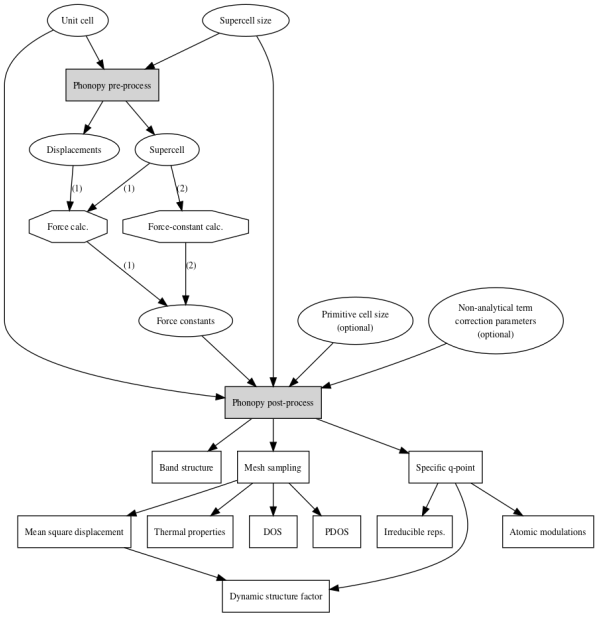

Figure 4.6.: Schematic of phononpy workflow used to compare different models [108].

MLIP captures the dynamics of CO_2 gas. The total energy of a system should be conserved in a NVE ensemble by definition. In MD simulations, the potential energy E_{pot} of the system is converted to kinetic energy E_{kin} and vice versa, keeping the total energy $E_{\text{tot}} = E_{\text{pot}} + E_{\text{kin}}$ constant over time. We evaluate energy conservation by running a system of 50 carbon-dioxide molecules (150 atoms) in a 25Å $\times$ 25Å $\times$ 25Å periodic box. The system is first thermalized to 300K by sampling atomic velocities from a Maxwell-Boltzmann distribution and running a NVT ensemble for 10ps with a time step of 1 fs. Then, we re-sample the atomic velocities again from the Maxwell-Boltzmann distribution at 400K to introduce a temperature jump, and run a NVE ensemble for 10ps again. We repeat this procedure a total of 10 times and keep track of all relevant parameters during the simulation. Additionally, we calculate and compare the diffusion coefficients obtained from the MD simulations to the experimentally obtained diffusion coefficients of CO_2 gas at different temperatures [109]. The diffusion coefficient D is a measure of how fast particles spread out over time, and can be computed from the Mean Squared Displacement (MSD) using the

$$D = \lim_{\Delta t \to \infty} \frac{1}{6t} \langle |r(t + \Delta t) - r(t_0)|^2 \rangle_{t_0} \tag{4.5}$$

which we approximate. A high diffusion coefficient means that particles are moving quickly and spreading out rapidly, while a low diffusion coefficient means that particles are moving slowly and spreading out more slowly. Capturing this behavior correctly is a requirement for later simulating gas adsorption in MOFs, so we want to make sure that our models can do this correctly. The relationship between temperature T, collision frequency $\Omega(T)$, and diffusion

coefficient D in gasses is

$$D \propto \frac{T^{3/2}}{p\Omega(T)} \tag{4.6}$$

where p is the pressure of the gas in bar [110]. The results can be found in subsection 5.1.4.

Inference Speed Being accurate is not the only requirement for MLIPs to be practical. Inference speed compared to ab-initio methods like DFT can often be equally if not more important, especially for long-running MD simulations. We measure inference speeds of force prediction as primary MD task on homogenous uniformly sampled systems of up 100.000 Helium atoms with an average density ρ of 0.033 atoms/Å^3. To reduce variance of measurements caused by startup tasks like Just-In-Time (JIT) compilation, a 40-step warm-up is performed on a system of 5 atoms. Then, we measure the inference speed of MLIPs on systems of 10, 25, 50, 100, 250, 500, 1.000, 2.500, 5.000, 10.000, 25.000, 50.000 and 100.000 atoms. We repeat the measurements a total of 20 times for each system size and model, and discard the first measurement to reduce size-depended startup tasks found in models like Orb-v3 [58]. To take caching out of the equation, no atomic systems were reused. While Rhodes, Vandenhaute, Šimkus, et al. [58] report up to 40% performance gains from reducing neighbor limits of Orb-v3 to 20, we assume that any practical application will always prefer to avoid neighbor limits[2] to reduce the risk of energy drift and instabilities in long-running MD simulations. We use an AMD Ryzen Threadripper PRO 5975WX 32-Core processor with 256GB of RAM for CPU inference and a NVIDIA® H100 NVL® GPU with 100GB of VRAM for GPU inference. When possible, all models were compiled using `torch.compile` [60] through ASE calculator options and for JAX models using JAX's `jit` [74] compilation. The results can be found in subsection 5.1.5.

4.4. MOF Experiments

Metal-Organic Frameworks (MOFs) have become highly popular among materials scientists, primarily due to their exceptional porosity and tunable properties [111]. They consist of metal nodes (ions or clusters) connected by organic linkers, resulting in a highly porous structure with an enormous internal surface area. For example, a single gram of MOF-5 has a surface area of 2900 m^2 [111], which is roughly equivalent to the area of half a soccer field. These features, among others, make MOFs particular interesting for gas storage and separation applications. Specifically, Mg-MOF-74[3] is of particular interest to the scientific community for its excellent carbon-dioxide adsorption [113]. CO_2 is the primary greenhouse gas responsible for climate change [115, 116, 117, 118], so materials that can efficiently capture and store carbon-dioxide [119, 9, 120] could play a critical role reversing pollution and its effects. MOFs can also be used to store other gasses like methane and hydrogen [10], making them versatile

[2]They advertise a neighbor limit of 120 as practically infinite
[3]Aka. CPO-27-Mg or Mg_2(DOBDC), DOBDC = 2,5-dihydroxyterephthalic acid [112]

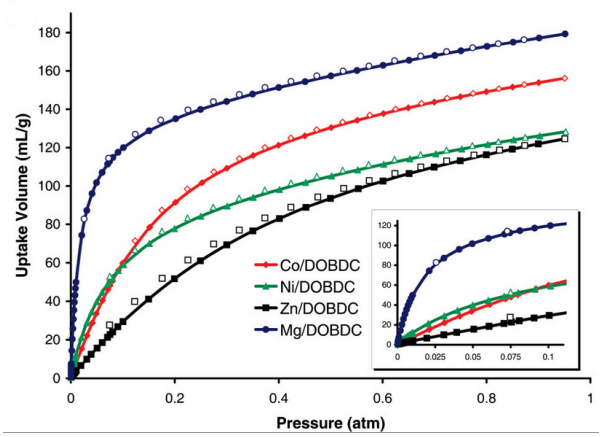

Figure 4.7.: CO_2 adsorption in different MOFs: Mg-MOF-74 has the highest adsorption capabilities [114].

materials for various gas storage applications. Having fast and accurate potentials to simulate CO_2 adsorption in different MOFs structures could greatly accelerate the discovery of new materials for carbon capture applications via HTS. The goal is to benchmark different MLIP models on their ability to simulate carbon-dioxide adsorption in Mg-MOF-74, compared to empirical potentials. The two primary forces acting on the CO_2 molecules in the MOF are vdW forces and electrostatic forces [9].

Electrostatic interactions are fundamental forces between charged particles and play a critical role in a wide range of physical, chemical, and biological processes. These forces can be modelled with Coulomb's law [121], which describes the potential interaction energy between two point charges in a vacuum as

$$E_{\text{elec}} = \frac{q_1 q_2}{4\pi\epsilon_0 r} \tag{4.7}$$

where q_1 and q_2 are the charges of the interacting particles, r is their separation distance, and ϵ_0 is the permittivity of free space. The $1/r$ dependence of electrostatic interactions makes them exceptionally long-ranged, meaning they cannot be neglected even at distances exceeding 25 Å. In our MOF example, the interaction energy between the Mg^{2+} metal node ($q \approx 1.45e$ [3]) in Mg-MOF-74 and the carbon atom of a CO_2 molecule ($q \approx 0.7e$ [3]) at a distance of 15 Å remains 0.9710 eV (141 kJ/mol). The corresponding Coulomb force, which decays with a factor of $1/r^2$, is still 1.07 eV/Å (103 kJ/mol), a non-negligible value. Nevertheless, MLIPs often restrict their "receptive field" to only a few Ångströms primarily for performance – typically 6–10 Å [58, 54]. Capturing such long-range interactions would solely rely on message-passing propagation, a process that would require the atoms to be connected through a chain of neighbors. This can be particularly challenging in porous materials like MOFs, where the MOF atoms are dense (meaning neighborhood limits are quickly reached) and often far apart from the gas molecules (cutoff distances are easily exceeded).

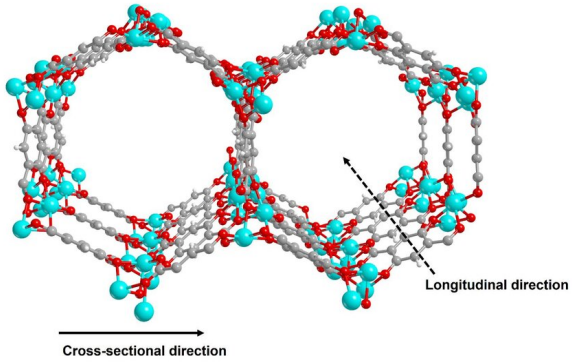

Figure 4.8.: Structure of Mg-MOF-74: Red, cyan, grey, and white spheres represent oxygen, magnesium, carbon, and hydrogen atoms [113].

Van-der-Waals interactions are the other primary force acting on CO_2 molecules in Mg-MOF-74, and as discussed in subsection 2.2.2 they require dispersion corrections to be modelled accurately in DFT simulations. The effects of vdW forces fall off with $1/r^6$ and are therefore rather short-ranged compared to electrostatic interactions. While some models like Orb-v3 [58] include dispersion corrections directly in their inference pipeline, it is possible to add dispersion corrections as a post-processing step to any MLIP model. In ASE this can be quickly done by jointly adding both model calculator and a dispersion calculator (like d3 for example) to a `SumCalculator` object that simply sums the energies and forces of both calculators. Contrasting the need for a post-processing correction, the UMA model family [15] is directly trained on various dispersion-corrected datasets, including `OMol25` [122, 123]. This means that UMA has learned to capture dispersion interactions directly from data, without the need for an explicit dispersion correction.

4.4.1. Chemical Potential

To investigate the chemical potential of a gas molecule in a reference structure, Widom insertion [124] can be used. We describe the primary forces acting on a CO_2 molecule in Mg-MOF-74 in section 4.4, so in theory following these forces correctly should result in fairly accurate chemical potential estimates. The process is simple: E_{struct} and E_{gas} are computed, the potential energy of the structure alone and the potential energy of the isolated gas molecule alone, respectively. Then, the gas molecule is randomly inserted into the structure at N uniformly sampled positions with random rotations, and then the combined energy $E_{total,i}$ of the structure with the inserted gas molecule is computed. This discrepancy between the combined energy and the sum of the individual energies gives insight into the interaction energy between the gas molecule and the structure at that insertion point. We perform Widom insertion of CO_2 molecules into Mg-MOF-74 at 300K with exactly 100,000 insertions per model. We then create a hexagonal top-down supercell with dimensions 50×50 Å and aggregate sampled energy points per cell with the excess chemical potential formula. The excess chemical

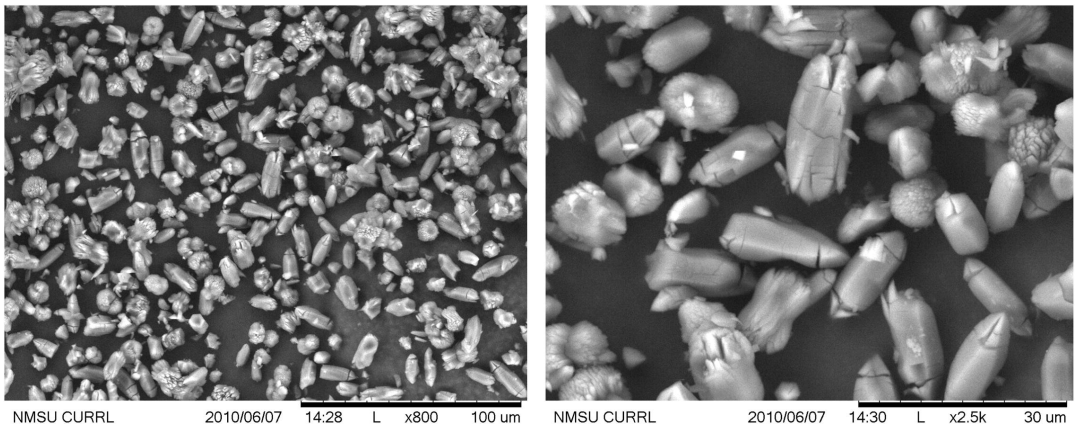

Figure 4.9.: A SEM image of a Mg-MOF-74 sample, particle size in the range of 5–25µm [125].

potential μ_{ex} can be estimated with

$$\mu_{\text{ex}} = -k_B T \ln\left(\frac{1}{N}\sum_{i=1}^{N}\exp\left(-\frac{E_{total,i} - (E_{struct} + E_{gas})}{k_B T}\right)\right) \tag{4.8}$$

where $k_B \approx 1$ is the Boltzmann constant and $T = 300$ is the temperature, both in electron volts. We convert the final result to kJ/mol for easier interpretability by multiplying with 96.485 kJ/(mol · eV). The results can be found in subsection 5.2.1.

4.4.2. Simulating Adsorption

But studying the chemical potential alone is not sufficient to understand the full adsorption process of CO_2 in Mg-MOF-74. To benchmark the models in a more realistic environment, we simulate the adsorption of CO_2 in Mg-MOF-74 via MD simulations. We closely follow and make use of the scripts provided by Y. Fu, Y. Yao, Forse, et al. [3] and adapt them to work with our MLIP models in LAMMPS, which previously simulated CO_2 adsorption in Mg-MOF-74 with classical force fields. Because they make use of LAMMPS for their Grand Canonical Monte Carlo (GCMC) simulation, the need for a MLIP bridge emerged and was fulfilled in subsection 4.2.4. A GCMC simulation is basically just a normal MD simulation with the structure fixed, with one additional step: every few MD steps, a molecule – in our case CO_2 – is either inserted or deleted from the system based on the chemical potential and temperature. This allows the system to equilibrate with a gas reservoir at a given pressure and temperature, simulating the adsorption process accurately. We simulate 10 different pressure settings from 0 to 1 bar at 300K each for 3,000,000 MD steps with a timestep of 2 fs. To improve the pressure resolution at lower pressures, the simulations use logarithmic pressure settings as shown in Table 4.1. For simplicity, we convert 1 bar to 100 kPa instead of 101.325 kPa during post-processing. Every 10 simulation steps, we dump the entire system state to a `lammpstr` file, about 8 GB per simulation. To compress the output data, we convert the

Simulation	0	1	2	3	4	5	6	7	8	9
Pressure (bar)	0.000	0.073	0.151	0.237	0.330	0.433	0.547	0.677	0.825	1.000
Pressure (kPa)	0.000	7.3	15.1	23.7	33.0	43.3	54.7	67.7	82.5	100.0

Table 4.1.: Pressure sampling points computed as $1 - \ln(\text{linspace}(e, 1, 10))$.

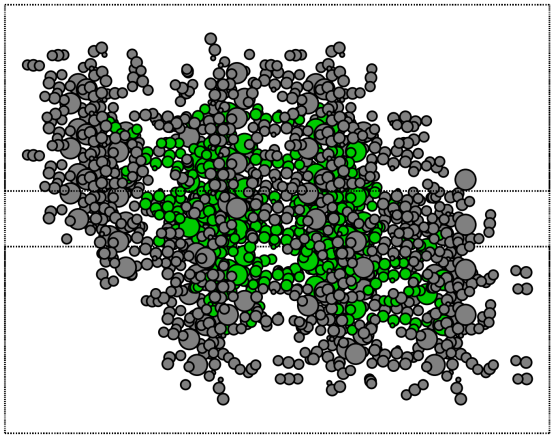

Figure 4.10.: LAMMPS simulating periodic boundary conditions of Mg-MOF-74: Green spheres are local atoms, gray spheres are ghost atoms. The triclinic cell has been replaced by a simple orthogonal box, as the periodicity is handled by the ghost atoms.

`lammpstr` files to a custom delta-compressed binary format with `gzip` compression, resulting in a trajectory file of about 100MB per simulation. To compute the equilibrium adsorption capacity, we only consider the last 50% of each simulation, and average the number of adsorbed CO_2 molecules over time.

Unfortunately, the adsorption isotherm calculation could only be performed using the Orb-v3 direct-force model and the original empirical potentials from Y. Fu, Y. Yao, Forse, et al. [3], as all other models were simply too slow. Running the fast direct-force Orb-v3 model on the LAMMPS-extended Mg-MOF-74 carbon-dioxide filled system[4] took about three days on multiple H100 NVL® GPUs running in parallel for all 10 pressure points with a time step of 2 fs. For this task we choose the Omol variant of direct-force Orb-v3 [58, 122], hoping it might better capture vdW interactions due to Omol25 [122] being trained on dispersion-corrected data. Using table Table 5.5 from the inference speed results, we estimate other models could take weeks or even months to complete the same task. The results of the two simulations can be found in subsection 5.2.2.

[4]with approximately 800 atoms, ghost and local

5. Evaluation and Interpretation

5.1. Test Suite Results

We now show and discuss the results obtained from various benchmarks performed on the test suite introduced in section 4.3. Different models are show in different colors for better visibility, DFT references are always shown in black. Not all models were available for all experiments, either due to technical difficulties in setting up the model or due to resource constraints.

5.1.1. Phonon Probing

This section inspects the results of the calculations described in paragraph 4.3.2. But before we start evaluating the overall phonon performances of the models, we probe a random material from the test suite, in this case a $BaPdF_4$ [126, 61, 127, 128] crystal, and examine the phonon band structures predicted by each model next to a reference structure produced by `Phonopy` [102, 103].

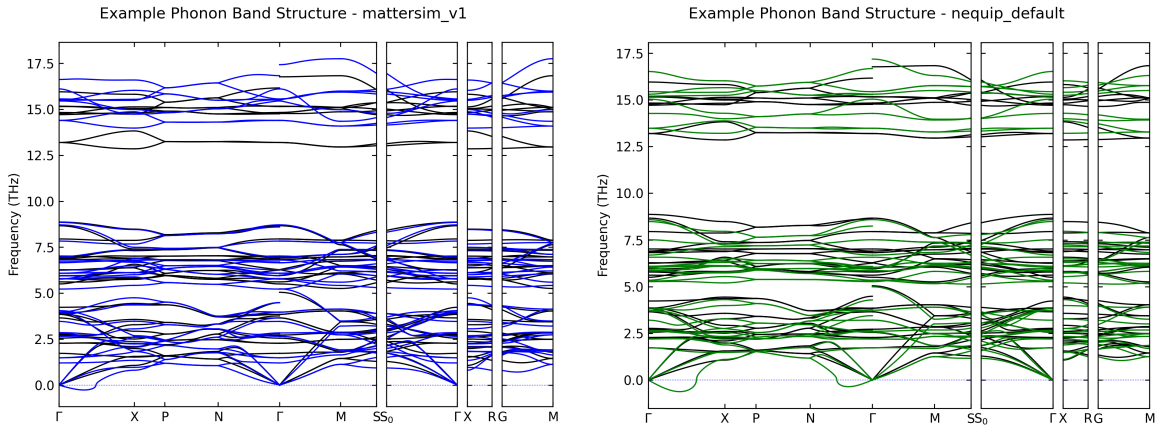

Figure 5.1.: Phonon band structures of mp-12623 predicted by MatterSim v1 (left) and Nequip (right). The DFT reference is shown in black.

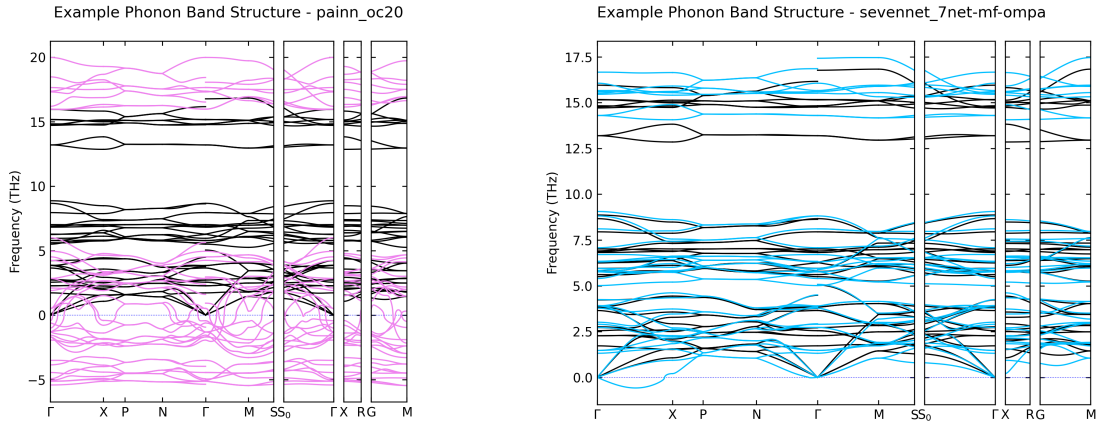

Figure 5.2.: Phonon band structures of mp-12623 predicted by PaiNN (left) and SevenNet (right). The DFT reference is shown in black.

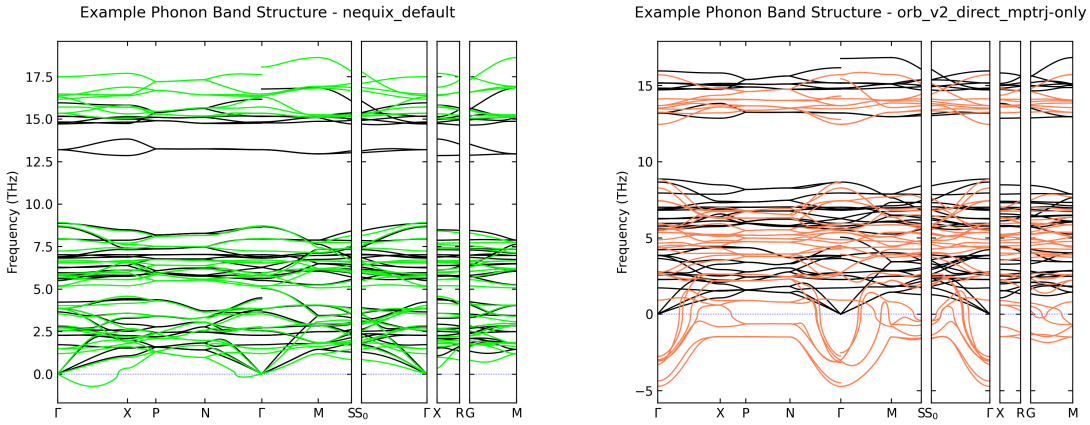

Figure 5.3.: Phonon band structures of mp-12623 predicted by Nequix (left) and Orb-v2 (right). The DFT reference is shown in black.

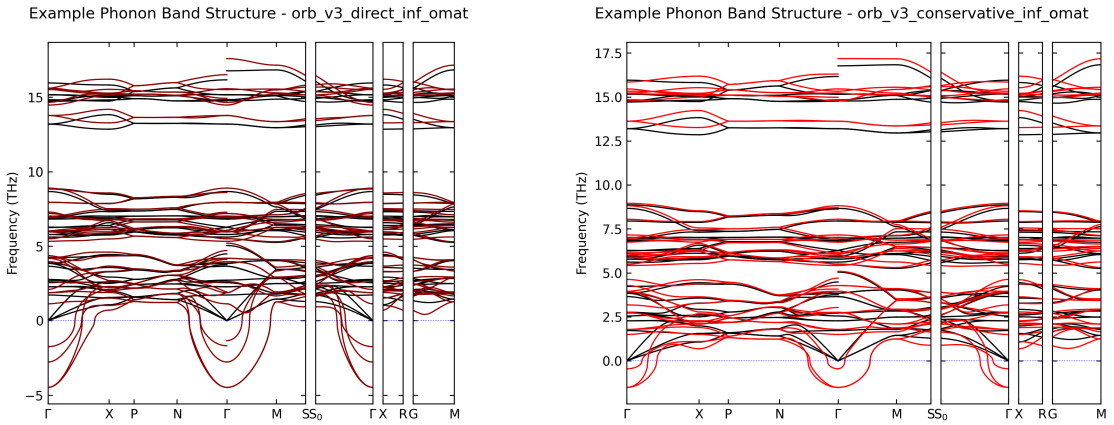

Figure 5.4.: Phonon band structures of mp-12623 predicted by the Orb-v3 direct-force (left) and conservative (right) models. Both run at fp-high precision, the default. The DFT reference is shown in black.

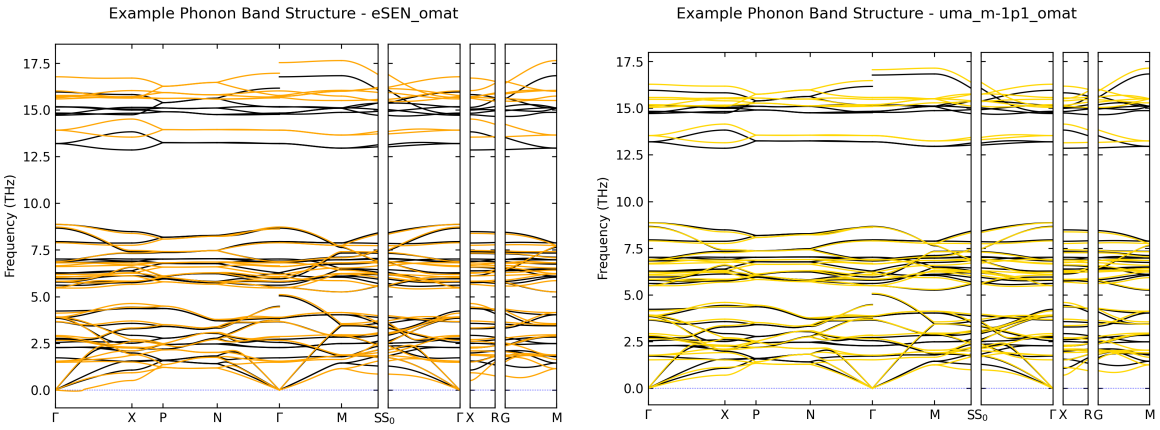

Figure 5.5.: Phonon band structures of mp-12623 predicted by eSEN (left) and UMA (right). The DFT PBE reference is shown in black.

Orb Imaginary Modes All Orb models predict imaginary frequencies for the lower acoustic modes, especially close to the Γ point, which are plotted as negative frequencies in the band structure[1]. Positive frequencies correspond to stable vibrations, meaning that the displacement of an atom results in an increase in energy required, while imaginary frequencies correspond to unstable vibrations resulting in the structure collapsing or transforming. The most likely explanation for these imaginary frequencies is that the Orb models don't consider the structure to be fully relaxed, a strong requirement for stable phonon calculations [129] as a direct result of an inaccurate PES prediction around the equilibrium structure. This is further supported by the fact that the conservative Orb-v3 model exhibits better[2] imaginary modes than the direct-force Orb-v3 model. As the direct model is distilled from the conservative model, the argument can be made that the distillation process made the predictions significantly less accurate for this particular crystalline structure.

Mode Frequency Shift Even though all models capture band gaps[3] quite well, they tend to overestimate the actual phonon frequencies of high-energy optical modes compared to the DFT PBE reference. The only exception is Orb-v2, which *underestimates all* phonon frequencies. In this example, UMA is the closest to the DFT PBE reference, with Nequip as a close second. UMA is also the only model to model the lowest acoustic mode along the $\Gamma - X - P$ path correctly, indicating an accurate understanding of the equilibrium PES of the material involved.

5.1.2. Phonon Benchmark

But how do the models perform on the phonon benchmark across the 400 different materials tested? Instead of just reporting the Mean Absolute Error (MAE) of phonon frequencies as is common in literature [58], we instead emphasize and practice the importance of looking at the distribution of errors, as they are often more telling than the mean alone. Thus, we show raincloud plots [130] of the deviations from DFT PBE, across all materials for different thermodynamic quantities derived from the phonon calculations.

[1] While not mathematically correct, it is common to call negative modes "imaginary" [129].
[2] Closer to reference
[3] Frequency difference between two phonon modes

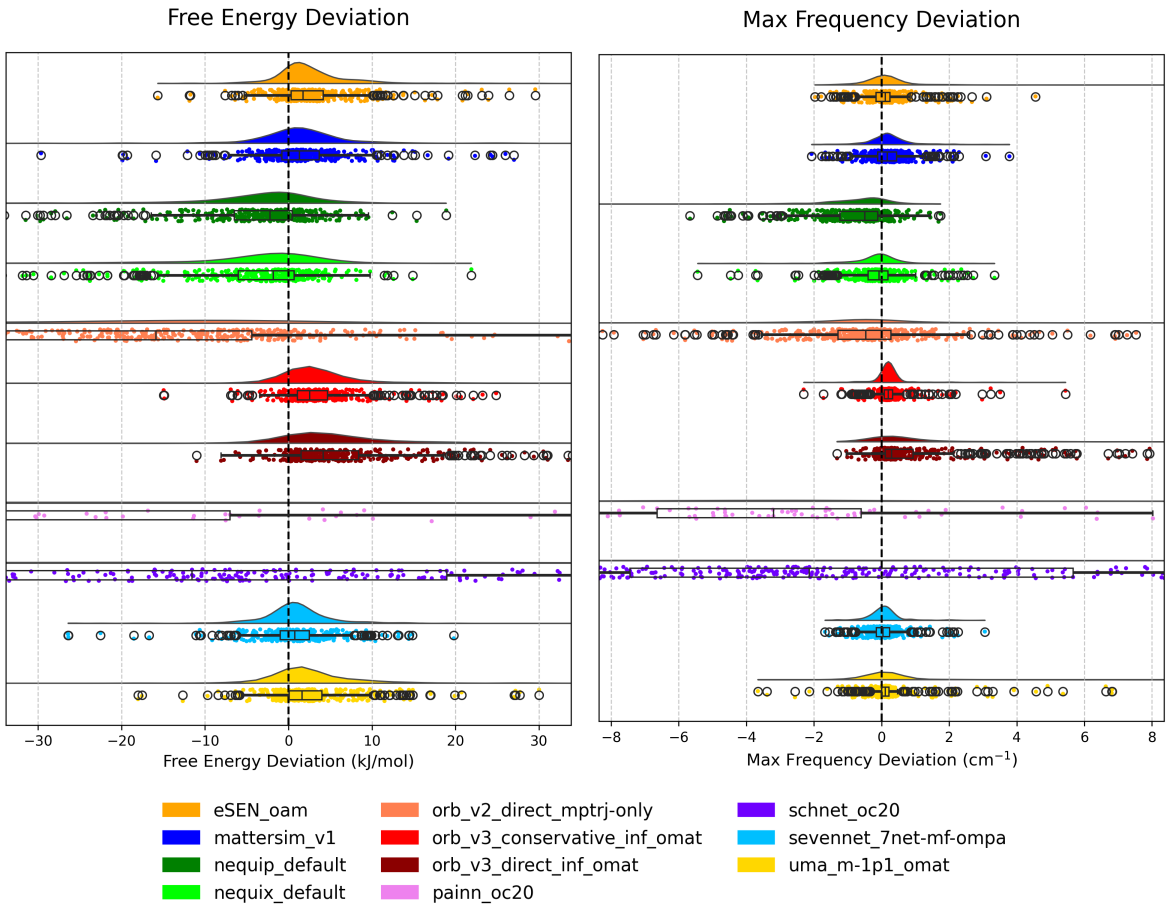

Figure 5.6.: Free energy deviation from DFT PBE reference across different materials (left). Maximum phonon frequency deviation from DFT PBE reference across different materials (right).

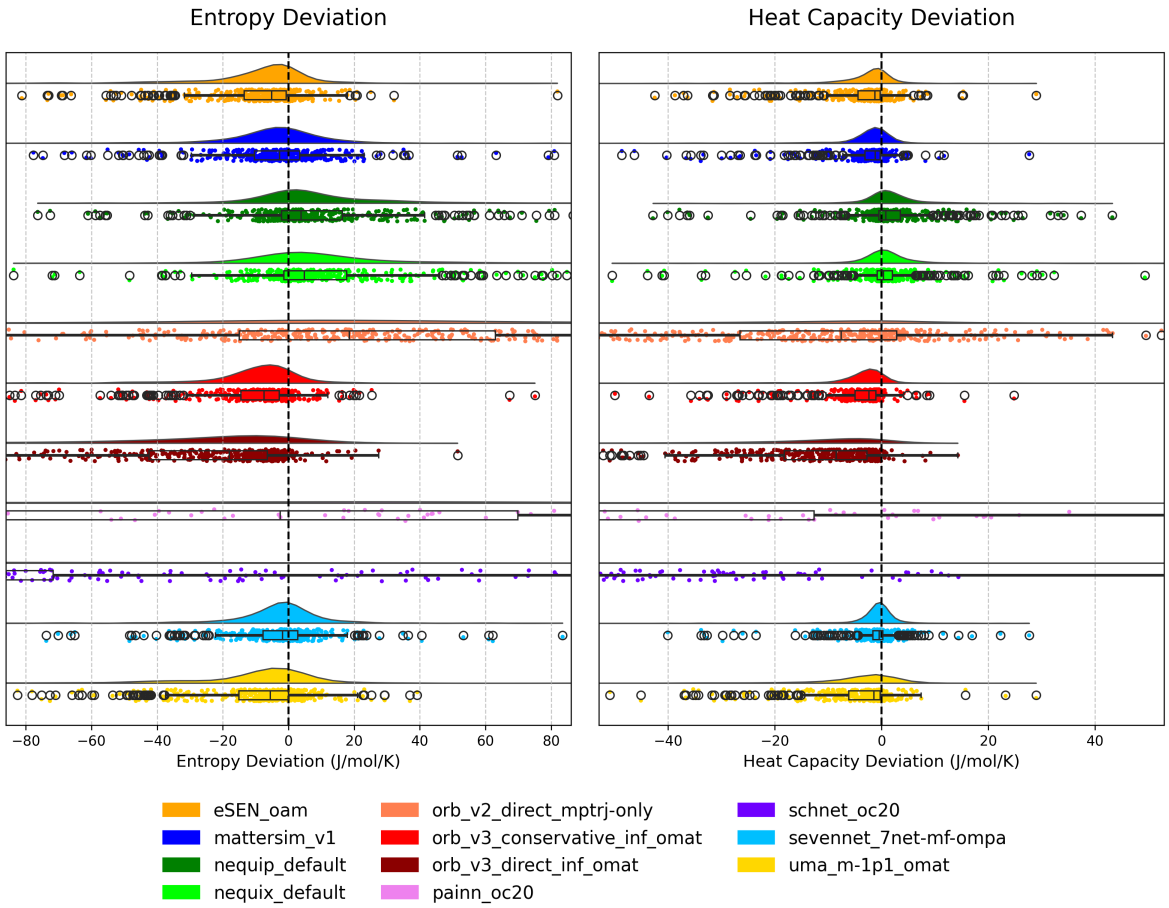

Figure 5.7.: Entropy deviation from DFT PBE reference across different materials (left). Constant volume heat capacity deviation from DFT PBE reference across different materials (right).

Overall Phonon Benchmark Performance SevenNet [131] seems to be the most consistent model across all tested materials on all thermodynamic quantities derived from the phonon calculations, closely followed by MatterSim v1 [61]. This could be due to the fact that both models were trained on the MPtrj dataset [61], which is a superset of the `mdr-phonondb` [105] dataset used for the phonon benchmark. The same is true for almost all other models except UMA [15], which performs quite well in this benchmark despite not being trained on the MPtrj dataset. Orb-v2 struggles despite being trained on MPtrj as well, likely due to underfitting. SchNet and PaiNN are not trained on MPtrj and fail to deliver meaningful results in this benchmark. The conservative Orb-v3 model was trained on MPtrj as well and performs reasonably well on all phonon-derived thermodynamic quantities, while its distilled direct-force counterpart performs significantly worse.

Fundamentally, our phonon frequency analysis probes the second derivative of the PES at some energy minimum of a structure [129], so these results might not be fully representative of how well the models perform in actual MD simulations. This is further discussed in chapter 6.

5.1.3. Diatomic Energy Curves

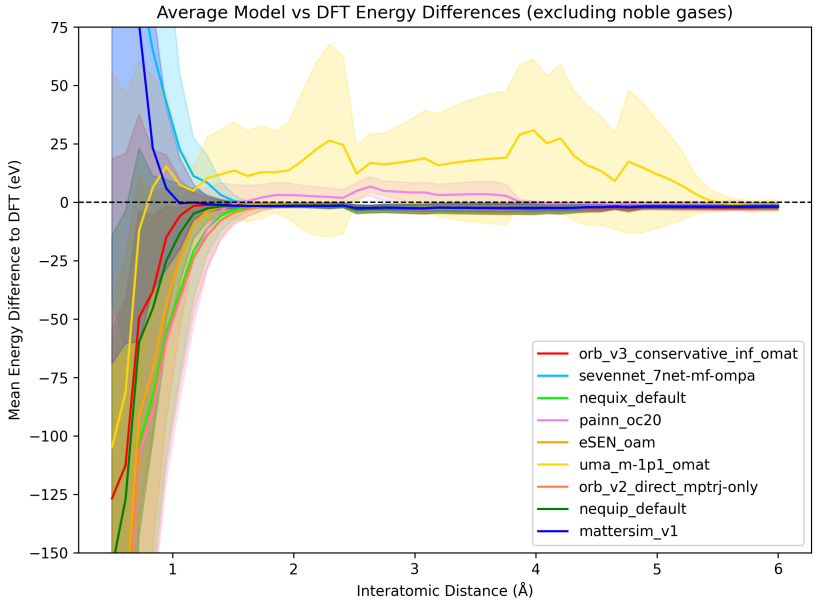

Figure 5.8.: Average Model vs DFT Energy Differences across selected elements. The highest region of uncertainty is below 1 Å, where most models struggle to predict accurate energies.

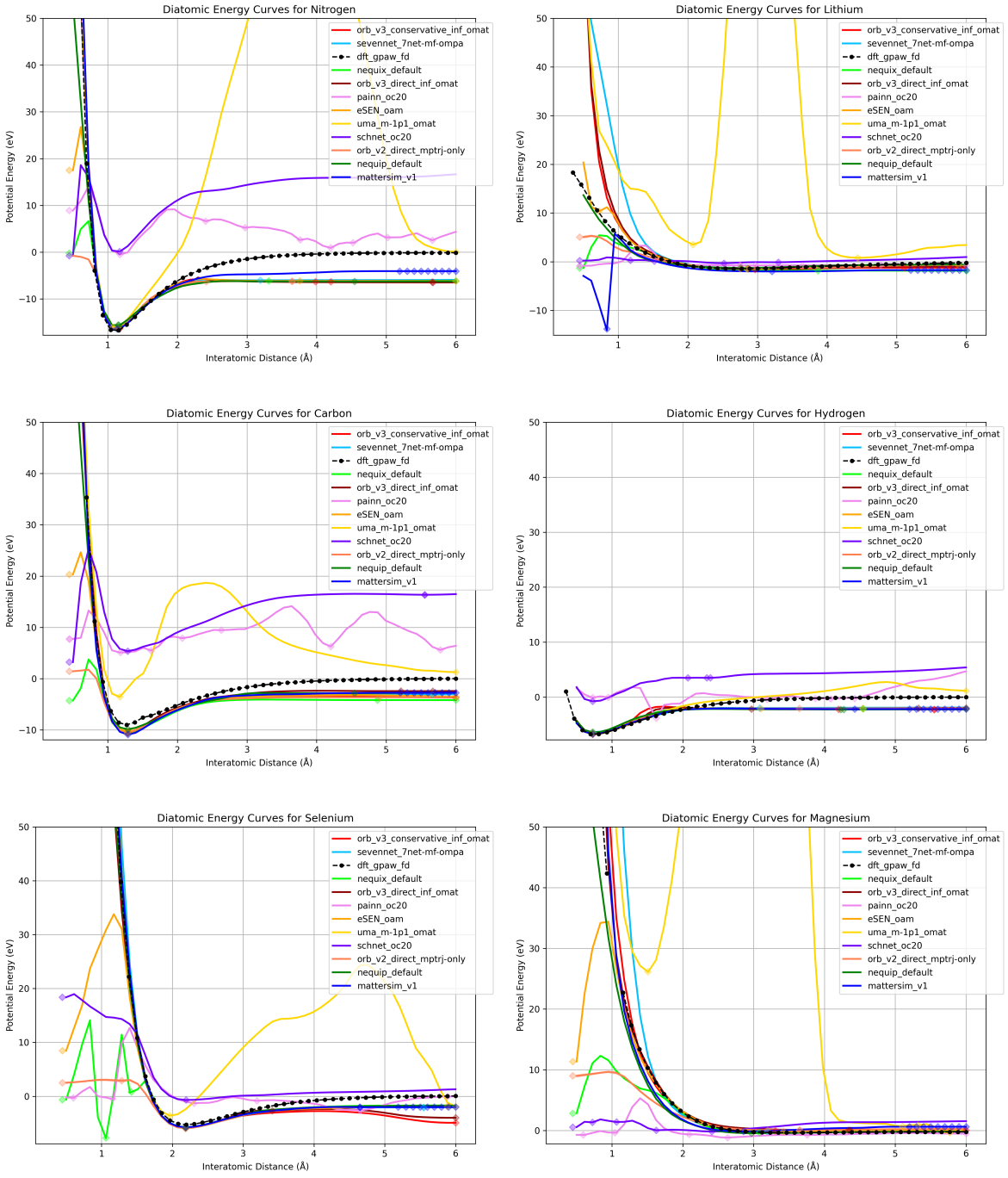

Figure 5.9.: Diatomic potential energy curves for different elements predicted by different models with DFT ab-initio references are shown in black. Diamond markers indicate equilibrium bond distances predicted by the models from a randomly sampled initial distance.

We now examine the diatomic potential energy curves described paragraph 4.3.2. All models exhibit almost perfect rotational invariance with barely visible confidence intervals, which is not too surprising considering the distance is the only relevant parameter for these systems and available to all models. There are three key observations to be made from Figure 5.9 and Figure 5.8, UMA's unusual behavior, atom locking and energy offsets.

Explaining UMA Surprisingly, the UMA model exhibits the largest DFT-diverging behavior for distances above 1 Å. This is *highly* unusual for a SOTA foundation model, and it is not clear why exactly this happens. The test has been repeated multiple times with different simulation-specific settings, but the results remain the same. One explanation could be that the model is simply overfitting on the training data, memorizing specific configurations instead of generalizing well to unseen configurations. This is likely, considering the exceptionally large model size of 1.5 billion parameters, compared to Nequix's 708k parameters [16] and Orb-v3's 25.5 million parameters [58]. Another possibility is that the model is explicitly trained to replicate relaxation trajectories, where the only relevant goal is to reach a local minimum as fast as possible with high determinism. In Lithium for example, the wall around $2.1 - 3.9$Å could serve as a decision boundary for the global minima at ≈ 2.645Å [132], where anything closer is strongly attracted and anything further away is strongly repelled. Another possible reason for this behavior is the fact that UMA was trained on datasets from different levels of theory, including ωB97M-V/def2-TZVPD [122, 123], PBE/PBE+U [23], RPBE [133] improving on revPBE [134] and PBE+D3 [27]. While ωB97M-V/def2-TZVPD is a hybrid[4] meta[5]-GGA functional with VV10 nonlocal correlation (vdW) [135], PBE is a "standard" GGA functional without any corrections for long-range interactions [23]. We refer the reader to section 2.2 for more information on why long-range corrections are necessary for DFT to model real-world systems accurately. This could confuse the model, especially when the raw energy is included in the training objective. These PES mispredictions by UMA have big implications for MD, as they introduce an exothermic barrier at the decision boundary, which prohibits atoms from bonding and separating, ever. In our initial plots we noticed a clear correlation between UMA diverging and GPAW Linear Combination of Atomic Orbitals (LCAO) not converging, unlike the results from GPAW FD method which we now use as reference in Figure 5.9. Future work could focus on investigating this behavior further.

Atom Locking In the pauli-repulsion regime, where quantum mechanical effects from the antisymmetry of the wavefunction results in an exceptionally strong repulsive force, Nequix [16], eSEN [62], SchNet [13] and PaiNN [14] predict either no repulsion at all or predict *attraction* instead, a behavior we call *atom locking*. Even though such distances are rather uncommon in MD simulations, they can occur in high-temperature simulations or when larger time steps Δt are used. This behavior likely emerges from the fact that there is not a lot of training data

[4] See subsection 2.2.2
[5] Depends on the kinetic energy density

available of atoms closer than ≈ 0.77 Å as seen in Figure 5.10 (left). If an atom is this close to another atom, it will either drift away quickly or it is being held there by other unrelated forces. When training data is obtained from DFT relaxations, the second case will naturally show up more often in the training data, leading the models in the best case to extrapolate (guessing) or in the worst case leading to believe an attractive force holding the atoms together.

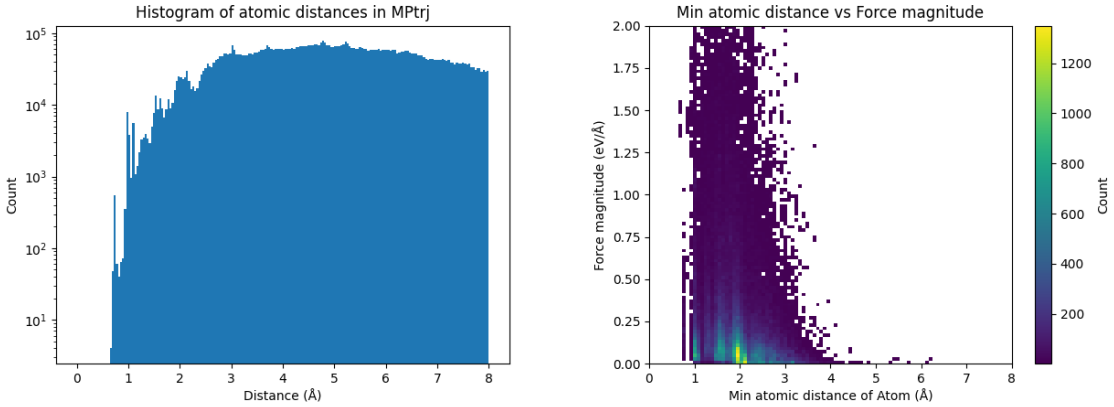

Figure 5.10.: MPTrj distance and force examination subsampled from 5000 structures of the relaxation-based MPtrj dataset [61, 136]. Distribution of pairwise atomic distances: short distances below 0.5 Å are uncommon (left). Minimum atom distance vs force magnitudes: the highest forces occur in the short-range regime (right). Forces were computed with the Orb-v3 model [58].

Comparing Figure 5.10 (left) with Figure 5.8: it becomes clear that the highest forces occur in the short-range regime below 0.5 Å, where the models are also the most uncertain about the energies. Datasets avoid this regime by using relaxation first and sampling later to avoid having exploding forces and energies, a mistake in our opinion. This could be mitigated by training on MD trajectories with various temperatures at larger time steps, which would limit how large the forces get, while still sampling a more diverse set of atomic distances, including the short-range regime.

Energy Offsets Most models tend to underestimate the potential energy in the limit $r \to 6$, with no model predicting $\lim_{r\to\infty} f_{\text{E-pot}}(r) = 0$ correctly. This behavior seems to be uncorrelated with the training dataset or model architecture used.

Element	H	He	Li	Be	B	C	O	...
Reference Energy	-6.0246	-4.9066	-5.2301	-4.9520	-5.1980	-6.1703	-8.0315	...
Asymptotic Energy	-1.9785	–	-0.9358	0.0393	-0.6206	-2.4184	-5.2408	...

Table 5.1.: Vienna Ab initio Simulation Package (VASP) shifted reference energies (eV) used by the Orb models [58] vs predicted asymptotic energies (eV) of the Orb-v3 conservative model.

Instead, this behavior likely emerges from a trick to enhance training stability: by shifting the predicted atomic energies by an element-specific constant energy reference [54, 16] one gets a near-zero mean energy target as training objective, improving training stability and convergence speed [29]. Rhodes, Vandenhaute, Šimkus, et al. [58] for example use the VASP reference energies "computed by running vasp optimisations on a single atom of each atom-type" [54, 137]. This behavior and possible solutions are subject to future work.

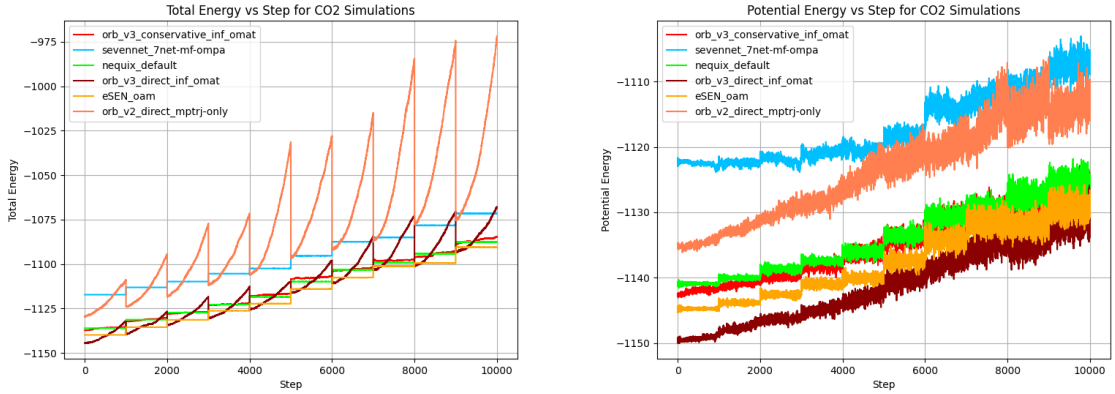

Figure 5.11.: Total energy (left) and potential energy (right) over time for different models.

5.1.4. Carbon Dioxide Stability

In Figure 5.11 we show the total and potential energy over time for different models over time during the CO_2 gas stability test. Orb-v2 and direct Orb-v3 exhibit significant energy drift over time, 241.52 meV/ps and 78.14 meV/ps on average, respectively. The argument can be made that this a consequence of the explicit architecture, and we agree to some extent. Learning a smooth energy surface is arguably easier than learning a smooth force surface, because it allows the model to focus on the relationship between atoms rather than figuring out how this relationship translates into forces, where it has to learn what a force *is* and how it rotates with the system, etc. But the conservative Orb-v3 model also experiences a slight energy drift of 9.42 meV/ps on average, despite being "conservative" by design.

Model	Drift (meV/ps)
Orb-v2	241.52
Orb-v3 Direct Inf	78.14
Orb-v3 Conservative Inf	9.42
SevenNet MF-ompa	0.30
Nequix Default	0.26
eSEN-30M-OAM	0.26

Table 5.2.: Energy drift in meV/ps for different models during the CO_2 gas stability test averaged over all temperatures.

In Figure 5.12 we show the MSD of CO_2 molecules over time for different models (left) and use that to compute the diffusion coefficients at different temperatures (right).

They show Orb-v2 and direct Orb-v3 exhibiting a diffusion coefficient twice and thrice as large as all other models. However, Donald R. Burgess [109] provides experimental diffusion

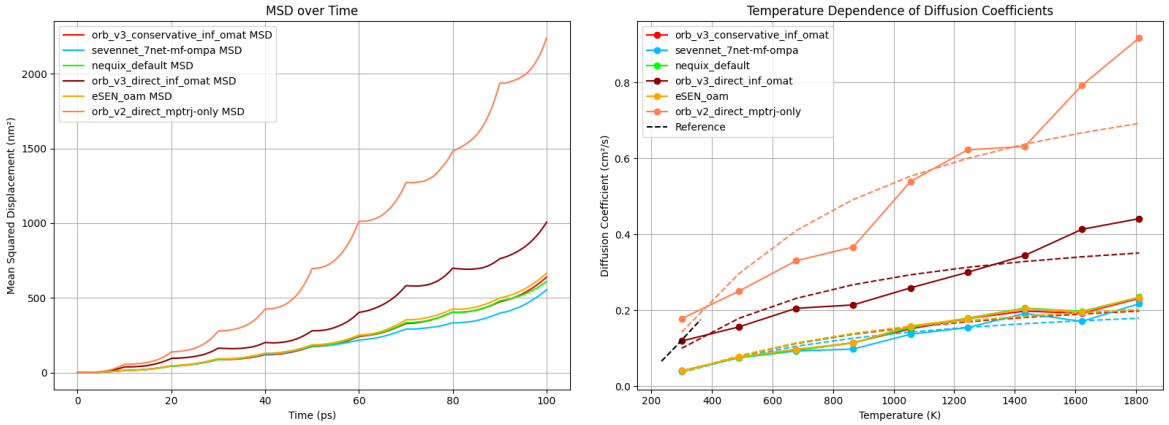

Figure 5.12.: MSD of CO_2 molecules over time for different models (left). Diffusion coefficients at different temperatures for different models (right).

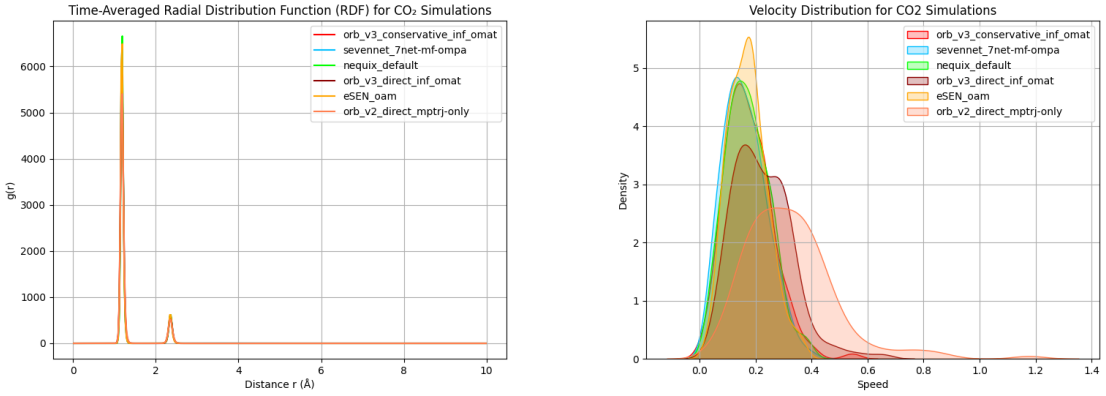

Figure 5.13.: Radial distribution function of atoms in the system (left). Velocity distribution of CO_2 molecules (right).

Model	$D_{\text{ref}}/cm^2s^{-1}$	A	B
Experimental [109]	0.120	0.015	-637.089
Orb-v2	0.178	0.0533	-568.4568
Orb-v3 Direct	0.191	-0.7975	-451.4263
Orb-v3 Conservative	0.004	-1.2809	-612.5149
eSEN-30M-OAM	0.004	-1.2570	-618.9303

Table 5.3.: Diffusion coefficients of CO_2 gas at 298K and 1 bar pressure

coefficients for CO_2 gas at 298K at a similar density to our test setup, which alights best with Orb-v2's prediction. The reason for this discrepancy is unclear, and could be coincidental. Orb-v2 suffers from significant energy drift naturally leading to an overestimation of the diffusion coefficient, as the system heats up over time, increasing the average kinetic energy of the atoms. It could be that the small system size and short simulation time could lead to an underestimation [138] and Orb-v2's energy drift just happens to counteract this effect. The results of the CO_2 gas stability test (paragraph 4.3.2) are shown in Figure 5.12, Figure 5.11 and Figure 5.13. We fit parameters A and B to

$$ln(D) = A + \frac{B}{T} \tag{5.1}$$

a simple Arrhenius-type equation [139] closely following work done by Donald R. Burgess [109] to obtain Table 5.3.

In this particular test, UMA would repeatedly crash from a low-level PyTorch GPU error related to dynamic shapes, which is why it is not included in the results here.

5.1.5. Inference Speed Results

Despite having nearly 100GB of GPU memory available to them, some models simply required more and failed to run. For CPU inference with about 200GB of available RAM, some models spilled to swap space, using the disk as temporary memory. This means not all datapoints are available, and some have simply been skipped for time assuming that existing datapoints make the scaling behavior apparent enough to allow extrapolation. In Figure 5.14 and Figure 5.15 we show the absolute and relative inference times for different models and system sizes as described in paragraph 4.3.2. All models exhibit **linear scaling** with respect to the system size, meaning that MLIPs have the same computational complexity class as empirical force fields. Simulating a system twice as large simply requires twice the resources or double the time, which is not the case for any ab-initio method and a major advantage of MLIPs. However, the fact that all models scale linearly on the CPU and GPU does not mean that they are equally fast in practice. Using a GPU for inference results in speedups two orders of magnitude, as long as the system is larger than about 1.000 atoms. Inference times for smaller systems are

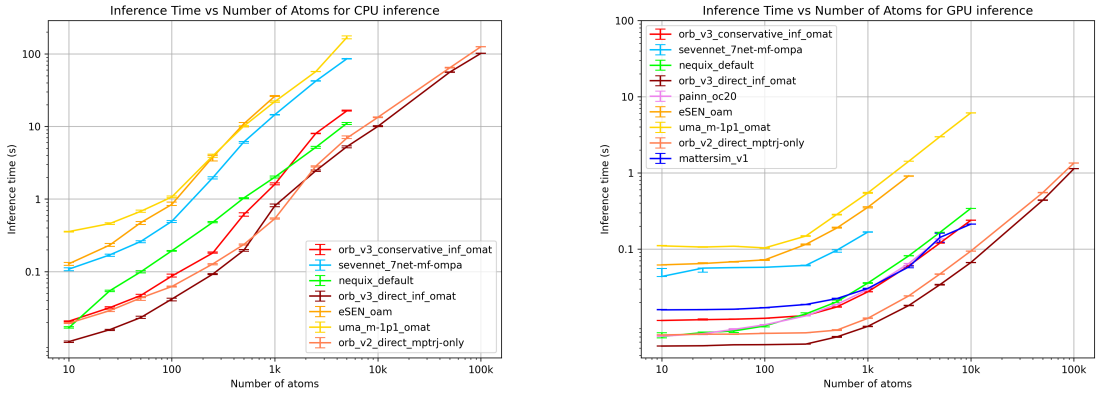

Figure 5.14.: Log-log plot of absolute model inference times for different system sizes on CPU (left) and GPU (right).

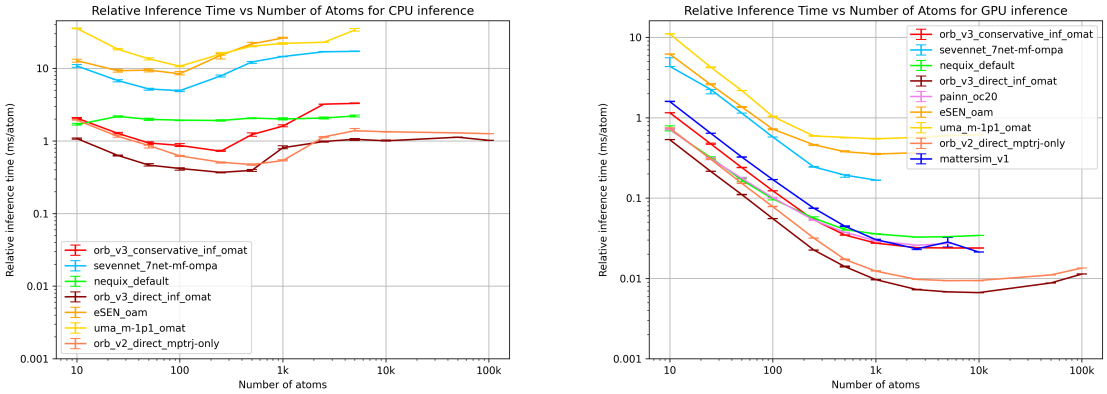

Figure 5.15.: Log-log plot of relative model inference times for different system sizes on CPU (left) and GPU (right).

bottlenecked by the general overhead and data transfer to and from the GPU. So for applications like HTS batching smaller systems to a single, clustered system can improve throughput by a large margin. Even though a single MD run can't be batched due to its sequential nature, independent parallel-running MD simulations of smaller systems can be batched together and benefit. But not only the GPU has a large effect on the linear factor: Orb-v3 Direct predicts forces of 100.000 atoms as fast as UMA can predict the forces of only about 2.000 atoms. But what are the reasons for these differences in inference speed for different models? To investigate this, we show some relevant model details in Table 5.4.

Model	Implicit	Equivariance	Framework	A.P.	N.L.	Cutoff
Orb-v3 Direct	×	×	PyTorch	25.5M	120	6 Å
Orb-v2	×	×	PyTorch	25.2M	20	6 Å
Nequix	✓	e3nn	JAX	708k	–	6 Å
PaiNN	✓	CFConv abl.	PyTorch	600k	–	4 Å
Orb-v3 Conservative	✓	×	PyTorch	25.5M	–	6 Å
MatterSim v1	✓	e3nn	PyTorch	4.5M	256	5 Å
SevenNet MF-ompa	✓	e3nn	PyTorch	25.2M	–	6 Å
eSEN OAM	✓	SO(2) conv	PyTorch	30.2M	–	6 Å
UMA M	✓	SO(2) conv	PyTorch	50.0M	–	6 Å

Table 5.4.: Performance-relevant model details, **sorted by inference speed** for 100 atoms on the GPU: The framework used to implement the model, whether the model is implicit or explicit, type of equivariant architecture used, the number of active parameters (A.P.), the neighbor limit (N.L.) and the cutoff radius used during inference.

Explicit models without an equivariant architecture like Orb-v2 and direct Orb-v3 run the fastest. Even with more parameters than Orb-v2 and a higher neighbor limit, Orb-v3 ends up being slightly faster due to reduced overhead of edge aggregation and message passing of 10 layers fewer [58]. On third place is Nequix, that is both implicit and uses the tensor products of e3nn [49] for equivariance, but has a relatively small number of parameters and uses JAX instead of PyTorch. This makes it about as fast as PaiNN, which uses a modifed CFConv architecture for equivariance as described in subsection 3.1.2. On 5th place is the conservative Orb-v3 model, the largest implicit model up to this point, which makes it significantly slower and memory-intensive than its direct-force counterpart. Next up are MatterSim v1 and SevenNet, with a large parameter count and expensive equivariant architecture. At the bottom are eSEN and UMA, both using SO(2) convolutions for equivariance, both with an implicit architecture, both with large parameter counts. While on paper SO(2) convolutions should be faster than e3nn's tensor products [49, 63], the fact that this operation is not vectorized [140] – and therefore ill-suited for GPU inference – makes it extremely slow in practice. Additionally, the rotation into the local reference frame of two atoms for each edge in the interaction graph

requires the computation of a custom Wigner D-matrix [32] for the Irreps used in the SO(2) convolutions, which adds additional computational overhead. UMA being slower than eSEN is most likely due to its higher parameter count and the overhead of the MoLE gating and linear expert merge [15]. In Table 5.6 we converted the raw inference speed into a more intuitive unit of nanoseconds per day (ns/day), assuming that a simulation is run with a time step of 1 fs[6]. This is a more common unit to express simulation speeds in the MD community, as it intuitively relates to how much simulation time can be covered in a day of wall-clock time.

Model	10 atoms	100 atoms (↓)	1,000 atoms	10,000 atoms	100,000 atoms
Orb-v3 Direct	**16.14**	**15.55**	**8.95**	**1.29**	**0.08**
Orb-v2	11.57	11.04	6.79	0.92	0.06
Nequix	12.00	8.91	2.41	0.25	–
PaiNN	11.98	8.56	2.94	–	–
Orb-v3 Conservative	7.46	7.00	3.15	0.36	–
MatterSim v1	5.40	5.10	2.84	0.41	–
SevenNet MF-ompa	1.82	1.51	0.51	–	–
eSEN OAM	1.40	1.19	0.24	–	–
UMA M	0.78	0.84	0.16	0.01	–

Table 5.5.: Inference speeds in ns/day for different models and system sizes on a single H100 GPU, sorted by inference speed for 100 atoms. Almost all models ran out of memory for 100,000 atoms.

Model	10 atoms	100 atoms (↓)	1,000 atoms	10,000 atoms	100,000 atoms
Orb-v3 Direct	**8.05**	**2.09**	0.11	**0.01**	0.00
Orb-v2	4.42	1.38	**0.16**	0.01	0.00
Orb-v3 Conservative	4.15	0.99	0.05	0.00	–
Nequix	5.06	0.45	0.04	0.00	–
SevenNet	0.39	0.06	0.00	–	–
UMA M	0.24	0.08	0.00	–	–

Table 5.6.: Inference speeds in ns/day for different models and system sizes on the CPU. To save time, not all models were run for the largest system sizes.

[6] 10^{-15} s

5.2. MOF Experiments

We now turn to the MOF-specific experiments, the more practical MLIP analysis.

5.2.1. Widom Insertion Results

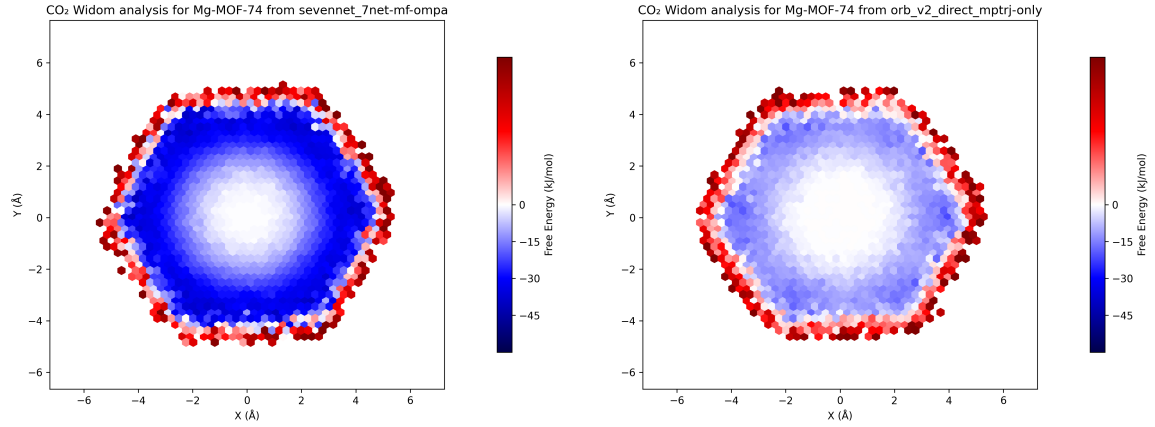

Figure 5.16.: Hexbin plot of CO_2 insertion energies predicted by the SevenNet (left) and Orb-v2 (right) model.

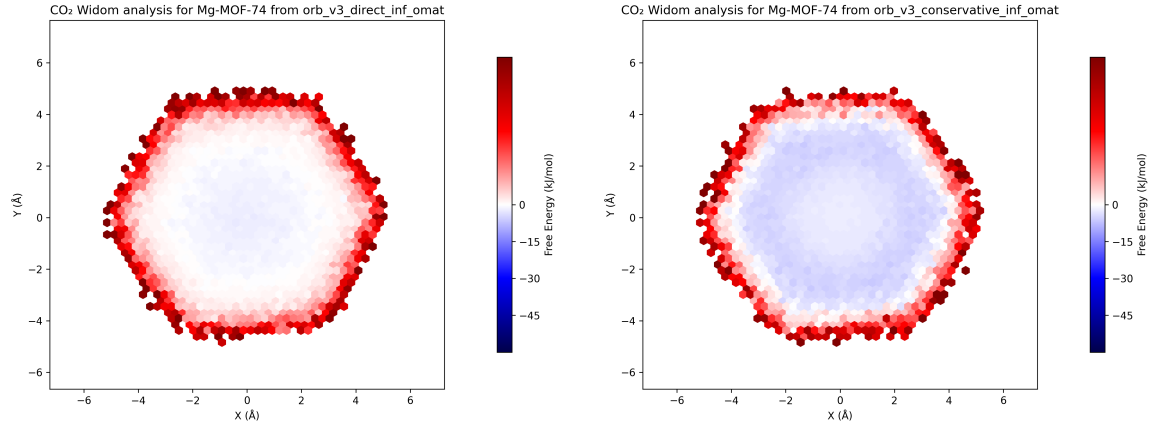

Figure 5.17.: Hexbin plots of CO_2 insertion energies predicted by the Orb-v3 direct-force on (left) and conservative (right) model.

The plots above show the Widom insertion energy calculations described in subsection 4.4.1, where a CO_2 molecule is inserted at random positions inside the Mg-MOF-74 structure. All models predict a red hexagonal boundary on the inner side of the MOF structure, a small region where the pauli-repulsion forces start to dominate [18]. For improved visibility and relevance tiles with energies above 5 eV were omitted from the plots. Ideally, we should observe

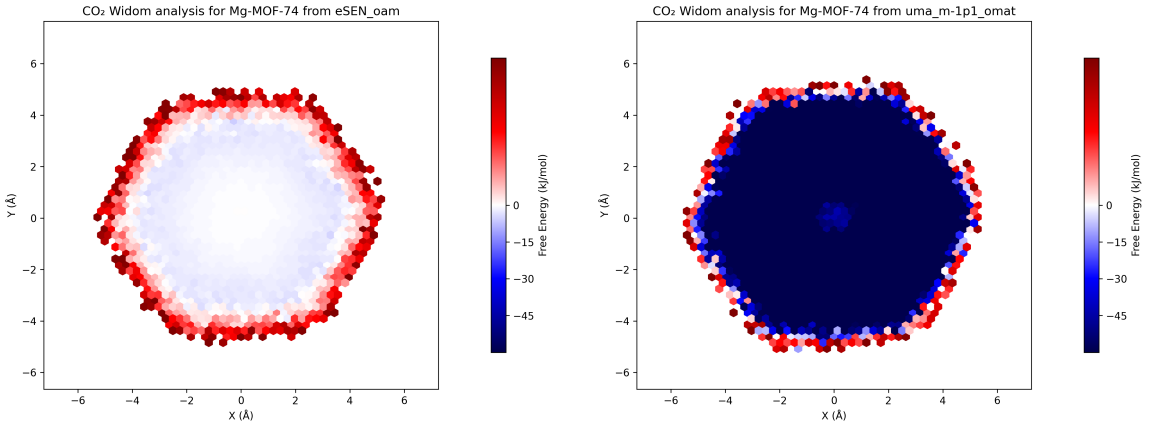

Figure 5.18.: Hexbin plots of CO_2 insertion energies predicted by the eSEN (left) and UMA (right) model.

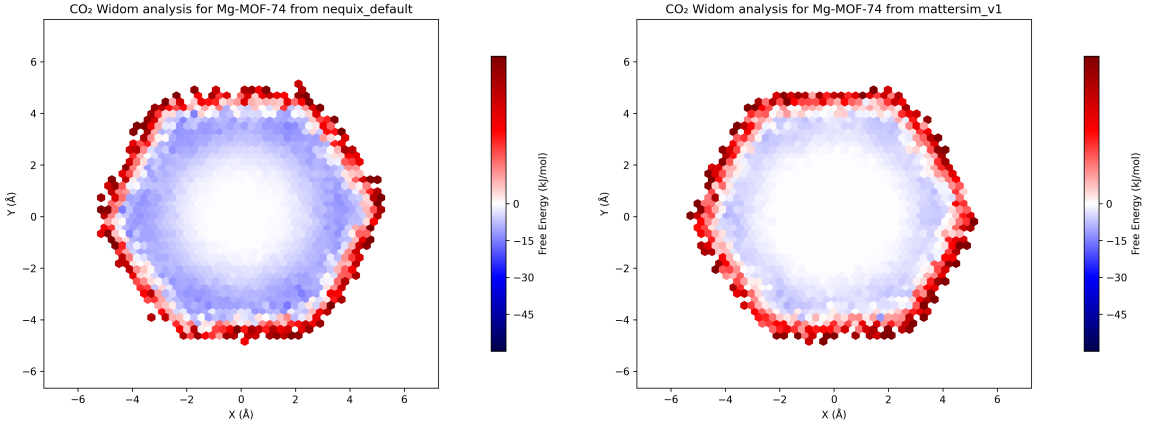

Figure 5.19.: Hexbin plots of CO_2 insertion energies predicted by the Nequix (left) and MatterSim v1 (right) model.

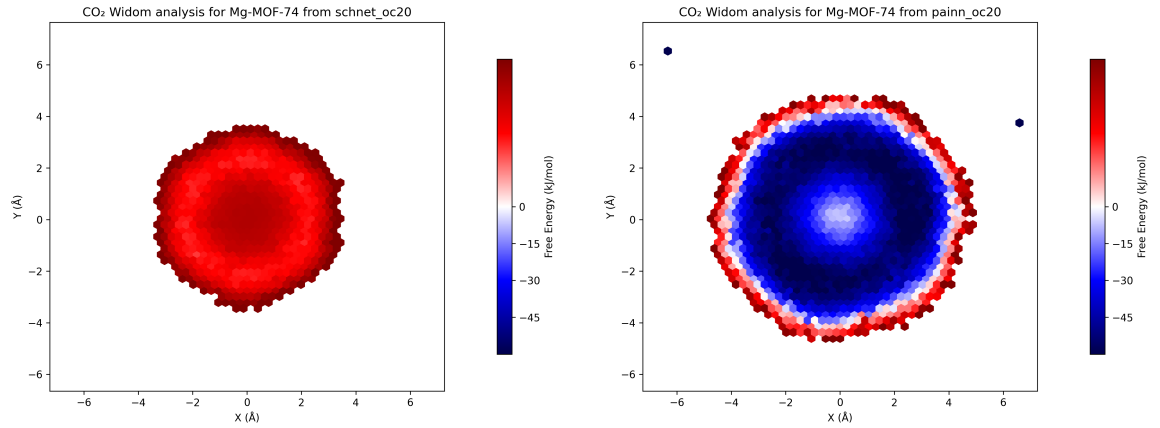

Figure 5.20.: Hexbin plot of CO_2 insertion energies predicted by the SchNet (left) and PaiNN (right) model.

"pockets" of favorable insertion energies (blue regions) near the magnesium ions at the corners of the MOF structure [3]. This is however not really the case, and the results seem to vary significantly between different models; and since running DFT is not feasible for this system and required number of insertions, a comparison with a higher level of theory is out of reach.

UMA overestimates the insertion energies across the entire structure *significantly*, similar to the behavior observed in the diatomic tests (Refer to paragraph 5.1.3). Orb-v2 seems to perform significantly better than all succeeding Orb-v3 models, predicting favorable insertion sites close to the magnesium ions. eSEN and Orb-v3 conservative predict almost no favorable insertion sites at all, while Orb-v3 direct-force predicts some favorable sites, but not as pronounced as Orb-v2. PaiNN produces a donut-like shape of favorable insertion energies around the center, predicting favorable sites significantly closer to the center of the MOF structure than all other models. Both Orb-v3 model variations perform D3 dispersion corrections after inference, which does not seem to mitigate the problem [27, 26].

5.2.2. Adsorption Results

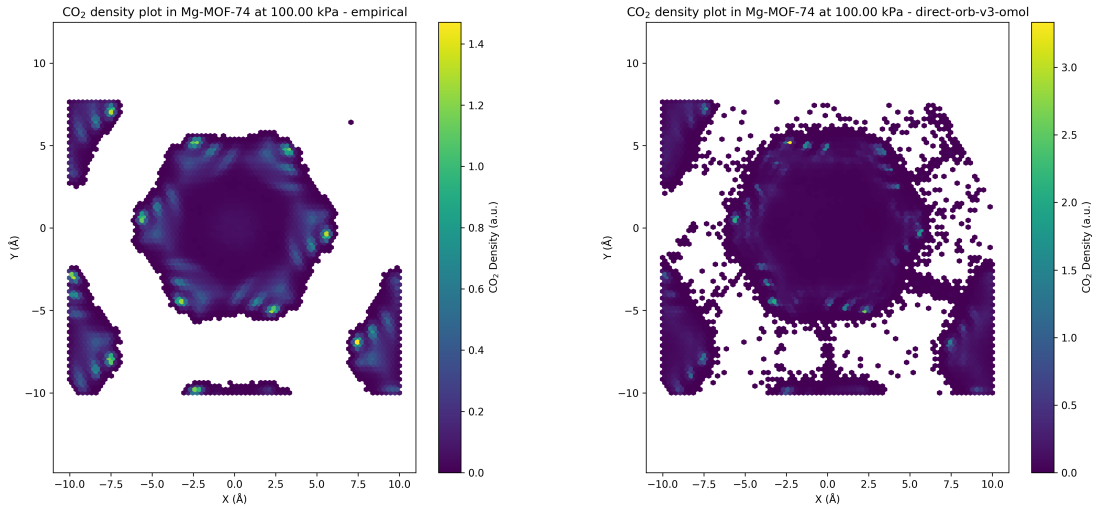

Figure 5.21.: CO_2 atom density plots in Mg-MOF-74 at 298K at 1 bar pressure: Recreated empirical results (left) [3] and Orb-v3 direct-force results (right) [58]. Results are obtained from a 1:100 subsample of the GCMC simulation run.

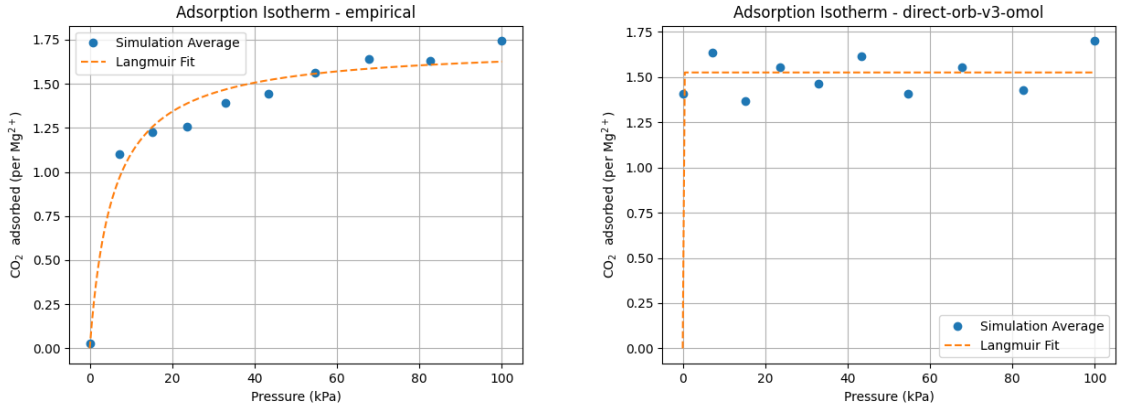

Figure 5.22.: CO_2 adsorption isotherms in Mg-MOF-74 at 298K: Recreated empirical results (left) [3] and Orb-v3 direct-force omol model results (right) [58].

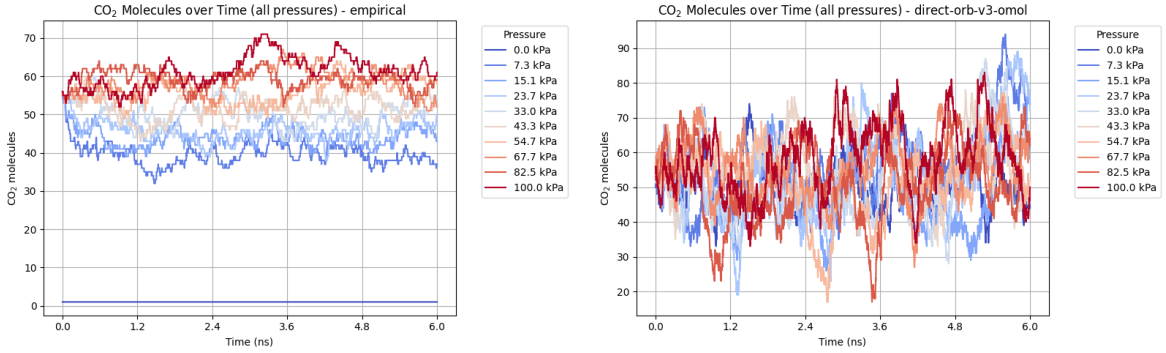

Figure 5.23.: Number of adsorbed CO_2 molecules over time for different pressure settings of the GCMC simulation in Mg-MOF-74 at 298K: Recreated empirical results (left) [3] and Orb-v3 direct-force omol model results (right) [58].

While the Orb-v3 direct-force omol model can reproduce some of the results from Y. Fu, Y. Yao, Forse, et al. [3], it fails to capture the essence of CO_2 adsorption in Mg-MOF-74 correctly. In Figure 5.21 we show the density plots of the CO_2 atoms inside the MOF structure, where we can see the wave-like patterns originating from the magnesium ions at the corners of the MOF predicted by the empirical model (left), with a slight resemblance of this pattern in the Orb-v3 direct-force results (right). Additionally, the GCMC algorithm sometimes inserts CO_2 molecules directly into the MOF structure itself for some unknown reason, which might have a negative effect on the results generally. In Figure 5.22 we show the adsorption isotherms predicted by the empirical model (left) and the Orb-v3 direct-force omol model (right), how many CO_2 molecules stay in the GCMC simulation at different pressures. While the empirical model predicts a logarithmic increase in adsorbed CO_2 molecules with increasing pressure, the Orb-v3 direct-force omol model predicts a constant number of adsorbed CO_2 molecules for all pressure settings. We fit the Langmuir adsorption model [141]

$$q = \frac{q_{\max} b P}{1 + bP} \tag{5.2}$$

to both isotherms, where q is the amount adsorbed at pressure P, $q_{\max}$ is the maximum adsorption capacity, and b is a constant related to the affinity of the adsorbent for the adsorbate. This model is plotted as a dashed line in Figure 5.22, showing a good fit for the empirical results while the Orb-v3 direct-force omol model predicts almost no pressure-related correlation for CO_2 adsorption. Finally, we plot the number of CO_2 molecules inside the GCMC simulation over time for different pressure settings in Figure 5.23. While the empirical model shows a small variation in the number of CO_2 molecules over unique pressure-dependent mean values over time, the results obtained from the Orb-v3 direct-force omol model seem random with a high variance without any pressure-dependent mean value. A potential area of future work could be the evaluation – with the necessary computational resources at hand – of more models or orb-v3 variations to find better performing models for this task.

6. Discussion and Conclusion

6.1. Summary of Work

In this work we explored how MLIPs work and how they are built, different architectures and design choices that go into making them, and how they overcome the challenges of modeling the PES. Through a series of experiments, we evaluated various SOTA MLIP models, analyzing their strengths and weaknesses. We were successful in building a universal ASE-MLIP interface for LAMMPS, allowing us to run an originally empirical Mg-MOF-74 CO_2-adsorption GCMC simulation with a MLIP instead, only to discover that it doesn't reproduce the more physically-sound empirical results.

6.2. Interpreting the Results

MLIPs are fast Comparing the speed of MLIPs with ab-initio methods is a day and night difference. Even with significant variance between different models, all MLIPs scale linearly with system size and are – unlike ab-initio methods – indifferent to atomic species. This makes them exceptionally practical for MD simulations of almost any size, and can theoretically scale from a single machine to supercomputers with ease.

Advertised benchmarks can be misleading Our results demonstrate that benchmarks commonly used to evaluate MLIPs, including phonon-derived properties with by extension κ_{SRME}, are often insufficient, incomplete or misleading evaluation metrics, as they rely on force predictions in very specific equilibrium regimes of the PES. Our results show that some models with exceptionally poor PES accuracy were still able to excel in phonon benchmarks – the primary evaluation metric in common literature. The scientific field of MLIPs exhibits a pressing need of improved evaluation metrics that more reliably capture a model's ability to replicate the PES.

Every model has its weakness No MLIP performs well in every benchmark. Some models induce deal-breaking energy drifts, others show severe instabilities in the pauli-repulsion regime, while yet others seem to prohibit bond braking. And some models outright refuse to run specific standard benchmarks by repeatedly crashing mid-simulation for an unknown low-level CUDA error, no matter how many magic incantations we tried to appease them with. So any MLIP can – occasionally – stop working at any time for any reason, a dealbreaker again.

6.3. Implications and Significance

MLIPs should be closely monitored if they ever find their way into practical applications. Even though they do show impressive speedups over ab-initio methods, their maturity for practical applications is still questionable. They are far from the plug-and-play solution they may advertise themselves as, and should be treated with a healthy dose of skepticism.

6.4. Limitations

While we were able to evaluate the cheapest MLIP model on the CO_2 GCMC simulation, the more accurate models were simply too expensive to run in a reasonable time frame. This prevented us from evaluating more models the GCMC simulation, a feat that might take weeks or months of H100 computing time. Additionally, due to time, space and complexity constraints, a significant portion of the experiments conducted had to be omitted from this work.

6.5. Future Work

Synthetic Datasets The current state of MLIP datasets also leaves much to be desired. Instead of using expensive ab-initio methods to generate DFT or CC datasets, we believe that synthetic datasets could be a better alternative for the development and evaluation of MLIP architectures. Generated from any combination of fictitious n-body potentials – freely adjustable in difficulty and complexity, synthetic datasets could ease the journey into real, messier datasets. The currently largest DFT dataset has about 500 million systems, a few dozen laptops could generate an equivalent amount of synthetic data in a week. This would allow not only for better control of the data distribution and coverage, but would make benchmarking and iteration cycles significantly faster.

Hybrid-MLIPs A promising direction to improve MLIPs could be the addition of classical potentials with learnable parameters as prior into the architecture, where easier parts of the PES like pauli-repulsion are modeled by a learnable empirical potential and the more difficult parts like electrostatics are then left to the MLP part of the MLIP to capture. This could make training easier and faster, improve physical consistency in Out-Of-Distribution (OOD) regimes and simultaneously reduce the amount of training data and parameters required.

6.6. Final Remarks

MLIPs for all their impressive technical sophistication feel primarily developed for academic research rather than practical use. They offer no guarantees and show no physical consistency in OOD regimes, a dealbreaker for many practical applications. They are extremely impressive feats of engineering and will undoubtedly revolutionize molecular simulations, but not just yet.

A. General Addenda

```python
import torch
from e3nn import o3

# Define irreps
scalar = o3.Irrep("0e")
vector = o3.Irrep("1o")
tensor_product = o3.FullTensorProduct(scalar, vector)

# Define a rotation (90 degrees around y-axis)
R = o3.angles_to_matrix(torch.tensor([0.0]), torch.tensor([torch.pi/2]), torch.tensor([0.0]))
D_R = tensor_product.irreps_out.D_from_matrix(R)

# Setup messages
a = torch.tensor([2.0])
b = torch.tensor([1.0, 0.0, 0.0])
message_rotated_before = tensor_product(a, R @ b)
message_rotated_after = D_R @ tensor_product(a, b)

# Check equivariance
print(torch.allclose(message_rotated_before, message_rotated_after, atol=1e-4))
# >> True
```

Listing A.1: Example code showing the equivariance of a scalar-vector tensor product using the `e3nn` [49] library.

```python
import numpy as np
from scipy.sparse import diags
import matplotlib.pyplot as plt

N = 500
r, dr = np.linspace(1e-5, 20.0, N, retstep=True)
l = 0

T_diagonals = [-2.0 * np.ones(N), np.ones(N - 1), np.ones(N - 1)]
T = -0.5 / dr**2 * diags(T_diagonals, [0, 1, -1]).toarray()
L = np.diag(l*(l+1) / (2 * r**2))
V = np.diag(-1 / r)
H = T + L + V

E, psi = np.linalg.eigh(H)
plt.plot(r, np.abs(psi[:,3])**2)
plt.xlabel('r')
plt.ylabel(r'Probability Density')
plt.show()
```

Listing A.2: Diagonalizing the Hessian matrix of a hydrogen wave function using finite differences.

```
# Use simpler units
units lj
atom_style atomic
# Periodic in all directions
boundary p p p

# 10x10x10 box
region box block 0 10 0 10 0 10
create_box 1 box
# Create 1000 atoms randomly inside the box (with seed 12345)
# We don't define the atom type, and name the atoms '1'
```

```
create_atoms 1 random 1000 12345 box

# Define interatomic potential being used
pair_style lj/cut 2.5
# Refine interaction parameters
pair_coeff 1 1 1.0 1.0 2.5

# Atoms '1' have mass 1.0
mass 1 1.0
# Give atoms an initial random velocity
# sampled from a Maxwell distribution at temperature 1.0 (with seed 12345)
velocity all create 1.0 12345

# Run simulation for 1000 timesteps
fix 1 all nve
# Output thermodynamic data every 100 timesteps
thermo 100
run 1000
```

Listing A.3: Example LAMMPS input script for a simple molecular dynamics simulation.

```
class MyMLIPCalculator(Calculator):
  implemented_properties = ['energy', 'forces']

  def __init__(self, model_path, **kwargs):
    Calculator.__init__(**kwargs)
    self.results = {}
    self.model = MLIP(model_path)

  def calculate(self, atoms=None, properties=None, system_changes=all_changes):
    if atoms is None:
      atoms = self.atoms
    self.atoms = atoms.clone()

    positions = atoms.get_positions()
    cell = atoms.get_cell()
    species = atoms.get_atomic_numbers()

    energy, forces = self.model.predict(positions, cell, species)

    self.results = {}
    if 'energy' in properties:
      self.results['energy'] = energy
    if 'forces' in properties:
      self.results['forces'] = forces
```

Listing A.4: Example calculator class integrating a MLIP model into ASE.

```
cdef extern from "mliap_data.h" namespace "LAMMPS_NS":
    cdef cppclass MLIAPData:
        int size_gradforce
        int ndescriptors # number of descriptors
        int nparams # number of model parameters per element
        int nelements # number of elements
        int ntotal # total number of owned and ghost atoms on this proc
        int nlistatoms # current number of non-NULL atoms in local atom lists
        int nlocal # current number of NULL and normal atoms in local atom lists
        int natomneigh # current number of atoms and ghosts in atom neighbor arrays
        int * numneighs # neighbors count for each atom
        int * iatoms # index of each atom
        int * pair_i # index of each i atom for each ij pair
        int * ielems # element of each atom
        int nneigh_max # number of ij neighbors allocated
        int npairs # number of ij neighbor pairs
        int * jatoms # index of each neighbor
        int * jelems # element of each neighbor
        int * elems # element of each atom in or not in the neighborlist
        double ** rij # distance vector of each neighbor
        int eflag # indicates if energy is needed
        int vflag # indicates if virial is needed

# Source: shortened from https://github.com/lammps/lammps/blob/1db2e937639713cc33211787975e29f139039c87/src/ML-IAP/mliap_unified_couple.pyx#
    L19
```

Listing A.5: The LAMMPS ML-IAP interace data object does not provide atomic positions, only distance vectors. Write-only and optional fields are excluded.

```
Subject: [PATCH] Enhance atom access for the Python interface
---
Index: src/ML-IAP/mliap_data.cpp
<+>UTF-8
===================================================================
diff --git a/src/ML-IAP/mliap_data.cpp b/src/ML-IAP/mliap_data.cpp
--- a/src/ML-IAP/mliap_data.cpp (revision 1db2e937639713cc33211787975e29f139039c87)
+++ b/src/ML-IAP/mliap_data.cpp (revision 3e19d3791016ac3222ec17c60fec3f48fd24310b)
@@ -18,6 +18,7 @@
 #include "mliap_data.h"

 #include "atom.h"
+#include "domain.h"
 #include "error.h"
 #include "memory.h"
 #include "mliap_descriptor.h"
@@ -29,11 +30,11 @@
 MLIAPData::MLIAPData(LAMMPS *lmp, int gradgradflag_in, int *map_in, class MLIAPModel *model_in,
                      class MLIAPDescriptor *descriptor_in, class PairMLIAP *pairmliap_in) :
     Pointers(lmp),
-    f(nullptr), gradforce(nullptr), betas(nullptr), descriptors(nullptr), eatoms(nullptr),
-    gamma(nullptr), gamma_row_index(nullptr), gamma_col_index(nullptr), egradient(nullptr),
-    numneighs(nullptr), iatoms(nullptr), ielems(nullptr), itypes(nullptr), pair_i(nullptr),
+    x(nullptr), f(nullptr), periodicity(nullptr), cell(nullptr), gradforce(nullptr), betas(nullptr),
+    descriptors(nullptr), eatoms(nullptr), gamma(nullptr), gamma_row_index(nullptr), gamma_col_index(nullptr),
+    egradient(nullptr), numneighs(nullptr), iatoms(nullptr), ielems(nullptr), itypes(nullptr), pair_i(nullptr),
     jatoms(nullptr), jelems(nullptr), elems(nullptr), lmp_firstneigh(nullptr), rij(nullptr),
-    graddesc(nullptr), model(nullptr), descriptor(nullptr), list(nullptr)
+    graddesc(nullptr), model(nullptr), descriptor(nullptr), list(nullptr), q(nullptr), magmoms(nullptr)
 {
   gradgradflag = gradgradflag_in;
   map = map_in;
@@ -60,6 +61,7 @@
   size_array_rows = 1 + ndims_force * natoms + ndims_virial;
   size_array_cols = nparams * nelements + 1;
   size_gradforce = ndims_force * nparams * nelements;
+  nneigh_atom_max = 0;

   nlistatoms_max = 0;
   natomneigh_max = 0;
@@ -108,10 +110,47 @@
 void MLIAPData::generate_neighdata(NeighList *list_in, int eflag_in, int vflag_in)
 {
   list = list_in;
-  f = atom->f;
-  double **x = atom->x;
+  x = atom->x;
+  double** x = atom->x;
+  f = atom->f;
   int *type = atom->type;

+  // pass charge and spin arrays if they exist
+  if (atom->q_flag && atom->q != nullptr) {
+    q = atom->q;
+  } else {
+    q = nullptr;
+  }
+  if (atom->mu_flag && atom->mu != nullptr) {
+    magmoms = atom->mu;
+  } else {
+    magmoms = nullptr;
+  }
+
+  periodicity = domain->periodicity;
+  memory->destroy(cell);
+  memory->create(cell, 3, 3, "MLIAPData:cell");
+  if (domain->triclinic) {
+    if (domain->triclinic_general) {
+      for (int i = 0; i < 3; ++i) {
+        cell[i][0] = domain->avec[i];
+        cell[i][1] = domain->bvec[i];
```

```
+        cell[i][2] = domain->cvec[i];
+      }
+    } else {
+      cell[0][0] = domain->xprd; cell[0][1] = domain->xy;    cell[0][2] = domain->xz;
+      cell[1][0] = 0.0;          cell[1][1] = domain->yprd;  cell[1][2] = domain->yz;
+      cell[2][0] = 0.0;          cell[2][1] = 0.0;           cell[2][2] = domain->zprd;
+    }
+  } else {
+    cell[0][0] = domain->xprd;
+    cell[1][1] = domain->yprd;
+    cell[2][2] = domain->zprd;
+  }
+  for (int i = 0; i < 3; i++)
+    for (int j = 0; j < 3; j++)
+      if (std::abs(cell[i][j]) < 1.0e-15) cell[i][j] = 0.0;
+
   int *ilist = list->ilist;
   int *numneigh = list->numneigh;
   int **firstneigh = list->firstneigh;
@@ -186,13 +225,24 @@
       const int jelem = map[jtype];

       if (rsq < descriptor->cutsq[ielem][jelem]) {
         pair_i[ij] = i;
         jatoms[ij] = j;
         jelems[ij] = jelem;
         rij[ij][0] = delx;
         rij[ij][1] = dely;
         rij[ij][2] = delz;
+        // this made problems somehow, but is no longer needed anyway
-        lmp_firstneigh[ii][ninside] = firstneigh[i][jj];
         ij++;
         ninside++;
       }
@@ -216,7 +266,6 @@
 /* ----------------------------------------------------------------------
    grow neighbor arrays to handle all neighbors
 ------------------------------------------------------------------------- */
-
 void MLIAPData::grow_neigharrays()
 {

@@ -228,7 +277,6 @@
     memory->grow(ielems, natomneigh, "MLIAPData:ielems");
     memory->grow(itypes, natomneigh, "MLIAPData:itypes");
     memory->grow(numneighs, natomneigh, "MLIAPData:numneighs");
-    memory->grow(lmp_firstneigh, natomneigh, nneigh_max, "MLIAPData:lmp_firstneigh");
     natomneigh_max = natomneigh;
   }

@@ -237,10 +285,11 @@
   int *ilist = list->ilist;
   int *numneigh = list->numneigh;
   int **firstneigh = list->firstneigh;
-  double **x = atom->x;
+  double** x = atom->x;
   int *type = atom->type;

   int nneigh = 0;
+  int max_ninside_this_step = 0;
   for (int ii = 0; ii < natomneigh; ii++) {
     const int i = ilist[ii];

@@ -268,10 +317,14 @@
     nneigh += ninside;
   }

+  if (nneigh_atom_max < nneigh || lmp_firstneigh == nullptr) {
+    nneigh_atom_max = nneigh;
+    memory->grow(lmp_firstneigh, natomneigh, nneigh, "MLIAPData:lmp_firstneigh");
+  }
+
   if (nneigh_max < nneigh) {
     memory->grow(pair_i, nneigh, "MLIAPData:pair_i");
     memory->grow(jatoms, nneigh, "MLIAPData:jatoms");
-    memory->grow(lmp_firstneigh, natomneigh, nneigh, "MLIAPData:lmp_firstneigh");
     memory->grow(jelems, nneigh, "MLIAPData:jelems");
     memory->grow(rij, nneigh, 3, "MLIAPData:rij");
```

```
     if (gradgradflag == 0) memory->grow(graddesc, nneigh, ndescriptors, 3, "MLIAPData:graddesc");
Index: src/ML-IAP/mliap_data.h
<+>UTF-8
===================================================================
diff --git a/src/ML-IAP/mliap_data.h b/src/ML-IAP/mliap_data.h
--- a/src/ML-IAP/mliap_data.h (revision 1db2e937639713cc33211787975e29f139039c87)
+++ b/src/ML-IAP/mliap_data.h (revision 4eae8dc7cabb6c9086c2ef2b9fa07970e9764a08)
@@ -30,15 +30,21 @@
   virtual void grow_neigharrays();
   double memory_usage();

+  int nneigh_atom_max;
   int size_array_rows, size_array_cols;
   int natoms;
   int size_gradforce;
   int yoffset, zoffset;
   int ndims_force, ndims_virial;
+  double **x;
+  int* periodicity;        // box periodicity in each dimension
+  double **cell;           // 3x3 cell vectors
   double **f;
   double **gradforce;
   double **betas;          // betas for all atoms in list
   double **descriptors;    // descriptors for all atoms in list
+  double **magmoms;        // magnetic moments for all atoms in list
+  double *q;               // charges for all atoms in list
   double *eatoms;          // energies for all atoms in list
   double energy;           // energy
   int ndescriptors;        // number of descriptors
Index: src/ML-IAP/mliap_unified_couple.pyx
<+>UTF-8
===================================================================
diff --git a/src/ML-IAP/mliap_unified_couple.pyx b/src/ML-IAP/mliap_unified_couple.pyx
--- a/src/ML-IAP/mliap_unified_couple.pyx (revision 1db2e937639713cc33211787975e29f139039c87)
+++ b/src/ML-IAP/mliap_unified_couple.pyx (revision b81515f96caabf1a09d01c1443c095e559bfbacc)
@@ -28,6 +28,11 @@
         int ndims_virial
         # -END- may not need -END-
         int size_gradforce
+        int* periodicity;       # box periodicity in each dimension
+        double **cell;          # 3x3 cell vectors
+        double ** x             # atom coordinates
+        double * q              # atom charges
+        double ** magmoms       # atom magnetic moments
         # ----- write only -----
         double ** f
         double ** gradforce
@@ -124,12 +129,42 @@
         cdef double[:, ::1] fij_arr = fij
         update_pair_forces(self.data, &fij_arr[0][0])

+    @property
+    def periodicity(self):
+        if self.data.periodicity is NULL:
+            return None
+        return np.asarray(<int[:3]> &self.data.periodicity[0])
+
+    @property
+    def cell(self):
+        if self.data.cell is NULL:
+            return None
+        return np.asarray(<double[:3, :3]> &self.data.cell[0][0])
+
     @property
     def f(self):
         if self.data.f is NULL:
             return None
         return np.asarray(<double[:self.ntotal, :3]> &self.data.f[0][0])

+    @property
+    def x(self):
+        if self.data.x is NULL:
+            return None
+        return np.asarray(<double[:self.ntotal, :3]> &self.data.x[0][0])
+
+    @property
+    def charges(self):
+        if self.data.q is NULL:
```

```
+            return None
+        return np.asarray(<double[:self.ntotal]> &self.data.q[0])
+
+    @property
+    def magmoms(self):
+        if self.data.magmoms is NULL:
+            return None
+        return np.asarray(<double[:self.ntotal, :3]> &self.data.magmoms[0][0])
+
     @property
     def size_gradforce(self):
         return self.data.size_gradforce
```

Listing A.6: Patch for the LAMMPS ML-IAP interface to provide MLIPs with the necessary atomic positions, cell vectors and periodicity information of the system.

```python
import numpy as np
from ase import Atoms
from ase.calculators.calculator import Calculator
from lammps.mliap.mliap_unified_abc import MLIAPUnified

class UniversalASEUnifiedMLIAPInterface(MLIAPUnified):
  elements = [
    "H", "He", "Li", "Be", "B", "C", "N", "O", "F", "Ne", "Na", "Mg", "Al", "Si", "P", "S", "Cl", "Ar",
    "K", "Ca", "Sc", "Ti", "V", "Cr", "Mn", "Fe", "Co", "Ni", "Cu", "Zn", "Ga", "Ge", "As", "Se", "Br", "Kr",
    "Rb", "Sr", "Y", "Zr", "Nb", "Mo", "Tc", "Ru", "Rh", "Pd", "Ag", "Cd", "In", "Sn", "Sb", "Te", "I", "Xe",
    "Cs", "Ba", "La", "Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er", "Tm", "Yb", "Lu",
    "Hf", "Ta", "W", "Re", "Os", "Ir", "Pt", "Au", "Hg", "Tl", "Pb", "Bi", "Po", "At", "Rn", "Fr", "Ra",
    "Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk", "Cf", "Es", "Fm", "Md", "No", "Lr", "Rf", "Db",
    "Sg", "Bh", "Hs", "Mt", "Ds", "Rg", "Cn", "Fl", "Lv", "Ts", "Og"
  ]
  def __init__(
    self, ase_calculator: Calculator, rcutfac=1,
    lammps_units: str = "not-set"
  ):
    super().__init__(None, self.elements, 1, 3, rcutfac)
    self.ase_calculator = ase_calculator
    if 'forces' not in ase_calculator.implemented_properties:
      raise ValueError("The provided ASE calculator does not support forces.")
    try:
      self.eV_to_lammps_unit = {"real": 23.0621, "metal": 1.0}[lammps_units]
    except KeyError:
      raise ValueError(f"lammps_units={lammps_units} not supported.")
    self.properties = ['energy', 'forces', 'stress']

  def compute_forces(self, data):
    # LAMMPS -> ASE
    atoms = self.ase_atoms_from_data(data)
    atoms.calc = self.ase_calculator
    forces = atoms.get_forces()
    potential_energy = atoms.get_potential_energy() * self.eV_to_lammps_unit

    # ASE -> LAMMPS
    n_local_atoms = data.nlistatoms
    local_forces = forces[ :n_local_atoms, : ]
    data.energy = potential_energy
    data.f[ :n_local_atoms, : ] = local_forces * self.eV_to_lammps_unit

  def ase_atoms_from_data(self, data):
    symbols = [self.elements[i] for i in data.elems]
    positions = data.x[:data.ntotal].copy()
    positions -= positions.min(axis=0) - 2.5
    min_pos = positions.min(axis=0)
    max_pos = positions.max(axis=0)
    cell = (max_pos - min_pos + 5.0)
    atoms = Atoms(
      symbols=symbols, positions=positions,
      cell=cell, pbc=[False, False, False],
      charges=data.charges, magmoms=data.magmoms
    )
    net_charge = data.charges[:data.nlocal].sum()
    if data.magmoms is not None:
      net_magmoms = np.sum(np.abs(data.magmoms[:data.nlocal]), axis=0)
    else:
      net_magmoms = np.zeros(3)
    net_magmom = np.sum(net_magmoms)
```

```
    atoms.info["net_charge"] = atoms.info["charge"] = net_charge.item()
    atoms.info["net_spin"] = atoms.info["spin"] = net_magmom + 1
    return atoms

  def compute_gradients(self, data):
    pass

  def compute_descriptors(self, data):
    pass
```

Listing A.7: A universal ASE-LAMMPS interface that can integrate any ASE calculator into LAMMPS via the ML-IAP package.

```
import lammps.mliap
from orb_models.forcefield import pretrained
from orb_models.forcefield.calculator import ORBCalculator

before_loading = \
  """
  [...]
  units  real
  dimension  3
  atom_style full
  [...]
  """

after_loading = \
  """
  pair_style mliap unified EXISTS
  [...]
  run [...]
  """

# Load lammps
lmp = lammps.lammps(cmdargs=['-echo', 'both'])

# Activate MLIAP interface
lammps.mliap.activate_mliappy(lmp)

# Run all commands that need to be done before loading the model
lmp.commands_string(before_loading)

# Load model
model = pretrained.orb_v3_direct_omol(compile=True, device=torch.device("cuda"))
# Wrap the model in ASE interface
calc = ORBCalculator(model=model, device=torch.device("cuda"))
# Wrap ASE interface in custom LAMMPS MLIAP interface,
# where the units above need to be specified
unified = UniversalASEUnifiedMLIAPInterface(calc, lammps_units="real")

# Inject the unified interface into LAMMPS MLIAP
lammps.mliap.load_unified(unified)

# Run the simulation
lmp.commands_string(after_loading_rep)

# Simulation completed here
lmp.close()
```

Listing A.8: Example integration of the Orb-v3 model into LAMMPS via ASE and the universal ASE-LAMMPS interface.

```
testsuite/
|-- .venv/
|-- data/
|   |-- mptraj.py
|   ^-- loaders.py
|-- driver/
|   |-- driver.py
|   |-- nequip_driver.py
|   |-- orb_driver.py
|   |-- sevennet_driver.py
|   |-- uma_driver.py
```

```
|   ^-- ...
|-- tests/
|   |-- diatomic_energy_tests.py
|   |-- inference_times.py
|   |-- nve_energy_drift.py
|   |-- phonons.py
|   ^-- ...
|-- results/
|   |-- diatomic_energy_tests/
|   |   |-- nequip_default.pkl
|   |   |-- orb_v2_direct_mptrj-only
|   |   ^-- ...
|   |-- inference_times/
|   |   |-- nequip_default.pkl
|   |   ^-- ...
|   ^-- ...
|-- plotting/
|   |-- plot_diatomic_energy.py
|   ^-- ...
|-- figures/
|   |-- diatomic_energy_tests/
|   |   |-- nequip_default.png
|   |   ^-- ...
|   ^-- ...
^-- main.py
```

Listing A.9: Folder structure of the monolith-environment testsuite.

```
testsuite/
|-- .venv/
|-- common/
|   |-- __init__.py
|   |-- driver.py
|   ^-- tasks.py
|-- drivers/
|   |-- eSEN/
|   |   |-- .venv/
|   |   |-- __init__.py
|   |   |-- esen_driver.py
|   |   ^-- requirements.txt
|   ^-- UMA/
|       |-- .venv/
|       |-- __init__.py
|       |-- uma_driver.py
|       ^-- requirements.txt
^-- main.py
```

Listing A.10: Example folder structure of a improved testsuite setup with separate folders for each model.

```
set -e

conda create -n lammps python=3.11 -y
conda activate lammps

VENV_PATH=$(python -c "import sys; print(sys.prefix)")

if [[ ! -x "${VENV_PATH}/bin/python" ]]; then
  echo "Error: Python not found in ${VENV_PATH}"
  exit 1
fi

"${VENV_PATH}/bin/python" -m pip install --upgrade pip setuptools wheel
"${VENV_PATH}/bin/python" -m pip install --upgrade lammps cython numpy

rm -rf lammps/
git clone -b positions4mliaps https://github.com/Jpx3/lammps.git --depth=1

cd lammps/

cmake -B build-mliap \
  -D CMAKE_BUILD_TYPE=Release \
  -D PKG_RIGID=yes \
  -D PKG_MC=yes \
  -D PKG_KSPACE=yes \
```

```
  -D PKG_ML-IAP=ON \
  -D PKG_MOLECULE=on \
  -D PKG_ML-SNAP=ON \
  -D MLIAP_ENABLE_PYTHON=ON \
  -D PKG_PYTHON=ON \
  -D BUILD_SHARED_LIBS=ON \
  -D Python_EXECUTABLE="${VENV_PATH}/bin/python" \
  cmake

cmake --build build-mliap -j 16
cd build-mliap
make install-python

cp -rf liblammps.so.0 "${VENV_PATH}/lib/python3.11/site-packages/lammps/liblammps.so"
cp -rf cython "${VENV_PATH}/lib/python3.11/site-packages/lammps/"
```

Listing A.11: Compile script for LAMMPS with a patched ML-IAP interface.

Glossary

ab-initio From first principles. 1, 43, 55, 61, 70, 71, 87

AIMD Ab-inhibito Molecular Dynamics. 20

ASE Atomic Simulation Environment. 2, 3, 32, 33, 36, 38–40, 43, 45, 70

BO Born-Oppenheimer. 5

CC Coupled Cluster. 20, 71

CCSD(T) Coupled Cluster with Singles, Doubles and perturbative Triples. 7

CFConv Continuous-Filter Convolution. 21–24, 63, 85

CNNs Convolutional Neural Networks. 17, 22

CUDA Compute Unified Device Architecture. 31, 32, 39, 70

DDPM Denoising Diffusion Probabilistic Models. 17, 18, 24, 31

DFT Density Functional Theory. 1, 3, 8, 11, 20, 32, 39–41, 43, 45, 48–53, 55–57, 67, 71, 86, 87

e3nn E(3) equivariant neural networks. 24, 26, 27, 63

ELBO Evidence Lower Bound. 19

eSEN Equivariant Smooth Energy Network. 25–27, 29–31, 39, 56, 59, 61, 63, 64, 66, 67, 87

ETDM Explicitly Time-Dependent Method. 40

FD Finite Difference. 40, 56

FT Fourier Transform. 15

GANs Generative Adversarial Networks. 17

GAP Gaussian Approximation Potentials. 21

GATs Graph Attention Networks. 12

GCMC Grand Canonical Monte Carlo. 46, 68–71, 87, 88

GDML Gradient-domain Machine Learning. 21, 34

GGA Generalized Gradient Approximation. 10, 11, 40, 56

GNN Graph Neural Network. 11, 21

GPAW Grid-based Projector Augmented Wave. 32, 39, 40, 56

HTS High-Throughput Screening. 1, 44, 63

Irreps Irreducible Representations. 13–16, 26, 64

JIT Just-In-Time. 43

KL Kullback-Leibler. 18

LAMMPS Large-scale Atomic/Molecular Massively Parallel Simulator. 2, 32–38, 46, 47, 70

LCAO Linear Combination of Atomic Orbitals. 40, 56

LDA Local Density Approximation. 10, 11

LMC Langevin Monte Carlo. 17

MAE Mean Absolute Error. 51

MD Molecular Dynamics. 1, 2, 20, 30, 32, 35, 39, 42, 43, 46, 54, 56, 57, 63, 64, 70

ML Machine Learning. 1, 32, 33

ML-IAP Machine Learning Inter Atomic Potentials. 35

MLIP Machine-Learning Interatomic Potential. 1–3, 11, 20–22, 25, 26, 29–39, 41–46, 61, 65, 70, 71

MLP Multi-Layer Perceptron. 11, 12, 16, 22, 71

MoE Mixture of Experts. 31

MOF Metal-Organic Framework. 1–3, 42–47, 65, 67–70, 86–88

MoLE Mixture of Linear Experts. 30, 64

MPI Message Passing Interface. 40

MPNNs Message Passing Neural Networks. 11

MPtrj Materials Project Trajectories. 25, 54, 57, 87

MSD Mean Squared Displacement. 42, 59, 60, 87

NequIP Neural Equivariant Interatomic Potentials. 24, 35, 38

NNP Neural Network Potential. 25

NPT constant number of particles, pressure and temperature. 32

NVE constant number of particles, volume and energy. 41, 42

NVT constant number of particles, volume and temperature. 42

OAM Omat24, Alexandria, and MPTraj. 59, 61, 63, 64

OOD Out-Of-Distribution. 71

PaiNN Polarizable atom interaction Neural Network. 22, 54, 56, 63, 64, 67, 87

PBE Perdew–Burke–Ernzerhof exchange–correlation functional. 11, 40, 41, 50–53, 56, 86, 87

PES Potential Energy Surface. 22, 25, 51, 54, 56, 70, 71, 85

PPPM Particle-Particle Particle-Mesh. 32

RBF Radial Basis Functions. 21–23, 29

RMSE Root Mean Square Error. 25

SDE Stochastic Differential Equation. 18

SE(3) Special Euclidean Group in 3D. 13

SEM Scanning Electron Microscope. 46, 86

SMLD Score Matching with Langevin dynamics. 17

SO(2) Special Orthogonal Group in 2D. 26, 27, 63, 64, 85

SO(3) Special Orthogonal Group in 3D. 13–16, 24, 26, 28

SOTA State Of The Art. 2, 3, 20, 25, 31, 38, 56, 70

TZP Triple-Zeta with Polarization. 40

UMA Universal Models for Atoms. 26, 29, 39, 45, 50, 51, 54, 56, 61, 63, 64, 66, 67, 86, 87

VAEs Variational Autoencoders. 17

VASP Vienna Ab initio Simulation Package. 58, 89

vdW van der Waals. 7, 11, 40, 44, 45, 47, 56

XC Exchange-Correlation. 10, 11

List of Figures

List of Tables

Bibliography

[1] J. Lennard-Jones. "On the determination of molecular fields". In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 106.738 (1924), pp. 463–477. DOI: 10.1098/rspa.1924.0082.

[2] P. Hohenberg and W. Kohn. "Inhomogeneous Electron Gas". In: *Phys. Rev.* 136 (3B Nov. 1964), B864–B871. DOI: 10.1103/PhysRev.136.B864. URL: https://link.aps.org/doi/10.1103/PhysRev.136.B864.

[3] Y. Fu, Y. Yao, A. C. Forse, J. G. Vitillo, B. Civalleri, S. Yang, Y. Wang, J. A. Reimer, S. Bordiga, K. Chen, and B. Smit. "Solvent-derived defects suppress adsorption in MOF-74". In: *Nature Communications* 14 (2023), p. 2386. DOI: 10.1038/s41467-023-38155-8. URL: https://doi.org/10.1038/s41467-023-38155-8.

[4] M. J. Buehler, A. Hartmaier, H. Gao, M. Duchaineau, and F. F. Abraham. "Atomic plasticity: description and analysis of a one-billion atom simulation of ductile materials failure". In: *Computer Methods in Applied Mechanics and Engineering* 193.48 (2004). Advances in Computational Plasticity, pp. 5257–5282. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2003.12.066. URL: https://www.sciencedirect.com/science/article/pii/S0045782504002749.

[5] J. Perilla and K. Schulten. "Physical properties of the HIV-1 capsid from all-atom molecular dynamics simulations". In: *Nature Communications* 8 (2017), p. 15959. DOI: 10.1038/ncomms15959. URL: https://doi.org/10.1038/ncomms15959.

[6] K. Nordlund. "Atomistic simulations of plasma-wall interactions in fusion reactors". In: *Physica Scripta* T124 (May 2006), pp. 53–57. DOI: 10.1088/0031-8949/2006/T124/011.

[7] L. Yu, X. Chen, N. Yao, Y.-C. Gao, and Q. Zhang. "Constant-potential molecular dynamics simulation and its application in rechargeable batteries". In: *J. Mater. Chem. A* 11 (21 2023), pp. 11078–11088. DOI: 10.1039/D3TA01411H. URL: http://dx.doi.org/10.1039/D3TA01411H.

[8] A. Johansson, Y. Xie, C. J. Owen, J. S. Lim, L. Sun, J. Vandermause, and B. Kozinsky. *Micron-scale heterogeneous catalysis with Bayesian force fields from first principles and active learning*. 2022. arXiv: 2204.12573 [physics.comp-ph]. URL: https://arxiv.org/abs/2204.12573.

[9] K. Sumida, D. L. Rogow, J. A. Mason, T. M. McDonald, E. D. Bloch, Z. R. Herm, T.-H. Bae, and J. R. Long. "Carbon Dioxide Capture in Metal–Organic Frameworks". In: *Chemical Reviews* 112.2 (2012). PMID: 22204561, pp. 724–781. DOI: 10.1021/cr2003272.

[10] Z. Chen, P. Li, R. Anderson, X. Wang, X. Zhang, L. Robison, L. R. Redfern, S. Moribe, T. Islamoglu, D. A. Gómez-Gualdrón, T. Yildirim, J. F. Stoddart, and O. K. Farha. "Balancing volumetric and gravimetric uptake in highly porous materials for clean energy". In: *Science* 368.6488 (2020), pp. 297–303. DOI: 10.1126/science.aaz8881. eprint: https://www.science.org/doi/pdf/10.1126/science.aaz8881. URL: https://www.science.org/doi/abs/10.1126/science.aaz8881.

[11] K. Burke. "Perspective on density functional theory". In: *The Journal of Chemical Physics* 136.15 (Apr. 2012). ISSN: 1089-7690. DOI: 10.1063/1.4704546. URL: http://dx.doi.org/10.1063/1.4704546.

[12] R. Goeminne, L. Vanduyfhuys, V. Van Speybroeck, and T. Verstraelen. "DFT-Quality adsorption simulations in metal–organic frameworks enabled by machine learning Potentials". In: *Journal of Chemical Theory and Computation* 19.18 (2023), pp. 6313–6325.

[13] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller. *SchNet: A continuous-filter convolutional neural network for modeling quantum interactions*. 2017. arXiv: 1706.08566 [stat.ML]. URL: https://arxiv.org/abs/1706.08566.

[14] K. T. Schütt, O. T. Unke, and M. Gastegger. "Equivariant message passing for the prediction of tensorial properties and molecular spectra". In: *CoRR* abs/2102.03150 (2021). arXiv: 2102.03150. URL: https://arxiv.org/abs/2102.03150.

[15] B. M. Wood, M. Dzamba, X. Fu, M. Gao, M. Shuaibi, L. Barroso-Luque, K. Abdelmaqsoud, V. Gharakhanyan, J. R. Kitchin, D. S. Levine, K. Michel, A. Sriram, T. Cohen, A. Das, A. Rizvi, S. J. Sahoo, Z. W. Ulissi, and C. L. Zitnick. *UMA: A Family of Universal Models for Atoms*. 2025. arXiv: 2506.23971 [cs.LG]. URL: https://arxiv.org/abs/2506.23971.

[16] T. Koker and T. Smidt. *Training a Foundation Model for Materials on a Budget*. 2025. arXiv: 2508.16067 [physics.comp-ph]. URL: https://arxiv.org/abs/2508.16067.

[17] F. Bigi, J. W. Abbott, P. Loche, A. Mazitov, D. Tisi, M. F. Langer, A. Goscinski, P. Pegolo, S. Chong, R. Goswami, S. Chorna, M. Kellner, M. Ceriotti, and G. Fraux. *Metatensor and Metatomic: Foundational Libraries for Interoperable Atomistic Machine Learning*. Aug. 2025. DOI: 10.48550/arXiv.2508.15704.

[18] W. Pauli. "Über den Zusammenhang des Abschlusses der Elektronengruppen im Atom mit der Komplexstruktur der Spektren". In: *Zeitschrift für Physik* 31.1 (1925), pp. 765–783. ISSN: 0044-3328. DOI: 10.1007/BF02980631. URL: https://doi.org/10.1007/BF02980631.

[19] C. C. J. Roothaan. "New Developments in Molecular Orbital Theory". In: *Rev. Mod. Phys.* 23 (2 Apr. 1951), pp. 69–89. DOI: 10.1103/RevModPhys.23.69. URL: https://link.aps.org/doi/10.1103/RevModPhys.23.69.

[20] M. H. Rashid. "An Introduction to Density Functional Theory". In: (). Accessed: 2025-09-07. URL: https://rashid-phy.github.io/me/resources/DFT/An_Introduction_to_DFT.pdf.

[21] G. Giuliani and G. Vignale. *Quantum Theory of the Electron Liquid*. Cambridge University Press, 2005.

[22] W. Kohn and L. J. Sham. "Self-Consistent Equations Including Exchange and Correlation Effects". In: *Phys. Rev.* 140 (4A Nov. 1965), A1133–A1138. DOI: 10.1103/PhysRev.140.A1133. URL: https://link.aps.org/doi/10.1103/PhysRev.140.A1133.

[23] J. P. Perdew, K. Burke, and M. Ernzerhof. "Generalized Gradient Approximation Made Simple". In: *Phys. Rev. Lett.* 77 (18 Oct. 1996), pp. 3865–3868. DOI: 10.1103/PhysRevLett.77.3865. URL: https://link.aps.org/doi/10.1103/PhysRevLett.77.3865.

[24] A. D. Becke. "Density-functional exchange-energy approximation with correct asymptotic behavior". In: *Phys. Rev. A* 38 (6 Sept. 1988), pp. 3098–3100. DOI: 10.1103/PhysRevA.38.3098. URL: https://link.aps.org/doi/10.1103/PhysRevA.38.3098.

[25] A. D. Becke. "Density-functional thermochemistry. III. The role of exact exchange". In: *The Journal of Chemical Physics* 98.7 (1993), pp. 5648–5652. DOI: 10.1063/1.464913.

[26] S. Grimme, J. Antony, S. Ehrlich, and H. Krieg. "A consistent and accurate ab initio parametrization of density functional dispersion correction (DFT-D) for the 94 elements H–Pu". In: *The Journal of Chemical Physics* 132.15 (2010), p. 154104. DOI: 10.1063/1.3382344.

[27] S. Grimme. "Density functional theory with London dispersion corrections". In: *WIREs Computational Molecular Science* 1.2 (2011), pp. 211–228. DOI: https://doi.org/10.1002/wcms.30. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.30. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcms.30.

[28] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. *Neural Message Passing for Quantum Chemistry*. 2017. arXiv: 1704.01212 [cs.LG]. URL: https://arxiv.org/abs/1704.01212.

[29] X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[30] E. Noether. "Invariante Variationsprobleme". In: *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* (1918), pp. 235–257. URL: https://arxiv.org/abs/physics/0503066.

[31] K. Sheriff. *Demistifying E3NN: A Hands-On Introduction to Euclidean Neural Networks*. Accessed: 2025-11-02. 2023. URL: https://killiansheriff.github.io/DemistifyingE3NN/.

[32] E. P. Wigner. *Group Theory*. Original work published 1959. Elsevier Science, 2012. URL: https://www.perlego.com/book/1874081/group-theory-and-its-application-to-the-quantum-mechanics-of-atomic-spectra-pdf.

[33] A. Duval, S. V. Mathis, C. K. Joshi, V. Schmidt, S. Miret, F. D. Malliaros, T. Cohen, P. Liò, Y. Bengio, and M. Bronstein. *A Hitchhiker's Guide to Geometric GNNs for 3D Atomic Systems*. 2024. arXiv: 2312.07511 [cs.LG]. URL: https://arxiv.org/abs/2312.07511.

[34] G. Peyré. *Spherical harmonics are the equivalent on the sphere of the Fourier basis: Illustration*. 2022. URL: https://twitter.com/gabrielpeyre/status/1557592647828742145.

[35] O. Tweezers. *Figure 5.2: Spherical Harmonics*. URL: http://opticaltweezers.org/chapter-5-electromagnetic-theory/figure-5-2-spherical-harmonics/.

[36] D. P. Kingma and M. Welling. *Auto-Encoding Variational Bayes*. 2013. arXiv: 1312.6114 [stat.ML].

[37] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML]. URL: https://arxiv.org/abs/1406.2661.

[38] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. "Language Models are Unsupervised Multitask Learners". In: *OpenAI* (2019). URL: https://openai.com/blog/language-unsupervised/.

[39] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language Models are Few-Shot Learners". In: *arXiv preprint arXiv:2005.14165* (2020). URL: https://arxiv.org/abs/2005.14165.

[40] OpenAI. "GPT-4 Technical Report". In: *arXiv preprint arXiv:2303.08774* (2023). URL: https://arxiv.org/abs/2303.08774.

[41] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG]. URL: https://arxiv.org/abs/2011.13456.

[42] S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller. "Machine learning of accurate energy-conserving molecular force fields". In: *Science Advances* 3.5 (May 2017). ISSN: 2375-2548. DOI: 10.1126/sciadv.1603015. URL: http://dx.doi.org/10.1126/sciadv.1603015.

[43] A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi. "Gaussian Approximation Potentials: The Accuracy of Quantum Mechanics, without the Electrons". In: *Phys. Rev. Lett.* 104 (13 Apr. 2010), p. 136403. DOI: 10.1103/PhysRevLett.104.136403. URL: https://link.aps.org/doi/10.1103/PhysRevLett.104.136403.

[44] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng. *Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains*. 2020. arXiv: 2006.10739 [cs.CV].

[45] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[46] N. Thomas, T. E. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley. "Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds". In: *CoRR* abs/1802.08219 (2018). arXiv: 1802.08219. URL: http://arxiv.org/abs/1802.08219.

[47] J. Klicpera, J. Groß, and S. Günnemann. "Directional Message Passing for Molecular Graphs". In: *CoRR* abs/2003.03123 (2020). arXiv: 2003.03123. URL: https://arxiv.org/abs/2003.03123.

[48] S. Batzner, A. Musaelian, L. Sun, M. Geiger, J. P. Mailoa, M. Kornbluth, N. Molinari, T. E. Smidt, and B. Kozinsky. "E(3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials". In: *Nature Communications* 13.1 (May 2022). ISSN: 2041-1723. DOI: 10.1038/s41467-022-29939-5. URL: http://dx.doi.org/10.1038/s41467-022-29939-5.

[49] M. Geiger and T. Smidt. *e3nn: Euclidean Neural Networks*. 2022. DOI: 10.48550/ARXIV.2207.09453. URL: https://arxiv.org/abs/2207.09453.

[50] M. Weiler, M. Geiger, M. Welling, W. Boomsma, and T. Cohen. *3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data*. 2018. arXiv: 1807.02547 [cs.LG]. URL: https://arxiv.org/abs/1807.02547.

[51] R. Kondor, Z. Lin, and S. Trivedi. *Clebsch-Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network*. 2018. arXiv: 1806.09231 [stat.ML]. URL: https://arxiv.org/abs/1806.09231.

[52] M. Geiger, T. Smidt, A. M., B. K. Miller, W. Boomsma, B. Dice, K. Lapchevskyi, M. Weiler, M. Tyszkiewicz, S. Batzner, D. Madisetti, M. Uhrin, J. Frellsen, N. Jung, S. Sanborn, M. Wen, J. Rackers, M. Rød, and M. Bailey. *Euclidean neural networks: e3nn*. Version 0.5.0. Apr. 2022. DOI: 10.5281/zenodo.6459381. URL: https://doi.org/10.5281/zenodo.6459381.

[53] NVIDIA. *cuEquivariance*. Version 0.6.1. Aug. 2025. URL: https://github.com/NVIDIA/cuEquivariance.

[54] M. Neumann, J. Gin, B. Rhodes, S. Bennett, Z. Li, H. Choubisa, A. Hussey, and J. Godwin. *Orb: A Fast, Scalable Neural Network Potential*. 2024. arXiv: 2410.22570 [cond-mat.mtrl-sci]. URL: https://arxiv.org/abs/2410.22570.

[55] S. Zaidi, M. Schaarschmidt, J. Martens, H. Kim, Y. W. Teh, A. Sanchez-Gonzalez, P. Battaglia, R. Pascanu, and J. Godwin. *Pre-training via Denoising for Molecular Property Prediction*. 2022. arXiv: 2206.00133 [cs.LG]. URL: https://arxiv.org/abs/2206.00133.

[56] L. Harcombe and T. T. Duignan. *On the Connection Between Diffusion Models and Molecular Dynamics*. 2025. arXiv: 2504.03187 [cs.LG]. URL: https://arxiv.org/abs/2504.03187.

[57] J. Riebesell, R. E. A. Goodall, P. Benner, Y. Chiang, B. Deng, G. Ceder, M. Asta, A. A. Lee, A. Jain, and K. A. Persson. "A framework to evaluate machine learning crystal stability predictions". In: *Nature Machine Intelligence* 7.6 (2025), pp. 836–847. ISSN: 2522-5839. DOI: 10.1038/s42256-025-01055-1. URL: https://doi.org/10.1038/s42256-025-01055-1.

[58] B. Rhodes, S. Vandenhaute, V. Šimkus, J. Gin, J. Godwin, T. Duignan, and M. Neumann. *Orb-v3: atomistic simulation at scale*. 2025. arXiv: 2504.06231 [cond-mat.mtrl-sci]. URL: https://arxiv.org/abs/2504.06231.

[59] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[60] Z. DeVito, C. Geng, and X. Si. *TorchScript: Optimized execution of PyTorch models*. 2023. URL: https://program-transformations.github.io/slides/pytorch_neurips.pdf.

[61] B. Deng. *Materials Project Trajectory (MPtrj) dataset*. figshare. 2023. DOI: 10.6084/m9.figshare.23713842.

[62] X. Fu, B. M. Wood, L. Barroso-Luque, D. S. Levine, M. Gao, M. Dzamba, and C. L. Zitnick. *Learning Smooth and Expressive Interatomic Potentials for Physical Property Prediction*. 2025. arXiv: 2502.12147 [physics.comp-ph]. URL: https://arxiv.org/abs/2502.12147.

[63] S. Passaro and C. L. Zitnick. *Reducing SO(3) Convolutions to SO(2) for Efficient Equivariant GNNs*. 2023. arXiv: 2302.03655 [cs.LG]. URL: https://arxiv.org/abs/2302.03655.

[64] C. L. Zitnick, A. Das, A. Kolluru, J. Lan, M. Shuaibi, A. Sriram, Z. Ulissi, and B. Wood. *Spherical Channels for Modeling Atomic Interactions*. 2022. arXiv: 2206.14331 [physics.chem-ph]. URL: https://arxiv.org/abs/2206.14331.

[65] M. Bôcher. *Introduction to the Theory of Fourier's Series*. Harvard University, 1906. URL: https://books.google.de/books?id=gQ43AQAAMAAJ.

[66] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2.

[67] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3 (Mar. 1991), pp. 79–87. DOI: 10.1162/neco.1991.3.1.79.

[68] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean. "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer". In: *CoRR* abs/1701.06538 (2017). arXiv: 1701.06538. URL: http://arxiv.org/abs/1701.06538.

[69] NVIDIA Corporation. *CUDA Toolkit Documentation, Version 13.2*. Software. 2025. URL: https://developer.nvidia.com/cuda-toolkit.

[70] A. Hjorth Larsen, J. Jørgen Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. Bjerre Jensen, J. Kermode, J. R. Kitchin, E. Leonhard Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. Bergmann Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen. "The atomic simulation environment - a Python library for working with atoms". In: *Journal of Physics: Condensed Matter* 29.27 (June 2017), p. 273002. DOI: 10.1088/1361-648X/aa680e. URL: https://doi.org/10.1088/1361-648X/aa680e.

[71] H. Bekker, H. Berendsen, E. Dijkstra, S. Achterop, R. Drunen, D. van der Spoel, A. Sijbers, H. Keegstra, B. Reitsma, and M. Renardus. "Gromacs: A parallel computer for molecular dynamics simulations". In: *Physics Computing* 92 (Jan. 1993), pp. 252–256.

[72] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl. "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers". In: *SoftwareX* 1-2 (2015), pp. 19–25. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2015.06.001. URL: https://www.sciencedirect.com/science/article/pii/S2352711015000059.

[73] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales". In: *Comp. Phys. Comm.* 271 (2022), pp. 108–171. DOI: 10.1016/j.cpc.2021.108171.

[74] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. *JAX: composable transformations of Python+NumPy programs*. http://github.com/google/jax. 2018.

[75] *ASE Calculator Documentation*. URL: https://ase-lib.org/ase/calculators/calculators.html.

[76] J. J. Mortensen, A. H. Larsen, M. Kuisma, A. V. Ivanov, A. Taghizadeh, A. Peterson, A. Haldar, A. O. Dohn, C. Schäfer, E. Ö. Jónsson, E. D. Hermes, F. A. Nilsson, G. Kastlunger, G. Levi, H. Jónsson, H. Häkkinen, J. Fojt, J. Kangsabanik, J. Sødequist, J. Lehtomäki, J. Heske, J. Enkovaara, K. T. Winther, M. Dulak, M. M. Melander, M. Ovesen, M. Louhivuori, M. Walter, M. Gjerding, O. Lopez-Acevedo, P. Erhart, R. Warmbier, R. Würdemann, S. Kaappa, S. Latini, T. M. Boland, T. Bligaard, T. Skovhus, T. Susi, T. Maxson, T. Rossi, X. Chen, Y. L. A. Schmerwitz, J. Schiøtz, T. Olsen, K. W. Jacobsen, and K. S. Thygesen. "GPAW: An open Python package for electronic structure calculations". In: *The Journal of Chemical Physics* 160.9 (Mar. 2024), p. 092503. ISSN: 0021-9606. DOI: 10.1063/5.0182685. eprint: https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/5.0182685/19717263/092503_1_5.0182685.pdf. URL: https://doi.org/10.1063/5.0182685.

[77] S. Plimpton. "Fast Parallel Algorithms for Short-Range Molecular Dynamics". In: *Journal of Computational Physics* 117.1 (1995), pp. 1–19. ISSN: 0021-9991. DOI: https://doi.org/10.1006/jcph.1995.1039. URL: https://www.sciencedirect.com/science/article/pii/S002199918571039X.

[78] *LAMMPS Authors*. URL: https://www.lammps.org/authors.html.

[79] *LAMMPS Packages Documentation*. URL: https://docs.lammps.org/Packages.html.

[80] J. Eastwood, R. Hockney, and D. Lawrence. "P3M3DP—The three-dimensional periodic particle-particle/ particle-mesh program". In: *Computer Physics Communications* 19.2 (1980), pp. 215–261. ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(80)90052-1. URL: https://www.sciencedirect.com/science/article/pii/0010465580900521.

[81] H. C. Edwards, C. R. Trott, and D. Sunderland. "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns". In: *Journal of Parallel and Distributed Computing* 74.12 (2014). Domain-Specific Languages and High-Level Frameworks for High-Performance Computing, pp. 3202–3216. ISSN: 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2014.07.003. URL: http://www.sciencedirect.com/science/article/pii/S0743731514001257.

[82] C. R. Trott, D. Lebrun-Grandié, D. Arndt, J. Ciesko, V. Dang, N. Ellingwood, R. Gayatri, E. Harvey, D. S. Hollman, D. Ibanez, N. Liber, J. Madsen, J. Miles, D. Poliakoff, A. Powell, S. Rajamanickam, M. Simberg, D. Sunderland, B. Turcksin, and J. Wilke. "Kokkos 3: Programming Model Extensions for the Exascale Era". In: *IEEE Transactions on Parallel and Distributed Systems* 33.4 (2022), pp. 805–817. DOI: 10.1109/TPDS.2021.3097283.

[83] H. Yang, C. Hu, Y. Zhou, X. Liu, Y. Shi, J. Li, G. Li, Z. Chen, S. Chen, C. Zeni, M. Horton, R. Pinsler, A. Fowler, D. Zügner, T. Xie, J. Smith, L. Sun, Q. Wang, L. Kong, C. Liu, H. Hao, and Z. Lu. "MatterSim: A Deep Learning Atomistic Model Across Elements, Temperatures and Pressures". In: *arXiv preprint arXiv:2405.04967* (2024). arXiv: 2405.04967 [cond-mat.mtrl-sci]. URL: https://arxiv.org/abs/2405.04967.

[84] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith. "Cython: The Best of Both Worlds". In: *Computing in Science & Engineering* 13.2 (2011), pp. 31–39. DOI: 10.1109/MCSE.2010.118.

[85] C. R. Harris, K. J. Millman, van der Walt, and more. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[86] S. R. Anaya. *LAMMPS Pair Force Update Rule*. Accessed: 2025-09-09. URL: https://github.com/lammps/lammps/blob/75f9ae5de93f7e1705ba94180929f3677e3a68ae/src/ML-IAP/mliap_unified.cpp#L264.

[87] *NequIP LAMMPS ML-IAP Integration using Autograd*. URL: https://github.com/mir-group/nequip/blob/2471fee489e7a90ce42634205fbe4f6c36f741e3/nequip/integrations/lammps_mliap/lmp_mliap_wrapper.py#L183.

[88] *LAMMPS GCMC package spam-calling pair potential*. Accessed: 2025-10-14. URL: https://github.com/Jpx3/lammps/blob/ff3159cc767256add0b356c9a62be9a4c750c826/src/MC/fix_gcmc.cpp#L2277.

[89] I. Batatia, D. P. Kovacs, G. N. C. Simm, C. Ortner, and G. Csanyi. "MACE: Higher Order Equivariant Message Passing Neural Networks for Fast and Accurate Force Fields". In: *Advances in Neural Information Processing Systems*. Ed. by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho. 2022. URL: https://openreview.net/forum?id=YPpSngE-ZU.

[90] ICAMS. *Integrating MACE into LAMMPS*. URL: https://github.com/ICAMS/lammps-user-pace.

[91] wcwitt. *ML-IAP with JAX: lammps/lammps pull draft Nr. 4691*. Sept. 8, 2025. URL: https://github.com/lammps/lammps/pull/4691.

[92] G. van Rossum. *The Python Library Reference, release 3.8.2*. 2020. URL: https://docs.python.org/3/library/pickle.html.

[93] C. W. Tan, M. L. Descoteaux, M. Kotak, G. de Miranda Nascimento, S. R. Kavanagh, L. Zichi, M. Wang, A. Saluja, Y. R. Hu, T. Smidt, A. Johansson, W. C. Witt, B. Kozinsky, and A. Musaelian. *High-performance training and inference for deep equivariant interatomic potentials*. 2025. arXiv: 2504.16068 [physics.comp-ph]. URL: https://arxiv.org/abs/2504.16068.

[94] Y. Litman, V. Kapil, Y. M. Y. Feldman, D. Tisi, T. Begušić, K. Fidanyan, G. Fraux, J. Higer, M. Kellner, T. E. Li, E. S. Pós, E. Stocco, G. Trenins, B. Hirshberg, M. Rossi, and M. Ceriotti. "i-PI 3.0: A flexible and efficient framework for advanced atomistic simulations". In: *The Journal of Chemical Physics* 161.6 (Aug. 2024), p. 062504. ISSN: 0021-9606. DOI: 10.1063/5.0215869. eprint: https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/5.0215869/20111898/062504_1_5.0215869.pdf. URL: https://doi.org/10.1063/5.0215869.

[95] The Debian Project. *Debian GNU/Linux*. https://www.debian.org/. Accessed: 2025-09-08.

[96] Microsoft. *Hyper-V*. https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/. Accessed: 2025-09-08.

[97] *NequIP LAMMPS ML-IAP Integration Documentation*. Accessed: 2025-09-08. URL: https://nequip.readthedocs.io/en/develop/integrations/lammps/mliap.html#building-ml-iap.

[98] Units Converters. *eV to kcal/mol Converter*. Accessed: 2025-10-30. URL: https://www.unitsconverters.com/en/Ev/Particle-To-Kcal/Mol/Utu-6180-7727.

[99] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson. "A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters". English. In: *The Journal of Chemical Physics* 76.1 (1982), pp. 637–649. URL: http://dx.doi.org/10.1063/1.442716.

[100] P. Grigorev, L. Frérot, F. Birks, A. Gola, J. Golebiowski, J. Grießer, J. L. Hörmann, A. Klemenz, G. Moras, W. G. Nöhring, J. A. Oldenstaedt, P. Patel, T. Reichenbach, T. Rocke, L. Shenoy, M. Walter, S. Wengert, L. Zhang, J. R. Kermode, and L. Pastewka. "matscipy: materials science at the atomic scale with Python". In: *Journal of Open Source Software* 9.93 (Jan. 2024), p. 5668. DOI: 10.21105/joss.05668. URL: https://joss.theoj.org/papers/10.21105/joss.05668.

[101] Astral. *UV: A Fast, All-in-One Python Package Manager*. https://astral.sh/blog/uv. Accessed: 2025-09-24. 2025.

[102] A. Togo, L. Chaput, T. Tadano, and I. Tanaka. "Implementation strategies in phonopy and phono3py". In: *J. Phys. Condens. Matter* 35.35 (2023), p. 353001. DOI: 10.1088/1361-648X/acd831.

[103] A. Togo. "First-principles Phonon Calculations with Phonopy and Phono3py". In: *J. Phys. Soc. Jpn.* 92.1 (2023), p. 012001. DOI: 10.7566/JPSJ.92.012001.

[104] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. Available at https://www.mpi-forum.org/docs/. MPI Forum. 1994.

[105] A. Togo. *MDR phonon calculation database*. 2025. DOI: 10.48505/nims.4197. URL: https://doi.org/10.48505/nims.4197.

[106] N. W. Ashcroft and N. D. Mermin. “Solid State Physics”. In: *Physik in unserer Zeit* 9.1 (1978), pp. 33–33. DOI: https://doi.org/10.1002/piuz.19780090109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/piuz.19780090109. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/piuz.19780090109.

[107] S. Bandi, C. Jiang, and C. A. Marianetti. *Benchmarking phonon anharmonicity in machine learning interatomic potentials*. 2024. arXiv: 2402.18891 [cond-mat.mtrl-sci].

[108] A. Togo. *Phonopy Workflow*. 2025. URL: https://phonopy.github.io/phonopy/workflow.html.

[109] J. Donald R. Burgess. *Self-Diffusion and Binary-Diffusion Coefficients in Gases*. NIST Technical Note TN 2279. National Institute of Standards and Technology, Feb. 2024. DOI: 10.6028/NIST.TN.2279.

[110] T. R. Marrero and E. A. Mason. “Gaseous Diffusion Coefficients”. In: *Journal of Physical and Chemical Reference Data* 1 (1972). Published Online: 28 October 2009, p. 3. DOI: 10.1063/1.3253094. URL: https://srd.nist.gov/jpcrdreprint/1.3253094.pdf.

[111] H. Furukawa, K. E. Cordova, and M. O’Keeffe. “The Chemistry and Applications of Metal-Organic Frameworks”. In: *Science* 341 (2013), pp. 1234–1239.

[112] T. Xiao and D. Liu. “The most advanced synthesis and a wide range of applications of MOF-74 and its derivatives”. In: *Microporous and Mesoporous Materials* 283 (2019), pp. 88–103. ISSN: 1387-1811. DOI: https://doi.org/10.1016/j.micromeso.2019.03.002. URL: https://www.sciencedirect.com/science/article/pii/S1387181119301283.

[113] G. Degaga, A. Studinger, and L. Valenzano-Slough. *A Quantum Mechanical Description of the Diffusion Properties of C1-C2 Hydrocarbon Molecules in MOF-74-Mg*. May 2022. DOI: 10.26434/chemrxiv-2022-1zrjl.

[114] S. R. Caskey, A. G. Wong-Foy, and A. J. Matzger. “Dramatic tuning of carbon dioxide uptake via metal substitution in a coordination polymer with cylindrical pores”. In: *Journal of the American Chemical Society* 130.33 (2008), pp. 10870–10871.

[115] I. P. on Climate Change (IPCC). *Climate Change 2021: The Physical Science Basis*. Accessed: 2025-10-24. Cambridge University Press, 2021. URL: https://www.ipcc.ch/report/ar6/wg1/.

[116] J. T. Houghton. *Global Warming: The Complete Briefing*. Cambridge, UK: Cambridge University Press, 2004. ISBN: 9780521528740. URL: https://books.google.com/books/about/Global_Warming.html?id=jE9mwoLXdwYC.

[117] M. E. Mann, R. S. Bradley, and M. K. Hughes. “Global-Scale Temperature Patterns and Climate Forcing Over the Past Six Centuries”. In: *Nature* 392.6678 (1998), pp. 779–787. DOI: 10.1038/33859. URL: https://www.nature.com/articles/nature02478.

[118] J. Lovelock. *The Revenge of Gaia: Earth's Climate Crisis and the Fate of Humanity*. London: Allen Lane, 2006. ISBN: 9780713998691. URL: https://books.google.com/books/about/The_Vanishing_Face_of_Gaia.html?id=GS9J-GrdrJcC.

[119] B. Metz, O. Davidson, H. d. Coninck, M. Loos, and L. A. Meyer. "Carbon Dioxide Capture and Storage". In: *Intergovernmental Panel on Climate Change (IPCC) Special Report* (2005). URL: https://www.ipcc.ch/report/carbon-dioxide-capture-and-storage/.

[120] J.-R. Li, Y. Ma, M. C. McCarthy, J. Sculley, J. Yu, H.-K. Jeong, P. B. Balbuena, and H.-C. Zhou. "Carbon dioxide capture-related gas adsorption and separation in metal-organic frameworks". In: *Coordination Chemistry Reviews* 255.15 (2011), pp. 1791–1823. ISSN: 0010-8545. DOI: https://doi.org/10.1016/j.ccr.2011.02.012. URL: https://www.sciencedirect.com/science/article/pii/S0010854511000701.

[121] D. J. Griffiths. *Introduction to Electrodynamics*. 4th. Boston, MA: Pearson, 2013. Chap. 1.

[122] D. S. Levine, M. Shuaibi, and E. W. C. S.-S. et al. *The Open Molecules 2025 (OMol25) Dataset, Evaluations, and Models*. 2025. arXiv: 2505.08762 [physics.chem-ph]. URL: https://arxiv.org/abs/2505.08762.

[123] FAIR. *UMA: Universal Models for Atoms*. Aug. 2025. URL: https://huggingface.co/spaces/facebook/fairchem_uma_demo.

[124] B. Widom. "Some Topics in the Theory of Liquids". In: *Journal of Chemical Physics* 39.11 (1963), pp. 2808–2812. DOI: 10.1063/1.1734110. URL: https://doi.org/10.1063/1.1734110.

[125] Z. Bao, L. Yu, Q. Ren, X. Lu, and S. Deng. "Adsorption of CO2 and CH4 on a magnesium-based metal organic framework". In: *Journal of Colloid and Interface Science* 353.2 (2011), pp. 549–556. ISSN: 0021-9797. DOI: https://doi.org/10.1016/j.jcis.2010.09.065. URL: https://www.sciencedirect.com/science/article/pii/S0021979710011161.

[126] K. Persson. *Materials Data on BaPdF4 (SG:140) by Materials Project*. Nov. 2014. DOI: 10.17188/1189089.

[127] A. Jain, S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, and K. a. Persson. "Commentary: The Materials Project: A materials genome approach to accelerating materials innovation". In: *APL Materials* 1.1 (2013), p. 011002. ISSN: 2166532X. DOI: 10.1063/1.4812323. URL: https://doi.org/10.1063/1.4812323.

[128] B. Mueller and R. Hoppe. "Neue ternaere Fluoride mit Pd(II): M(II) Pd F4, mit M(II)= Ca, Sr, Ba, Pb". In: *Materials Research Bulletin* 7 (1972), pp. 1297–1306.

[129] I. Pallikara, P. Kayastha, J. Skelton, and L. Whalley. "The Physical Significance of Imaginary Phonon Modes in Crystals". In: *Electronic Structure* 4 (July 2022). DOI: 10.1088/2516-1075/ac78b3.

[130] M. Allen, D. Poggiali, K. Whitaker, T. Marshall, J. van Langen, and R. Kievit. "Raincloud plots: a multi-platform tool for robust data visualization [version 2; peer review: 2 approved]". In: *Wellcome Open Research* 4.63 (2021). DOI: 10.12688/wellcomeopenres.15191.2.

[131] Y. Park, J. Kim, S. Hwang, and S. Han. "Scalable Parallel Algorithm for Graph Neural Network Interatomic Potentials in Molecular Dynamics Simulations". In: *J. Chem. Theory Comput.* 20.11 (2024), pp. 4857–4868. DOI: 10.1021/acs.jctc.4c00190.

[132] I. Cuesta, J. Sanchez-Marín, A. Merás, and N. Ben Amor. "Assessment for the mean value total dressing method: Comparison with coupled cluster including triples methods for BF, NO+, CN+, C2, BeO, NH3, CH2, H2O, BH, HF, SiH2, Li2, LiNa, LiBe+, NeH+, and O3". In: *The Journal of Chemical Physics* 107 (Oct. 1997), pp. 6306–6320. DOI: 10.1063/1.474293.

[133] B. Hammer, L. B. Hansen, and J. K. Nørskov. "Improved adsorption energetics within density-functional theory using revised Perdew–Burke–Ernzerhof functionals". In: *Physical Review B* 59.11 (1999), pp. 7413–7421. DOI: 10.1103/PhysRevB.59.7413.

[134] Y. Zhang and W. Yang. "Comment on "Generalized gradient approximation made simple"". In: *Physical Review Letters* 80.4 (1998), pp. 890–890. DOI: 10.1103/PhysRevLett.80.890.

[135] O. A. Vydrov and T. Van Voorhis. "Nonlocal van der Waals density functional: The simpler the better". In: *The Journal of Chemical Physics* 133.24 (2010), p. 244103.

[136] B. Deng, P. Zhong, K. Jun, J. Riebesell, K. Han, C. J. Bartel, and G. Ceder. "CHGNet as a pretrained universal neural network potential for charge-informed atomistic modelling". In: *Nature Machine Intelligence* (2023), pp. 1–11. DOI: 10.1038/s42256-023-00716-3.

[137] O. Materials. *ORB Reference Energies*. URL: https://github.com/orbital-materials/orb-models/blob/5a6ba14943c382c4f2c84c84bfa6401ce39355f6/orb_models/forcefield/reference_energies.py#L10.

[138] T. Iwashita, M. Nagao, A. Yoshimori, M. Terazima, and R. Akiyama. "Usefulness of higher-order system-size correction for macromolecule diffusion coefficients: A molecular dynamics study". In: *Chemical Physics Letters* 807 (2022), p. 140096. ISSN: 0009-2614. DOI: https://doi.org/10.1016/j.cplett.2022.140096. URL: https://www.sciencedirect.com/science/article/pii/S0009261422007539.

[139] S. Arrhenius. *Über die Dissociationswärme und den Einfluss der Temperatur auf den Dissociationsgrad der Elektrolyte*. Jan. 1889. DOI: 10.1515/zpch-1889-0408. URL: https://doi.org/10.1515/zpch-1889-0408.

[140] FAIR. *UMA SO(2) Convolutions Implementation*. URL: https://github.com/facebookresearch/fairchem/blob/0db89eaccd3ca4694efa5bf45d62c27fbf74fb86/src/fairchem/core/models/uma/nn/so2_layers.py.

[141] I. Langmuir. *The Adsorption of Gases on Plane Surfaces of Glass, Mica and Pt*. Nov. 1917. DOI: 10.1021/ja02242a004. URL: https://doi.org/10.1021/ja02242a004.