IMPROVING THE FIDELITY AND USABILITY OF MOLECULAR MODELS THROUGH HYBRIDIZATION AND MACHINE LEARNING TECHNIQUES

By

Mehmet Çağrı Kaymak

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2023

ABSTRACT

Molecular dynamics (MD) is a powerful computational method used to simulate the motion of atoms and molecules. MD simulations compute the evolution of a system of interacting particles by applying Newton's equations of motion, facilitating the study of a range of physical, chemical, and biological phenomena. While quantum mechanical (QM) simulations result in accurate predictions of geometries and energies essential for studying various phenomena, the computational complexity has led to the emergence of new approaches such as classical force fields, reactive force fields, and machine learning potentials (MLPs), each offering unique trade-offs. Classical force fields offer longer simulation times due to assumptions such as static bonds and charges, which prohibit the study of reactive systems. Reactive force fields, such as ReaxFF, bridge the gap between QM methods and classical force fields by allowing dynamic bonds and charges. The improved flexibility results in a higher computational load and a more complex functional form that is hand-crafted by domain experts. MLPs are a more recent approach that utilize large datasets to eliminate complex functional forms, while also leveraging the vast ecosystem of machine learning frameworks for enhanced computational efficiency and ease of development.

As the number of methodologies increases, the landscape of MD methods becomes more complex, with each method bringing unique attributes and challenges in simulating molecular systems. We introduce innovative hybridization techniques aiming to leverage the strengths of multiple modeling approaches, improving predictive capabilities and computational efficiency. We introduce a hybrid modeling approach called ReaxFF/AMBER that combines the reactivity and polarization capabilities of ReaxFF with the efficiency of classical force fields, facilitating the simulation of larger reactive regions. Although ReaxFF can offer high fidelity when trained carefully, the existing parameterization tools lack the efficiency and speed essential for creating new ReaxFF parameter sets for different applications of interest. We have proposed a novel parameter optimization approach, JAX-ReaxFF, leveraging the capabilities of a scalable machine learning framework to drastically reduce the training times

for ReaxFF, thus enhancing the development of new force fields for various applications. We have also modified JAX-ReaxFF to run end-to-end differentiable simulations on different architectures such as CPUs, GPUs, or TPUs with the help of JAX. JAX is a library known for high-performance numerical computing and it provides features such as automatic differentiation and optimization of Python functions. This approach also allows for improved integration with existing machine learning software infrastructure, offering enhanced flexibility and performance portability.

Lastly, we propose and compare various uncertainty quantification (UQ) methods suitable for MLPs. These methods are essential for active learning-based data generation approaches, which are crucial for training data-intensive machine learning models. While our primary focus is on MLPs, the datasets created using active learning methods could also enhance the parameterization efforts for classical and reactive force fields.

This thesis is dedicated to my family.

ACKNOWLEDGEMENTS

Reaching this milestone would not have been possible without the support of many. I want to take this moment to express my gratitude to my PhD advisor, Hasan Metin Aktulga. His valuable feedback and guidance have been instrumental throughout my PhD journey. His constant support and understanding, especially during challenging times, were greatly appreciated.

Besides my advisor, I want to thank my committee members for their valuable feedback and time. I am also grateful to my fellow graduate students for the collaborative exchange of ideas and camaraderie, which have significantly enhanced and brightened our shared path.

Finally, my profound and heartfelt gratitude is due to my parents, Ayhan and Feride Kaymak, for their endless encouragement. I must also thank my brother Kürşat for being there whenever I need him and adding joy to this journey, even from afar.

The published works presented in this work are supported in part by a NIH grant (Award No. GM130641), a NSF CDS&E grant (Award No. 1807622) and a NSF OAC grant (1835144). Computational resources provided by the Institute for Cyber-Enabled Research at Michigan State University were utilized for the computations.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background and Related Work

N-body simulation methods focus on the simulation and analysis of the dynamic interactions between multiple discrete entities in a complex physical or abstract system. These methods find applications in a wide range of scientific disciplines, including astrophysics, molecular dynamics, and fluid dynamics, among others. N-body computational techniques involve the numerical approximation of the forces and motions governing these interacting bodies, allowing researchers to gain insights into the behavior and evolution of complex systems, from celestial bodies in space to molecules in a chemical reaction. Simulation techniques address the challenge of solving equations for multiple interacting bodies by breaking down time into discrete intervals and integrating the behavior of individual particles over these discrete time steps. In this approach, particles are simplistically considered as point masses and interact with each other according to predefined potential functions. This field plays a pivotal role in understanding and predicting the behavior of systems with multiple interacting components, offering valuable insights into natural phenomena and facilitating advancements in various scientific and engineering domains.

Atomistic simulation methods are a suite of computational techniques where individual atoms and molecules are the main entities to be simulated and analyzed. By numerically solving the equations governing atomic interactions, atomistic simulations provide a means to explore various phenomena, such as chemical reactions, phase transitions, and mechanical properties, with high precision and detail. Various atomistic simulation methods are developed, including quantum mechanical methods, classical force fields, reactive force fields, and machine learning potentials, each tailored to address specific research questions and modeling requirements. Fig. 1.1 compares different simulations methods in terms of computational complexity and accuracy.

Coarse-grained simulations are computational models used in molecular and biological

studies where groups of atoms are simplified into single entities. This approach allows the exploration of larger systems and longer timescales, making it useful for studying macroscopic properties and behaviors such as protein folding and polymer dynamics. However, due to its simplified nature, some atomic-level details are omitted, making it essential to combine coarse-grained simulations with other detailed modeling techniques for a comprehensive understanding.

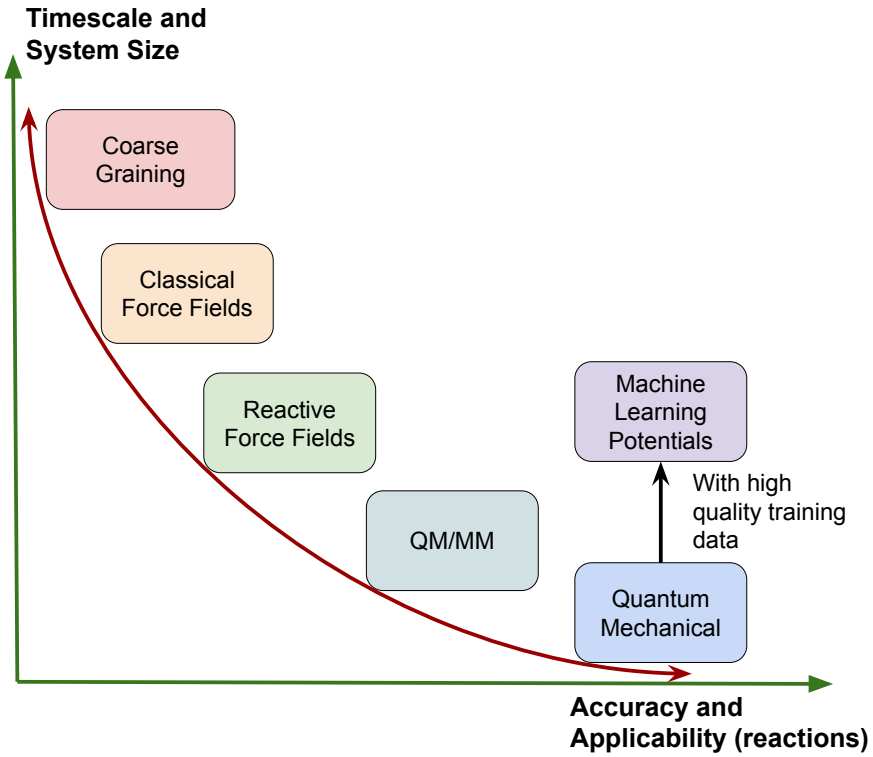


Figure 1.1 Simulation methods.

1.1.1 Quantum Mechanical Methods

Quantum Mechanical (QM) simulations enable the accurate prediction of geometries and energies by solving the Schrödinger's equation. QM methods, often referred to as *ab-initio* methods, start with first-principles quantum mechanics and minimize approximations when deriving solutions. Consequently, due to their modeling fidelity, *ab-initio* techniques generally yield outcomes that are more precise and dependable compared to atomistic methods that rely on various approximations as it is discussed in later sections.

One of the earliest *ab-initio* approach is the Hartree-Fock/self-consistent field (HF/SCF) method. At its core, the Hartree-Fock method dissects the complex multi-electron wave function of a molecular system into a collection of simpler one-electron wave functions known as molecular orbitals. In subsequent years, new advanced quantum mechanical methods have been developed. These include second-order Møller-Plesset perturbation theory (MP2), coupled cluster with single, double, and triple perturbative excitations (CCSD(T)), second-order perturbation theory within the complete active space framework (CASPT2), and approaches based on density functional theory (DFT) [24].

1.1.2 Classical Force Fields

It is important to note that *ab-initio* methods, while highly accurate, can be computationally intensive and are primarily suited for small to moderately sized systems with shorter simulation times due to their computational cost. To address this limitation and extend simulations to larger and more complex systems, researchers often transition to force field-based approaches. These approaches treat the atomic nucleus and its electrons as a unit particle and the interactions between atoms are governed by a force field (FF). This force field is a set of parameterized mathematical equations that capture atomic interactions such as bonds, valence angles, torsion, van der Waals, and Coulomb interactions. These simplifications greatly reduce the overall computational cost, but an important measure of the predictive power of force fields is their fidelity, i.e., how well they can reproduce the results of QM calculations and experimental studies. Development of high fidelity force fields relies heavily on optimization of various force field parameters based on carefully selected quantum-chemical and experimental reference data. With the help of these approximations and careful training, molecular dynamics (MD) methods have proven to be successful in atomistic simulations with billions of degrees of freedom [48].

MD methods are based on Newton's equations of motion, which describe the motion of a particle in terms of its mass, position, and the forces acting upon it. By repeatedly integrating these equations over successive time steps, MD simulations are able to compute

the evolution of a system of interacting particles. While one could argue that MD simulations might significantly differ from a real system's actual trajectory due to simplifications, approximations, and numerical errors, MD methods are grounded in the fundamental principle of classical statistical mechanics: the ergodic hypothesis. This hypothesis states that all states with a specified energy, volume, and number of particles are equally likely to be explored over time by the system [102]. As a result, although one cannot expect MD simulations to precisely replicate real trajectories, they allow to create ensembles of snapshots representing physically achievable system configurations. Hence, this allows researchers to apply concepts from statistical thermodynamics to investigate time-averaged properties such as density, temperature, and free energies.

Classical MD models as implemented in highly popular MD software such as AMBER [15], LAMMPS [103], GROMACS [40] and NAMD [72] are based on the assumption of static chemical bonds and, in general, static charges. These simplifications enable longer simulations that have been proven to be essential for applications such as drug discovery, protein folding and function [64]. In addition to force field development, there have been efforts to create custom-designed application-specific integrated circuits (ASICs) to extend simulation timescales beyond what was previously attainable [93, 92].

1.1.3 Reactive Force Fields

Classical force fields are not applicable to study phenomena where chemical reactions and charge polarization effects play a significant role. To address this gap, reactive force fields (e.g., ReaxFF [108], REBO [13], Tersoff [101]) have been developed. These bond order potentials allow bonds to form and break throughout the simulation and they can dynamically assign partial charges to atoms using suitable charge models such as the electronegativity equalization method (EEM) [65]. The functional forms for reactive potentials are significantly more complex than their classical counterparts due to the presence of dynamic bonds and charges. For instance, ReaxFF has a formulation that contains more than 100 parameters for a simulation with 3 elements, and is about two orders of magnitude more expensive

than a typical Lennard-Jones potential. Consequently, training reactive force fields is an even more difficult task due to the need to capture complex phenomena such as charge distributions and reactions, and due to the large number of parameters involved.

The ReaxFF potential is an excellent candidate for modernization and hybridization because it supports dynamic charges and bond breaking, unlike classical force fields. With its complex functional form, ReaxFF can express a wide variety of biological and chemical phenomena when properly parameterized [88]. Since it is an important building block for the further chapters, we provide further details about the ReaxFF potential in the following paragraphs.

1.1.3.1 ReaxFF Overview

ReaxFF divides the total potential energy into various parts, including bonded and non-bonded interactions as shown in Eq. (1.1). The model takes atom coordinates and required force field parameters for the set of elements present in the system as input, and calculates all terms constituting the potential energy together with the corresponding forces. The derivative of each potential energy term with respect to atom positions gives forces that are fundamental to the MD simulation. There are a number of ReaxFF implementations with different features and architectural support such as the original Fortran Reax code [108], PuReMD [2, 3, 53], GULP [30] and LAMMPS [73].

$$\begin{aligned} E_{\text{system}} = {} & E_{\text{bond}} + E_{\text{lone-pair}} + E_{\text{over}} + E_{\text{under}} \\ & + E_{\text{val}} + E_{\text{pen}} + E_{\text{tors}} + E_{\text{conj}} \\ & + E_{\text{Hbond}} + E_{\text{vdWaals}} + E_{\text{Coulomb}} \end{aligned} \tag{1.1}$$

An important aspect of ReaxFF that separates it from classical MD models are the notions of bond orders and dynamic partial charges (not shown in Eq. (1.1)). The bond order concept is used to determine the bond strength between pairs of atoms given their element types and distances. These pairwise bond orders are then subjected to corrections that take into account the information about all atoms surrounding each atom to obtain the predicted bonding information in a system. The corrected bond order constitutes the main input for

common potential energy terms such as bond energy (E_{bond}), valence angle energy (E_{val}), and torsion angle energy (E_{tors}). However, in a dynamic bonding model, since atoms may not attain their optimal coordinations, additional terms such as lone pair ($E_{\text{lone-pair}}$), over/under-coordination ($E_{\text{over}}, E_{\text{under}}$), three-body penalty ($E_{\text{pen}}$), and four-body conjugation (E_{conj}) energies are needed. For systems with hydrogen bonds, a special energy term (E_{Hbond}) is used. The van der Waals energy (E_{vdWaals}), which is based on the Morse potential, and the electrostatic energy term (E_{Coulomb}), which uses shielded and range-limited interactions based on dynamic charges calculated from charge models such as EEM [65], constitute the non-bonded terms in ReaxFF. Typically, bonded interactions are truncated at 5 Å, hydrogen bonds are effective up to 7.5 Å and non-bonded interactions are range limited to 10-12 Å depending on the system. Fig. 1.2 summarizes the calculations performed within a ReaxFF step.

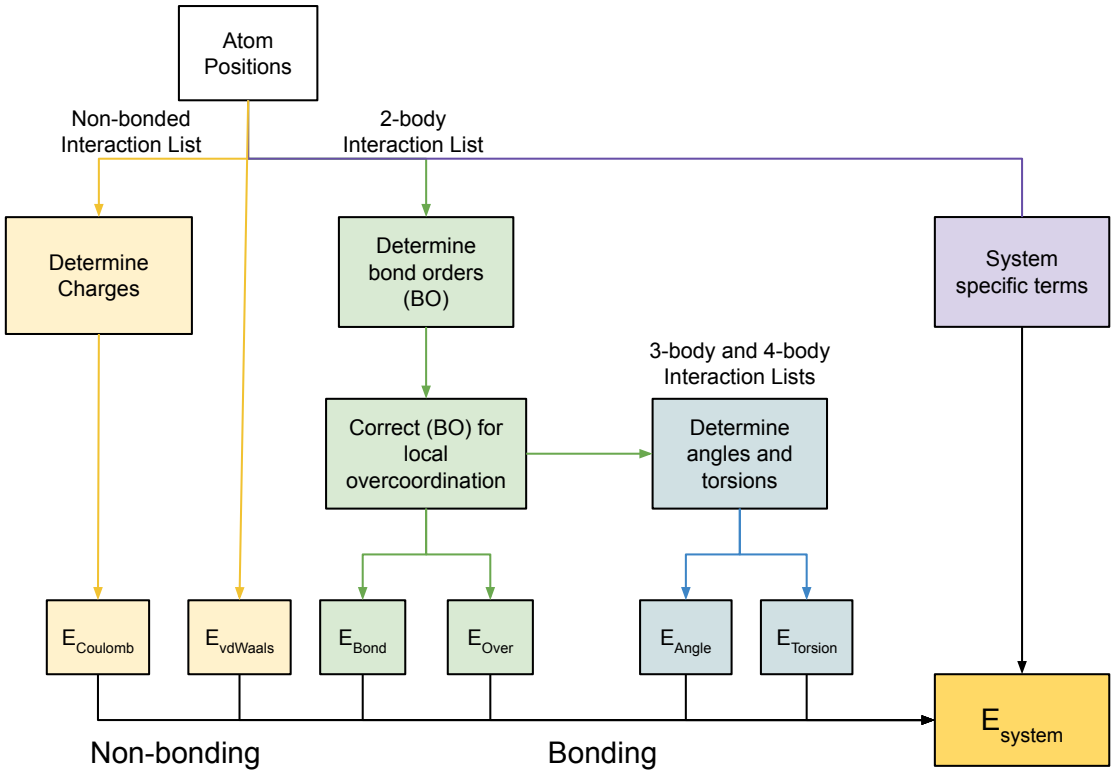


Figure 1.2 Flow graph of calculations performed in the Reax force field.

1.1.4 Machine Learning Potentials

The success of ML techniques in fields such as computer vision and natural language processing has triggered its wide-spread use also in scientific computing. More recently, we started witnessing an increase in the number of scientific applications adopting ML libraries such as Tensorflow [1], PyTorch [70], and JAX [11]. Such tools have enabled fast prototyping of new ideas as well as hardware portability without sacrificing much computational efficiency. Specifically, in molecular modeling and simulation, a new class of force fields called machine learning potentials (MLP) such as SNAP [104], the Behler/Parrinello potential [9], SchNet [86], OrbNet [76], and NequIP [8] has emerged. While the increased number of parameters and improved learning capacity of MLPs bring the accuracy of MD simulations closer to the QM level, the training data quality becomes critical for the quality of the MLPs. Although the classical and reactive MD models also require high quality training data and careful parameterization, due to the embedded physical bias, they typically require less data than their ML based counterparts.

Typically, MLPs encode the local neighborhood information of each atom into a latent vector, compressing and summarizing the local information. Subsequently, the latent vector is mapped to observables, typically energies or charges. The sum of the individual atom energies yields the total molecular energy, as shown in Eq. (1.2) where $E_{i,\text{ atomic}}$ represents the local atomic energy for atom i, E_{pot} is the total energy of the molecule with N atoms. The forces ($\vec{F_i}$) needed to drive the simulation are determined by taking the derivative of the total energy with respect to the atomic positions (thereby guaranteeing energy conservation).

$$E_{pot} = \sum_{i \in N} E_{i,\text{ atomic}} \tag{1.2}$$

$$\vec{F_i} = -\nabla_i E_{pot} \tag{1.3}$$

While it is possible to directly predict the total energy, representing it as a sum of individual atomic energies simplifies the training process and also makes computations easily

parallelizable, as calculations for each atom are independent of one another.

MLPs differ in how they encode local information. Fingerprint-based MLPs, like the Behler/Parrinello potential, rely on handcrafted descriptors, while more recent approaches utilize learnable feature extractors. Message Passing Neural Networks (MPNNs) offer a versatile framework grounded in graph-based deep learning. MPNNs represent molecular structures as graphs, with atoms as nodes and bonds as edges, utilizing message-passing algorithms to update node representations by aggregating information from neighboring nodes. This flexibility makes MPNNs well-suited for a range of molecular modeling tasks, including property prediction and generative modeling. Equivariant Neural Networks, meanwhile, are designed to respect the symmetries inherent in input data, making them particularly valuable for systems with spatial or rotational symmetries. They ensure that network predictions remain invariant or equivariant under specific transformations, such as rotations or translations. A study by Batzner et al. [8] demonstrates that, when carefully trained, they exhibit higher data efficiency, as they inherently account for the data's symmetries without requiring additional data or training.

The ANI potential [96], an early MLP, played a pivotal role in popularizing machine learning for molecular modeling and simulation. It relies on handcrafted local descriptors, building upon the Behler/Parrinello potential. Due to its widespread adoption and availability as an open-source implementation (TorchANI [33]), ANI serves as an ideal candidate to demonstrate new approaches. In Chapter 5, we showcase and compare various uncertainty quantification methods for MLPs using ANI as an illustrative example. Further details about the ANI architecture are provided in the subsequent section.

1.1.4.1 ANI Overview

ANI, unlike traditional force fields, employs artificial neural networks to define potential energy as an explicit function of atomic coordinates, without predefined bond concepts. ANI computes an atomic environmental vector (AEV) for each atom, which is invariant to translation and rotation. Using an atom type specific feed forward neural network, these

vectors are then mapped to a scalar value representing the atomic contribution to the total energy. The sum of these energies gives the total molecular energy as shown in Eq. (1.2).

AEVs contain radial and angular sections to vectorize the local environment. The local radial environment of atom i is encoded by utilizing Eq. (1.4) where η is used to control the width of the Gaussian distribution and R_s is used to shift the peak. R_{ij} is the distance between atoms i and j. To better capture the radial local environment, multiple η and R_s values are used. In Eq. (1.4), m is used as the index over a set of η and R_s values. Finally, individual $G^R_{i,m}$ values are concatenated to create the radial part of the AEV.

$$G^{\mathrm{R}}_{i,m} = \sum_{j\neq i}^{\text{all atoms}} \mathrm{e}^{-\eta\left(R_{ij}-R_{\mathrm{s}}\right)^2} f_{\mathrm{C}}\left(R_{ij}\right) \tag{1.4}$$

To limit the size of the local region, neighbors beyond a selected cutoff (R_C) are ignored. To make the transition around the cutoff region continuous, a piece-wise cutoff function is utilized (f_C).

$$f_{\mathrm{C}}\left(R_{ij}\right) = \begin{cases} 0.5 \times \cos\left(\frac{\pi R_{ij}}{R_{\mathrm{C}}}\right) + 0.5 & \text{for } R_{ij} \leq R_{\mathrm{C}} \\ 0.0 & \text{for } R_{ij} > R_{\mathrm{C}} \end{cases} \tag{1.5}$$

For the angular features of atom i, Eq. (1.6) is used where θ_{ijk} is the angle between atoms i, j and k, centered on atom i. $G^{\mathrm{A_{mod}}}_{i,m}$ is basically in the form of a sum over all neighboring atom pairs (j and k). While η and R_s work similar to (1.4), θ_s allows probing different regions of the angular environment by shifting the angle term. ζ controls the width of the peaks in the angular environment. f_C multipliers are used to make the function differentiable around the cutoff region. m is used as the index over a set of η, R_s, ζ and θ_s values. Finally, individual $G^{\mathrm{A_{mod}}}_{i,m}$ values are concatenated to create the angular part of the AEV.

$$\begin{aligned} G^{\mathrm{A_{mod}}}_{i,m} =& 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} \left(1 + \cos\left(\theta_{ijk} - \theta_{\mathrm{s}}\right)\right)^{\zeta} \\ & \times \exp\left[-\eta\left(\frac{R_{ij}+R_{ik}}{2} - R_{\mathrm{s}}\right)^2\right] f_{\mathrm{C}}\left(R_{ij}\right) f_{\mathrm{C}}\left(R_{ik}\right) \end{aligned} \tag{1.6}$$

The AEV of a given atom distinguishes between atom types by incorporating unique radial components for each neighboring atom type and unique angular components for each neighboring atom type pair, ensuring accurate representation of the atomic environment.

1.2 Contributions of This Thesis

In this thesis, we introduce new techniques that aim to improve the fidelity and usability of molecular models through hybridization and machine learning techniques.

As introduced earlier, there are various modeling approaches with their unique trade-offs. While simulating the whole system with the desired method is straightforward, some applications require different predictive capabilities and considerations such as polarizability and/or reactivity. Combining multiple modeling approaches in a hybrid framework could help researchers leverage the capabilities of multiple approaches. Mixed quantum mechanics/molecular mechanic (QM/MM) [25] methods help combining classical force fields such as AMBER with QM methods when the apriori knowledge of the reactions in a system of interest and spatially localized reactivity is available. Although this opens up new possibilities, QM approaches are still computationally infeasible when the reactive region is large or when long simulated time scales are necessary for the chemistry of interest. In Chapter 2, we propose a new solution called ReaxFF/AMBER that combines a reactive force field with a classical force field. As ReaxFF supports reactivity and polarization, it enables simulating larger reactive regions when a high fidelity force field is available for the system of interest. The same approach could be easily extended for the newly emerged MLPs.

While the AMBER/ReaxFF approach is proven to be successful when ReaxFF model is trained properly, the existing optimization frameworks lack the necessary speed that is essential to develop new force fields in a timely manner for new applications. The previous approaches rely on the existing ReaxFF implementations that are designed to run large simulations for long time windows. We propose a new parameter optimization approach called JAX-ReaxFF that utilizes an efficient and scalable machine learning framework called JAX. By leveraging auto-differentiation, auto-vectorization and just-in-time (JIT) compilation ca-

pabilities of JAX, we are able to reduce the training time for ReaxFF drastically. This is especially important when one wants to utilize large datasets that are crafted for more data-hungry MLPs. JAX-ReaxFF could easily leverage these datasets. Similar approach could also be used for classical force fields such as AMBER as demonsrated in [114].

Although JAX-ReaxFF is crafted specifically for parameter optimization tasks, a Python based implementation offers a portable, performant, and easy to maintain software. Also, Python is the main language for ML based MD tools, and this leads more ways to hybridize force fields similar to the ReaxFF/AMBER approach. As the cross-language communication between different libraries is troublesome, having different force fields and MD implementations in Python offers greater flexibility. With the auto-differentiation and performance portability capabilities of JAX, we present a new end-to-end differentiable and portable ReaxFF implementation that could be used for MD simulations, free energy calculations or parameter optimization. The new implementation is a part of a bigger JAX based MD library called JAX-MD, which already supports different ML and force field based simulations. This leads to better use of existing software infrastructure. As it is discussed in Chapter 4, the performance on modern GPUs is still on par with the existing software for ReaxFF. This work showcases the practicality of the ML infrastructure for complex force fields such as ReaxFF.

While we extend the capabilities of the chosen classical and reactive force fields by utilizing hybridization and the existing ML software infrastructure, the development of the MLPs lead to the need for bigger datasets and methods to create them. Moreover, unlike regular force fields with hand-crafted functional forms, MLPs typically utilize general building blocks for prediction which makes their performance harder to quantify. To remedy some of these issues, we propose and compare different uncertainty quantification (UQ) approaches suitable for MLPs in Chapter 5. As it gets harder to generate rich datasets for molecular modeling, active learning based approaches have gained popularity [74, 91, 97]. Since having a reliable, fast and preferably differentiable UQ metric is essential for active learning based

data generation, we evaluate different UQ metrics on various datasets using the ANI model as an example.

CHAPTER 2

REAXFF/AMBER—A FRAMEWORK FOR HYBRID REACTIVE/NONREACTIVE FORCE FIELD MOLECULAR DYNAMICS SIMULATIONS

The following chapter presents previously published work on adding reactive capabilities by utilizing ReaxFF in AMBER [77]. This work is reproduced with the permission of American Chemical Society.

As stated in Chapter 1, the AMBER MD software package is a widely used tool to study biological systems such as proteins. However, since it is a classical force field, it only supports static bonds and charges. This limits its abilities to simulate chemical reactions due to the rigid connectivity requirement. QM methods can mitigate these limitations such as formation or breaking of bonds and charge fluctuations due to geometry changes. However, QM models are usually applied to fragments of the regions involved in, say an enzymatic reaction, limiting the ability to explore the influence of environmental effects. Although QM based methods can be very accurate in predicting chemical reaction events, they remain limited to small systems simulated over short time scales. Hybrid quantum mechanics/molecular mechanics (QM/MM) methods were developed to combine the best features of empirical force fields (EFF) and QM models to tackle a range of chemical problems [90]. Reactive force fields represent the intermediate approach explored in this section. A hybrid ReaxFF/AMBER molecular dynamics (MD) tool is presented, which introduces ReaxFF capabilities to capture bond breaking and formation within the AMBER MD software package. The following sections motivate and discuss the results of this work.

2.1 Background and Motivation

In QM/MM methods, the total system is divided into two separate QM and MM zones. The QM zone is the chemically active region which is treated by a range of QM methods and the rest of the system is the MM zone which is treated using an EFF. Since the introduction of the QM/MM method, various approaches have been implemented, and this method has

found extensive applications to deal with complex systems in realistic environments because of the significant reduction in the computational cost compared to pure QM methods [90, 25, 36, 62, 61]. Different QM/MM simulation tools have long been supported in the AMBER MD package [71, 16]. Some QM methods including semiempirical neglect of diatomic overlap (NDDO)-type and density functional tight binding (DFTB) are built-in (i.e., internal) and are supported natively within AMBER [112, 87]. More advanced QM methods are supported via a file-based integration interface to external QM software packages [35, 43].

Despite highly innovative techniques, algorithmic improvements and fast implementations, the computational cost of the QM region still stands as the rate limiting factor in QM/MM simulations. ReaxFF is based on the bond length/bond order concept that bridges QM and MM methods in terms of functionality and computational costs. Importantly, ReaxFF provides a reasonable approximation of reactive phenomena at computational costs comparable to MM methods. The hybrid ReaxFF/AMBER molecular dynamics (MD) tool enables us to study local reactive events in large systems at a fraction of the computational costs of QM/MM models. Another major challenge with atomistic simulations is that chemical reactions through transition states can take place on a time scale that cannot be reached by regular molecular dynamics simulations. Therefore, approaches based on enhanced sampling methods are used to locate the transition state for a chemical reaction using QM/MM methods. Umbrella sampling [49] is one of the most well-known enhanced sampling methods and can readily be used in the new ReaxFF/AMBER tool. It is used herein to map out the reaction profile of the Claisen rearrangement as a validation study.

2.2 The ReaxFF/AMBER Integration

Similar to QM/MM methods, atoms are split into three categories in the ReaxFF/AMBER method: (i) ReaxFF atoms, which include all atoms in the chemically reactive region and are handled by a ReaxFF implementation, (ii) the ReaxFF/MM transition atoms, which include all atoms within a certain cutoff of the ReaxFF region and is handled by ReaxFF and AMBER collaboratively, and (iii) the MM atoms, which include all remaining atoms and is

handled exclusively by AMBER. These categories are illustrated in Fig. 2.1 with sphere-like shapes, but these regions can obviously be of any shape. In what follows, we describe the implementation of the ReaxFF/AMBER method.

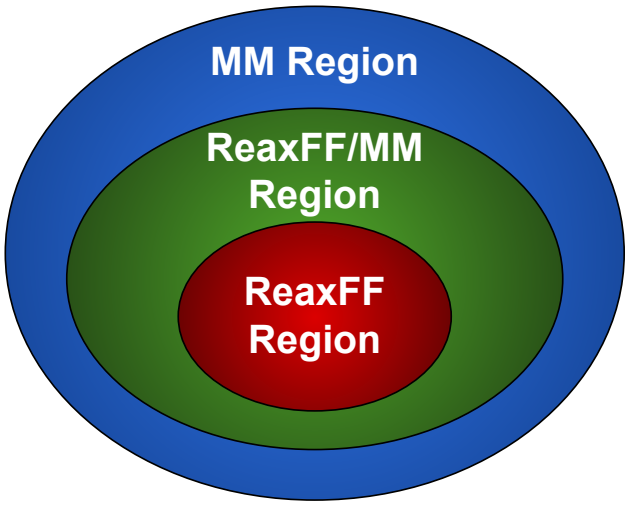


Figure 2.1 ReaxFF/AMBER regions in the integration implementation.

2.2.1 Implementation

AMBER is the simulation driver in the ReaxFF/AMBER MD integration. After AMBER categorizes atoms into their respective groups, it sends all relevant information for ReaxFF and ReaxFF/MM atoms to the ReaxFF program. The ReaxFF/AMBER tool currently uses the external model interface; that is, the necessary data transfers between ReaxFF and AMBER are performed using file-based data exchange. Therefore, after AMBER completes writing the data exchange files, it launches the ReaxFF program as an external binary and waits for its output files. The ReaxFF program then runs a zero-step nonperiodic simulation to calculate the dynamic charges, energies, and forces on the ReaxFF and ReaxFF/MM atoms. Upon completion, it writes this information back into another file which is finally read by AMBER to complete the energy and force computations for the ReaxFF/MM and MM regions.

In implementing the interface between the ReaxFF and AMBER programs, we have adopted the following procedure for a successful hybrid model:

- Dynamic charges on ReaxFF atoms are calculated under the influence of ReaxFF/MM atoms with static charges.
- All ReaxFF interactions between ReaxFF-ReaxFF pairs are calculated without any modifications.
- Electrostatic interactions between ReaxFF (w/dynamic charge)-ReaxFF/MM (w/static charge) atom pairs are calculated by ReaxFF.
- van der Waals interactions between ReaxFF-ReaxFF pairs/MM atom pairs are handled by AMBER (e.g., using a Lennard-Jones potential).
- Interactions between MM-ReaxFF/MM and MM-MM pairs are handled by AMBER as usual.

We should note that there are some limitations of the current ReaxFF/AMBER tool. Now, only systems with noncovalent bonds between ReaxFF and ReaxFF/MM regions can be studied. Also, only shared memory parallelism can be leveraged for the time-being. Nevertheless, as we demonstrate below, the current implementation serves as a proof-of-concept on the feasibility and advantages of this approach.

2.2.2 Dynamic Charges with the Modified Electronegativity Equalization Method (mEEM)

Since we allow the statically charged MM and transition region atoms to polarize the ReaxFF atoms, the dynamic charge model used by ReaxFF needs to be modified. As mentioned previously, ReaxFF can use the charge equilibration (QEq) or electronegativity equalization method (EEM) to determine the charges. Our current implementation is based on EEM. Before describing the necessary modifications, we briefly discuss the EEM charge model that is currently used by ReaxFF/AMBER. The EEM charge model [65] relies on the principle that charges should be distributed on atoms to satisfy constraints for both the net system charge and the equalized atom electronegativities. Let atomic charges be $\mathbf{q} = (q_1, q_2, \ldots, q_n)$ and the positions be $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n)$, where $q_i \in \mathbb{R}$ and $\mathbf{r}_i \in \mathbb{R}^3$. On

the basis of Sanderson's Electronegativity Equalization Principle [83], the electronegativity of all atoms needs to be equalized:

$$\epsilon_1 = \epsilon_2 = \cdots = \epsilon_n = \bar{\epsilon} \tag{2.1}$$

where ϵ_i is the electronegativity of atom i and $\bar{\epsilon}$ is the average molecular electronegativity. The other constraint forces the sum of the atomic charges to be equal to the given net system charge:

$$\sum_i q_i = q_{\text{net}}. \tag{2.2}$$

The constraints and the parameterized inter-atomic interactions can be merged into the following linear equation where the charges q are the solution of:

$$\begin{bmatrix} H & \mathbf{1}_n \\ \mathbf{1}_n^T & 0 \end{bmatrix} \begin{bmatrix} q \\ \bar{\epsilon} \end{bmatrix} = \begin{bmatrix} -\chi \\ q_{\text{net}} \end{bmatrix}. \tag{2.3}$$

Here, χ is an n by 1 vector of atomic electronegativities, and the H_{ij} values, that is, individual elements of the n by n H matrix, are defined as $\delta_{ij}\eta_i + (1-\delta_{ij}) \cdot F_{i,j}$, where δ_{ij} is the Kronecker delta operator, η_i is the idempotential, and F_{ij} is defined as

$$F_{i,j} = \begin{cases} \frac{1}{\sqrt[3]{r_{ij}^3 + \gamma_{ij}^{-3}}}, & r_{ij} \leq r_{\text{nonb}} \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

In the equations above, $r_{ij} = ||\mathbf{r}_j - \mathbf{r}_i||_2$ is the distance between atoms i and j, r_{nonb} is the cutoff radius, $\gamma_{ij} = \sqrt{\gamma_i \cdot \gamma_j}$ is a pairwise shielding term tuned for element types of atoms i and j to avoid unbounded electrostatic energy at short distances, and $\bar{\epsilon}$ is the dielectric constant of the medium. It has been demonstrated that the EEM model reproduces QM calculated Mulliken charges [14].

To account for the polarization effect of the transition region atoms on the core ReaxFF region, we introduce the ReaxFF/AMBER atoms as particles with fixed charges to the EEM

solver. This is done by modifying the system of equations as follows. Assuming that there are c ReaxFF atoms and t transition region atoms, then $H_{core} \in \mathbb{R}^{c\times c}$ captures the interactions within the core ReaxFF region and $H_{core-trans} \in \mathbb{R}^{c\times t}$ captures the impact of the transition region atoms on the ReaxFF atoms.

$$\begin{bmatrix} H_{core} & H_{core-trans} & \mathbf{1}_c \\ H_{trans-core} & H_{trans} & \mathbf{1}_t \\ \mathbf{1}_c^T & \mathbf{1}_t^T & 0 \end{bmatrix} \begin{bmatrix} q_{core} \\ q_{trans} \\ \bar{\epsilon} \end{bmatrix} = \begin{bmatrix} -\chi_{core} \\ -\chi_{trans} \\ q_{\text{net}} \end{bmatrix} \tag{2.5}$$

Since a direct solution of the above linear system scales cubically with the number of total atoms, typically iterative solvers are used to obtain approximations to the optimized dynamic charges. In such an iterative scheme, we only evolve the charges on the core atoms as q_{trans} are fixed values given in the MM force field. As such, the rows corresponding to the transition region atoms can be ignored in the above linear system, and we obtain

$$\begin{bmatrix} H_{core} & H_{core-trans} & \mathbf{1}_c \\ \mathbf{1}_c^T & \mathbf{1}_t^T & 0 \end{bmatrix} \begin{bmatrix} q_{core} \\ q_{trans} \\ \bar{\epsilon} \end{bmatrix} = \begin{bmatrix} -\chi_{core} \\ q_{\text{net}} \end{bmatrix}. \tag{2.6}$$

Since q_{trans} is fixed, it can be rearranged as

$$\begin{bmatrix} H_{core} & \mathbf{1}_c \\ \mathbf{1}_c^T & 0 \end{bmatrix} \begin{bmatrix} q_{core} \\ \bar{\epsilon} \end{bmatrix} = \begin{bmatrix} -\chi \\ q_{\text{net}} \end{bmatrix} - \begin{bmatrix} H_{core-trans} \\ \mathbf{1}_t^T \end{bmatrix} \begin{bmatrix} q_{trans} \end{bmatrix}. \tag{2.7}$$

These modifications ensure that the existence of the fixed charges does not slow down the charge equilibration step since the fixed charges and dynamic charges are separated.

The modifications above are sufficient to run nonperiodic ReaxFF/MM simulations, as well as periodic boundary simulations without long-range electrostatic interactions. When running periodic boundary simulations with long-range electrostatic interactions AMBER uses the particle mesh Ewald (PME) approach which calculates interactions within a certain cutoff distance directly and approximates the rest over a mesh [20]. To be able to account for

the long-range interactions in EEM, we have incorporated the impact of the mesh points on ReaxFF atoms by further extending the EEM matrix, that is, by adding a column and row containing the effects of the mesh points. This scheme does not conserve the energy in NVE simulations with periodic boundaries because ReaxFF uses tapered electrostatic interactions [18] that force the Coulomb effects to slowly decay to 0 at the interaction cutoff radius which is typically set to 10-12 Å. Tapering of the Coulomb interactions is a built-in feature of the ReaxFF formulation, and existing ReaxFF parameter sets have been trained with this design principle in mind. We believe that by retraining ReaxFF parameters without tapered Coulomb kernels, periodic boundary simulations with PME can successfully be enabled, but doing so goes beyond the scope of this paper, so we leave it as a topic for further exploration.

2.2.3 Validation

For validation of the resulting ReaxFF/AMBER combination, we performed experiments using benzene-in-water systems, one with periodic boundary conditions, and one without (Fig. 2.2). Benzene is not reactive in water, but the goal of these computational experiments was to validate that the ReaxFF/MM method can achieve energy conservation and produce reasonable dynamic charges.

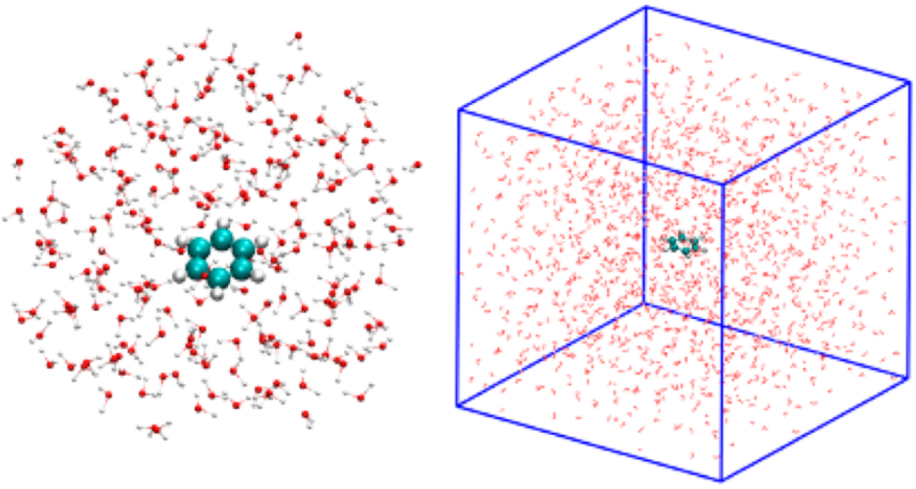


Figure 2.2 (Left) ReaxFF Benzene in a TIP3P water droplet with a total of 627 atoms, (right) ReaxFF Benzene in a TIP3P water box with a total of 4398 atoms.

For both simulations (nonperiodic and periodic), the systems are first energy-minimized

and then heated to 300 K using the Berendsen thermostat in AMBER. Finally, ReaxFF/MM NVE simulations are run using a time step of 0.25 fs to check energy conservation and charges. A relatively short time step was chosen as this is the recommended setting for ReaxFF simulations, especially in the presence of H atoms. For all simulations, the SHAKE algorithm was turned off.

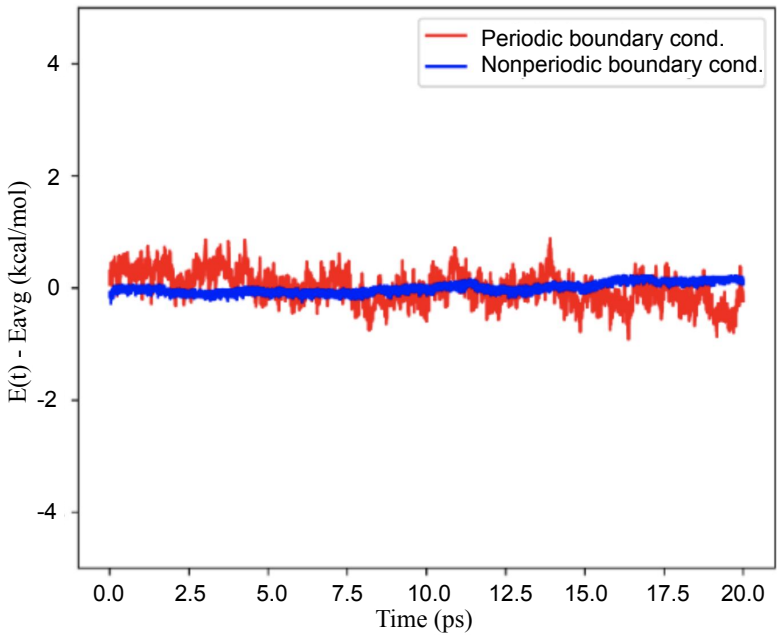


Figure 2.3 Simulation of a benzene molecule in water using the TIP3P water model and NVE ensemble. A time step of 0.25 fs was used. (Blue) Nonperiodic boundary condition with an infinite cutoff. (Red) Periodic boundary condition with a QM cutoff of 10 Å (qmcut = 10 Å) and ReaxFF/PME interaction turned off (qm_pme = 0).

As shown in Fig. 2.3, the energy is conserved for both simulations. Fig. 2.4 shows the carbon and hydrogen charges for Reax atoms (denoted as Reax-C and Reax-H, respectively), as well as the average across all C and H atoms in a time step (denoted as Reax-C-Avg and Reax-H-Avg, respectively). As can be seen, dynamic charges produced by ReaxFF/MM under the influence of statically charged transition region atoms are in line with the carbon and hydrogen charges given in the AMBER force field for benzene (denoted as AMBER-C and AMBER-H, respectively).

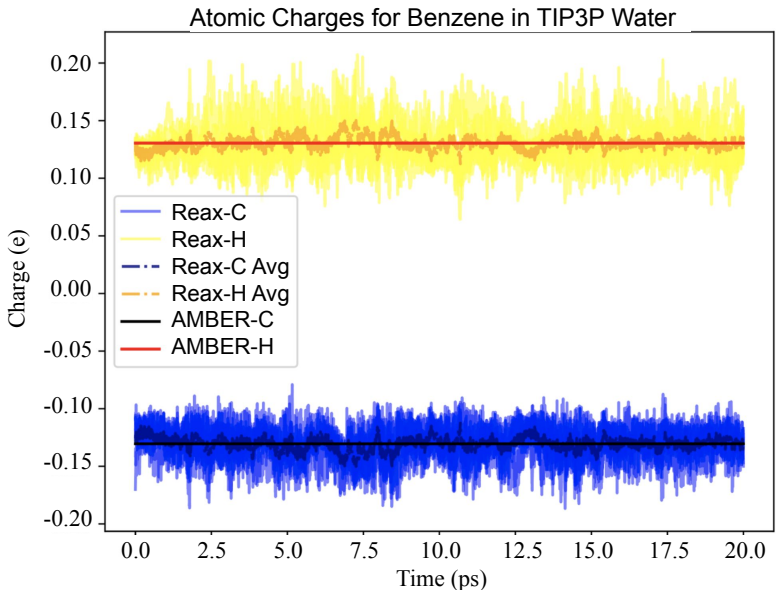


Figure 2.4 Dynamic ReaxFF charges and fixed AMBER charges with a ReaxFF/MM simulation of a benzene molecule in water with periodic boundary conditions using parameters qmcut = 10 Å and qm_pme = 0.

2.2.4 Performance Analysis

As mentioned before, the motivation for the ReaxFF/MM method is that it can significantly reduce the computational time that would be needed by a comparable QM/MM simulation. To illustrate this, we benchmarked the computational cost of separate equilibration simulations of allyl vinyl ether (AVE) which consists of 14 atoms using ReaxFF, PuReMD, SCC-DTFB, PM3, MM, the Hartree-Fock method (HF), and density functional theory (DFT) using the B3LYP functional. The HF and DFT calculations used the 6-31G* basis set. The benchmark simulations were performed using standalone ReaxFF, PuReMD, AMBER, and the QUICK software package [63] in sequential execution mode on an Intel Xeon E5-2670 v2 CPU which runs at 2.50 GHz. The benchmarked time per simulation steps are shown in Table 2.1.

In Table 2.1, PuReMD refers to a C language based efficient parallel implementation of the original ReaxFF implementation in Fortran. Time steps of 1 fs for MM, 0.5 fs for SCC-DFTB and PM3, and 0.25 fs for ReaxFF calculations are assumed for these benchmark calculations. These benchmark data show that ReaxFF method can be ∼2-5 times faster

Method	seconds/iteration (relative wrt MM)	time step (fs)	ns/day (relative wrt PuReMD)
SCC-DFTB	3.088×10^{-3} (39.8)	0.5	13.99 (0.2)
PM3	1.83×10^{-3} (23.6)	0.5	23.61 (0.4)
ReaxFF	6.25×10^{-4} (8.1)	0.25	34.56 (0.6)
PuReMD	3.63×10^{-4} (4.7)	0.25	59.50 (1.0)
MM	7.75×10^{-5} (1.0)	1.0	1114.84 (18.7)
HF	1.833 (23651.6)		
DFT	2.762 (35638.7)		

Table 2.1 Sequential execution time in seconds per time step and nanoseconds per day for SCC-DFTB, PM3, ReaxFF, PuReMD, MM, HF, and DFT calculations. B3LYP functional and the 6-31G* basis set were used for the HF and DFT geometry optimization calculations.

than PM3 or SCC- DFTB methods, 3 orders of magnitude faster than HF or DFT methods, and ∼18 times slower than conventional molecular mechanics methods. Therefore, ReaxFF can offer an alternative for QM/MM calculations to handle larger complex systems at longer time scales. Currently, ReaxFF/AMBER software uses the original Fortran ReaxFF code through a file-based data exchange interface. We analyzed the computational performance of the ReaxFF/AMBER software with the nonperiodic benzene-in-water simulation for 1000 steps using an infinite electrostatic interaction cutoff. The timing breakdown shown in Fig. 2.5 is averaged over the course of this simulation. Please see Section A.1 for specifications of the hardware and software used in these computational experiments.

Since the data exchange between ReaxFF and AMBER is file based, it requires the writing and reading of files at every simulation step. There are also the additional costs associated with launching the ReaxFF program as an external binary. Finally, the modifications to the standalone ReaxFF implementation is minimal and for that reason, ReaxFF allocates and initializes various data structures at every time step. Time spent on the I/O and system calls to the external program will be removed by introducing array-based data exchanges between the relevant ReaxFF and AMBER subroutines. Additionally, by keeping the data structures in memory persistently and reusing them at each time step, we could eliminate most of the time spent on the allocation and initialization parts. When all these potential

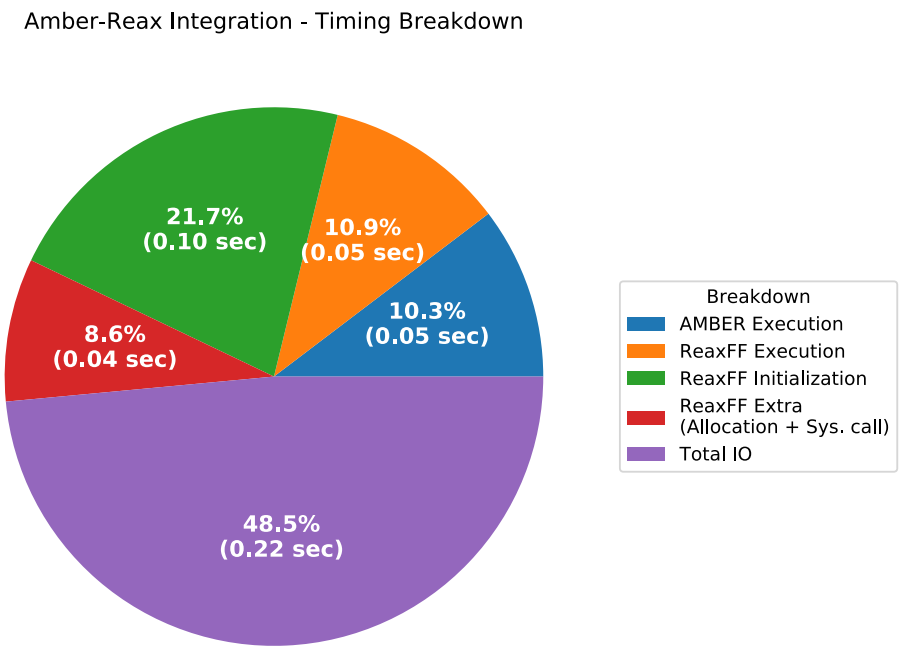


Figure 2.5 Breakdown of the total execution time of ReaxFF/AMBER software for the benzene-in-water simulation into its main components. The total time spent per step is 0.46 sec.

updates are considered, the ReaxFF/AMBER software would be accelerated significantly beyond what is shown here because only 10.9% of the overall time is spent on the necessary ReaxFF calculations (see Fig. 2.5).

2.3 Claisen Rearrangement Simulations with ReaxFF/Amber

To demonstrate the capabilities of the new ReaxFF/AMBER integration, we carried out ReaxFF/MM modeling of the classic Claisen rearrangement of allyl vinyl ether (AVE) solvated in an explicit TIP3P water model [47] as shown in Fig. 2.6. Since the Claisen rearrangement is a well-studied reaction [31, 32], it was chosen to evaluate the chemical accuracy of our new method. In these simulations, the solvent is treated by AMBER using the TIP3P water model and the reactant AVE is treated with ReaxFF. No covalent bonds cross the ReaxFF/AMBER boundary.

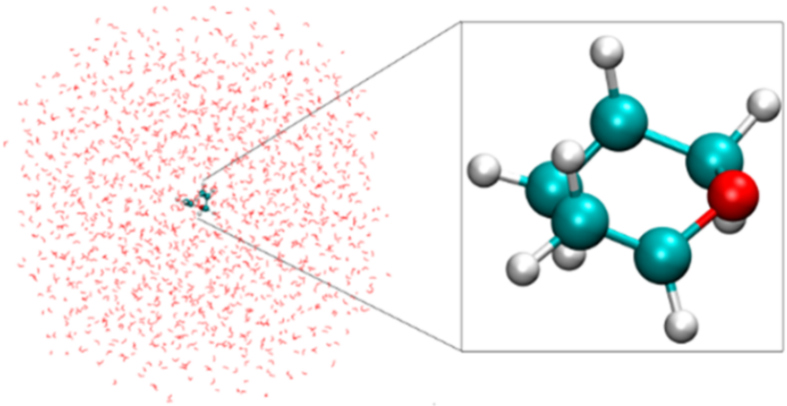


Figure 2.6 (left) Solvation of allyl vinyl ether (AVE) in octahedral TIP3P water with total number of 4319 atoms. (right) Molecular structure of AVE.

2.3.1 ReaxFF Parameter Optimization

The existing ReaxFF force field [117] was tested for simulating the Claisen rearrangement. The initial tests showed that the system got trapped in a local minimum after reaching the transition state (TS), and therefore a proper Claisen rearrangement could not be observed (Fig. 2.7). To resolve this, ab initio QM data for the Claisen rearrangement of AVE in

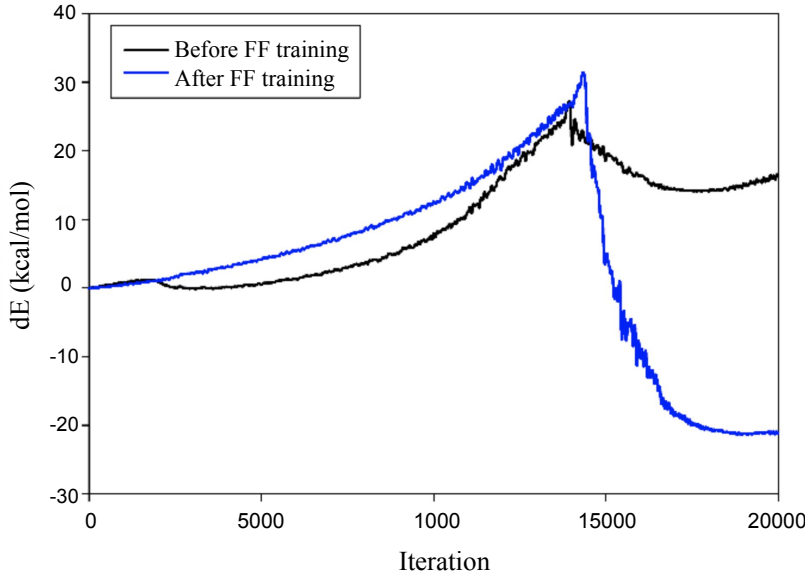


Figure 2.7 Claisen rearrangement of AVE in vacuum using original and trained ReaxFF force fields. The original force field leads to getting trapped in local minimum.

vacuum were generated. The intrinsic reaction coordinate (IRC) method was used to obtain QM data from two different chair-like and boat-like transition states. All the optimization

and IRC calculations were done with the Gaussian 16 package [26]. B3LYP functional and the 6-31G* basis set were used for the calculations. Through IRC calculations, a Claisen rearrangement transition of all chair-like and boat-like configurations and the energy changes were recorded. Comparison of these QM data against the initial ReaxFF force field results are shown in the top row of Fig. 2.8. These initial energy differences and the failure of the initial force field in capturing the Claisen rearrangement indicated the requirement of force field training against the generated QM data. The training data used for optimizing a ReaxFF parameter set can include QM data on charges, heat of formations (kcal/mol), energy minimized geometries (in angstrom or degree), lattice cell parameters (in angstrom or degree) and relative energies (kcal/mol). All the training data are added to the training set and the force field is reparametrized to minimize the error function:

$$e_i = \left(\frac{x_{i,QM} - x_{i,ReaxFF}}{\sigma_i} \right)^2 \tag{2.8}$$

where $x_{i,QM}$ and $x_{i,ReaxFF}$ are the QM and ReaxFF values of the ith entry of the training set, respectively, and σ_i are weight parameters that determine the desired accuracy for individual training data items. Force field parameters which are designated to be tuned are defined in a separate input file. The ReaxFF training feature in the standalone Fortran code then uses a line search scheme to optimize each parameter to be tuned one at a time.

The bottom row of Fig. 2.8 shows the results of our fitting against QM data for chair-like and boat-like transition states. Evaluation of the new force field after training showed satisfactory behavior in capturing the Claisen rearrangement in vacuum (Fig. 2.7). Therefore, this updated ReaxFF force field was used to perform free energy calculations of the Claisen rearrangement of AVE in the presence of explicit TIP3P water using ReaxFF/AMBER.

2.3.2 Free Energy Calculation with Umbrella Sampling

The MD driver in the ReaxFF/AMBER integration scheme is AMBER. Hence, we can use all of AMBER's advanced sampling techniques. One such feature is the umbrella sampling free energy calculation technique. Using the newly optimized parameter set, AMBER's um-

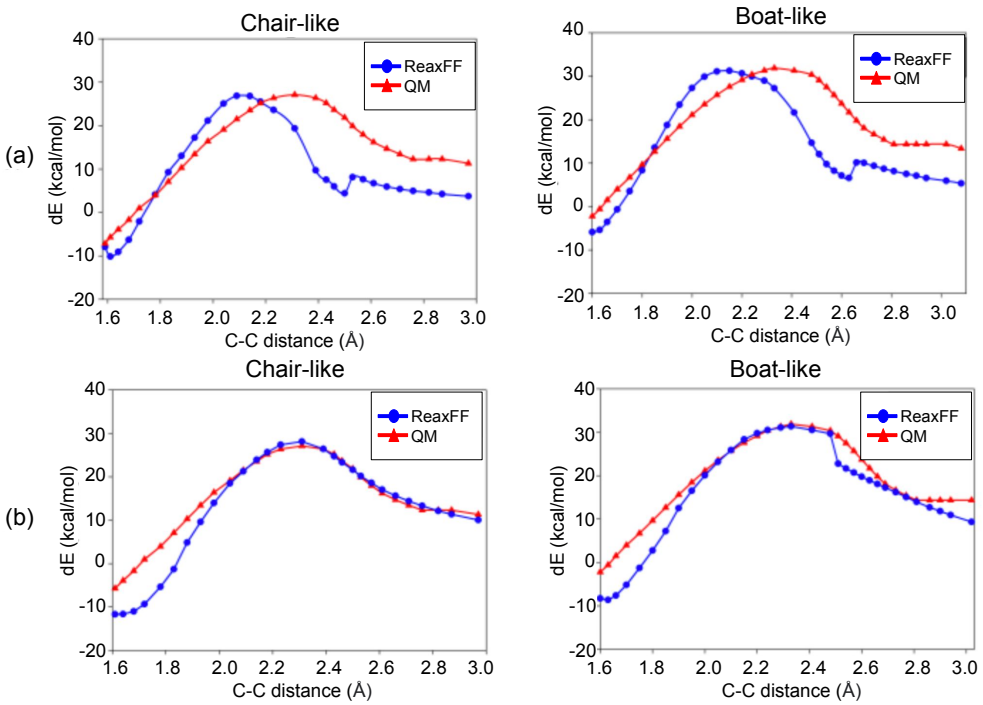


Figure 2.8 (a) QM (B3LYP/6-31G*) vs original ReaxFF force field results for the Claisen rearrangement of chair-like and boat-like structures. (b) QM (B3LYP/6-31G*) vs trained ReaxFF force field results for the Claisen rearrangement of chair-like and boat-like structures. The energies are with respect to the completely open AVE geometry optimized energy as the reference.

brella sampling feature coupled with the weighted histogram analysis method (WHAM) was used to generate the potential of mean force (PMF) of this reaction with ReaxFF/AMBER. The reaction coordinate chosen for the umbrella sampling simulations was the distance between two terminal AVE carbon atoms referred to as the $C_1 - C_5$ distance (Fig. 2.9).

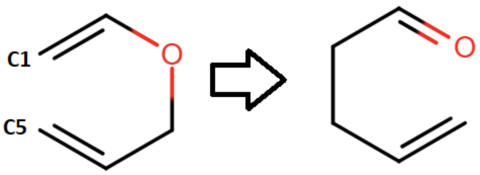


Figure 2.9 Claisen rearrangement of AVE while $C_1 - C_5$ distance is changed from 3.5 to 1.6 Å.

A series of harmonic potentials were used to constrain the reaction coordinates to the defined windows. The $C_1 - C_5$ distance was varied from 3.5 to 1.6 Å. For the regions near

the transition state (1.7-2.4 Å), 1000 kcal/mol/Å^2 was used to obtain enough sampling for each window, and for the regions far from the transition state (1.6-1.7 Å and 2.4-3.5 Å) a weaker restraint (200-700 kcal/mol/Å^2) was used. For the 1.6-3.0 Å range, 0.02 Å window intervals were defined and for the 3.0-3.5 Å range, 0.1 Å window intervals were defined. In total, 81 windows were created for sampling, while varying the $C_1 - C_5$ distance from 3.5 to 1.6 Å. All simulations were performed using the TIP3P rigid three site point charge water model. AVE was solvated in an octahedral box of TIP3P water molecules of 25 Å radius. All simulations were performed with periodic boundary conditions. The topology files for AVE were created using the general AMBER force field (GAFF) [113]. AMBER MM simulations using periodic boundary conditions were performed with a 10 Å cutoff for the real space nonbonded interactions, and the particle mesh Ewald (PME) algorithm was incorporated to account for long-range electrostatics beyond the cutoff.

After 2000 steps minimization in each window, we ran 25 ps NPT (constant number of atoms, constant pressure and constant temperature) equilibration using constant pressure Langevin dynamics with the Berendsen barostat at 300 K and 1 atm. The shake algorithm was enabled for these simulations in the MM region, and the time step of 0.25 fs was chosen for both ReaxFF (no shake) and AMBER regions. Data collection for umbrella sampling was started after the equilibration phase for 5 ps in each window. Umbrella sampling consisted of data collection from separate windows of the reaction coordinate simultaneously. By defining proper harmonic restraint constants in each window, we allowed neighboring windows to overlap and ensured there were enough windows to cover the entire reaction coordinate space. Data sampling were performed every 0.0125 ps of the production stage of umbrella sampling simulations. Finally, the PMF was calculated by combining the data from each window using WHAM.

2.3.3 Results and Discussion

To evaluate the potential of mean force (PMF) variations as a function of the reaction coordinate defined for the Claisen rearrangement, umbrella sampling calculations were em-

ployed. As mentioned, the simulations in different windows needed to be such that we could observe convergence on sampling to complete the umbrella sampling calculations. To make sure that all the windows overlap properly before performing the WHAM calculations, the histograms of all windows were generated (Fig. 2.10). This plot shows that there was proper overlap and no obvious gaps were observed. Using this data set, we used WHAM to construct the PMF of the Claisen rearrangement.

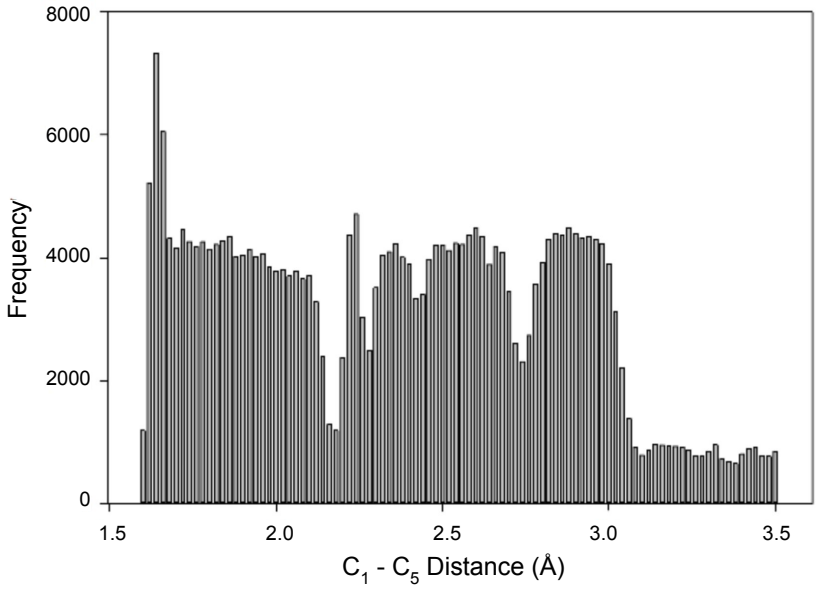


Figure 2.10 Histogram of $C_1 - C_5$ distance samplings from the umbrella sampling windows. The $C_1 - C_5$ distance was varied from 3.5 to 1.6 Å. For the regions far from the transition state (1.5-1.7 Å and 2.4-3.5 Å) a weaker restraint (200-700 kcal/mol/Å^2) was used and for the regions near the transition state (1.7-2.4 Å) a stronger restraint (1000 kcal/mol/Å^2) was used. For the 1.6-3.0 Å range, 0.02 Å interval windows were defined and for 3.0-3.5 Å windows interval of 0.1 Å were defined.

The PMF obtained from umbrella sampling calculations is shown in Fig. 2.11. The calculated PMF can be used to evaluate the Claisen rearrangement barrier height and the transition state configuration. A barrier height of 34.8 kcal/mol was obtained from these calculations. We also utilized an implementation of the self-consistent charge density functional tight-binding (SCC-DFTB) method, which is a semiempirical method based on density functional theory (DFT) [22], and also semiempirical neglect of diatomic overlap (NDDO) PM3 method [100] as part of the QM/MM support in the AMBER 18 MD program to perform

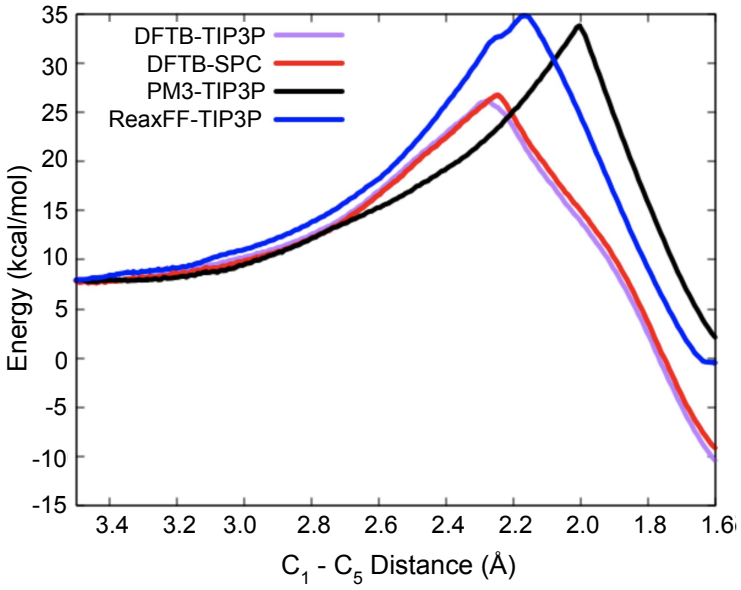


Figure 2.11 PMF of AVE Claisen rearrangement in TIP3P periodic water using combined ReaxFF/AMBER. The energies are with respect to the completely open AVE geometry optimized energy as the reference.

calculations on the Claisen rearrangement. Also, another SCC-DFTB QM/MM umbrella sampling PMF calculation using the SPC/E water model was performed to evaluate the impact of the water model on this simulation. The results of all these PMF calculations are shown in Fig. 2.11. The transition configuration of these different methods are shown in Fig. A.1. ReaxFF/AMBER integration successfully captured the Claisen rearrangement reaction.

The first kinetic study of thermal rearrangement of AVE in the gas-phase by Schuler and Murphy reported an activation energy of 30.6 kcal/mol [85]. This is consistent with other experimental reports of experimental activation energies for chair transitions [29, 115, 41]. This barrier height has been reported to be lower in water solvent than the value in the gas phase and nonpolar solvents [116]. Transition state bond lengths of 2.2-2.3 Å were reported using different simulation techniques [38].

2.4 Concluding Remarks

An AMBER/ReaxFF interface has been developed in this work that offers the capability of capturing chemical reactions as an alternative to AMBER QM/MM methods. The main

objective of this development is obtaining an approach to add a reactivity feature to the classically nonreactive molecular dynamics simulations with reduced computational costs compared to QM/MM methods. This new interface will be a useful tool for modeling big biomolecular systems with local reactive regions. The initial file-based implementation of this interface was assessed by calculation of the reaction profile of Claisen rearrangement of AVE solvated in the TIP3P water model. We conducted umbrella sampling simulations by generating a series of configurations along the distance between two end carbon atoms in AVE as the reaction coordinate, ran biasing simulations, and extracted the PMF. These results showed the capability of the ReaxFF/AMBER integration in capturing a Claisen rearrangement reaction despite the small inaccuracies in the reaction barrier height which can be enhanced by more thorough training of the ReaxFF force field.

The benchmark timing data revealed that a significant portion of the calculation time was spent on I/O and initialization/synchronization in this file-based version of the integration. As a result, we later developed an array-based implementation to eliminate these I/O and initialization/synchronization overheads. To further accelerate the calculations, we integrated PuReMD, an efficient parallel implementation of the original ReaxFF written in C. PuReMD offers substantial performance enhancements over the currently used Fortran-based ReaxFF reference implementations. It supports shared memory, distributed memory, and GPU implementations, enabling us to leverage modern massively parallel architectures. The performance of the array-based AMBER/ReaxFF (PuReMD) is presented in the follow-up work [78].

CHAPTER 3

JAX-REAXFF: A GRADIENT-BASED FRAMEWORK FOR FAST OPTIMIZATION OF REACTIVE FORCE FIELDS

This chapter was previously published as [50], and has been reformatted to meet the requirements of this dissertation. This work is reproduced with the permission of American Chemical Society.

The ReaxFF model bridges the gap between traditional classical models and quantum mechanical (QM) models by incorporating dynamic bonding and polarizability. As shown in Chapter 2, when combined with classical force fields such as AMBER, ReaxFF could enhance their capabilities to simulate interactions that require dynamic bonds and charges. However, to achieve realistic simulations using ReaxFF, model parameters must be optimized against high fidelity training data which typically come from QM calculations.

Existing parameter optimization methods for ReaxFF consist of black-box techniques using genetic algorithms or Monte-Carlo methods. Due to the stochastic behavior of these methods, the optimization process oftentimes requires millions of error evaluations for complex parameter fitting tasks, thereby significantly hampering the rapid development of high quality parameter sets. Rapid optimization of the parameters is essential for developing and refining Reax force fields because producing a force field which exhibits empirical accuracy in terms of dynamics typically requires multiple refinements to the training data as well as to the parameters under optimization. In this section, we present JAX-ReaxFF, a novel software tool that leverages modern machine learning infrastructure to enable fast optimization of ReaxFF parameters. By calculating gradients of the loss function using the JAX library, JAX-ReaxFF utilizes highly effective local optimization methods that are initiated from multiple guesses in the high dimensional optimization space to obtain high quality results. Leveraging the architectural portability of the JAX framework, JAX-ReaxFF can execute efficiently on multi-core CPUs, graphics processing units (GPUs), or even tensor processing units (TPUs). As a result of using the gradient information and modern hardware accel-

erators, we are able to decrease ReaxFF parameter optimization time from days to mere minutes. Furthermore, JAX-ReaxFF framework can also serve as a sandbox environment for domain scientists to explore customizing the ReaxFF functional form for more accurate modeling.

3.1 Background and Motivation

Before going into the details of JAX-ReaxFF, we provide some background on ReaxFF and existing software for ReaxFF parameter optimizations.

3.1.1 ReaxFF Training

ReaxFF parameters are grouped by the number of atoms involved in the interaction (e.g., single-body, two-body, three-body and four-body) in addition to the system-wide global parameters. Based on the distances and angles between atoms and corresponding model parameters, bonded, 3-body, 4-body, H-bond and non-bonded interaction lists are formed dynamically at each time step. For every interaction, corresponding parameters from the parameter set based on the element types of the atoms involved are used to calculate the E_{system}. As described in detail in Senftle et al. [88], there exist parameter sets for different kinds of simulations such as combustion, aqueous systems, metals, and biological systems. Even if there already is a parameter set for a simulation, it may require further tuning for a particular application. In some cases, the model needs to be trained from scratch which is a complex task. In general, as the number of elements in a parameter set increases, force field optimization becomes harder due to the increasing number of parameters involved.

ReaxFF training procedure requires three different inputs: i) *geometries*, a set of atom clusters crucial in describing the system of interest (e.g., bond stretching, angle and torsion scans, reaction transition states, crystal structures, etc.), ii) *training data*, properties of these atom clusters (such as energy minimized structures, relative energies for bond/angle/torsion scans, partial charges and forces) which are calculated using high-fidelity QM models, and iii) model parameters to be optimized along with a fitness function that combines different

types of training items as follows:

$$\text{Error}(m) = \sum_{i=1}^{N} \left(\frac{x_i - y_i}{\sigma_i} \right)^2 . \tag{3.1}$$

In Eq. (3.1), m is the model with a given set of force field parameters, x_i is the prediction by model m, y_i is the ground truth as calculated by QM, and σ_i^{-1} is the weight assigned to each training item.

Table 3.1 summarizes commonly used training data types (charge, energy, geometry, and force matching) and provides examples for each of them. Since ReaxFF dynamically assigns partial charges during a simulation, charge-based items could be included to help ReaxFF charges approximate QM-based atomic charges. An energy-based training data item uses a linear relationship of different geometries (expressed through their identifiers) because relative energies rather than the absolute energies drive the chemical and physical processes. As an illustrative example, the reaction

$$\mathrm{H_2} + \frac{1}{2}\mathrm{O_2} \rightarrow 2\mathrm{H_2O}$$

could be expressed as an energy-based training item with prescribed weights as

$$\frac{E_{\mathrm{H_2}}}{1} + \frac{E_{\mathrm{O_2}}}{2} - \frac{E_{\mathrm{H_2O}}}{0.5} = E_{\text{target}}.$$

For structural items, geometries must be energy minimized as accurate prediction of the lowest energy states is crucial. By including structural training items (bond distances, valence angles, and torsion angles), the ReaxFF model can be guided to accurately predict the structural properties of the target systems. For other training item types, energy minimization is optional but usually recommended. Energy minimizing the geometries before calculating the error of a given set of parameters oftentimes assists by preventing overfitting, since this process serves as data augmentation – atom positions change slightly for each energy minimization which overall makes it harder to overfit. Ultimately, whether a geometry should be energy minimized or not is up to the user.

Type	Training Item	Target	Description
Charge	ID1 1	0.5	Charge for atom 1 in geometry ID1 (in elementary charge)
Energy	ID1/1 - ID2/2 - ID3/3	50	Scaled energy difference between specified geometries (in kcal/mol)
	ID1/1	-150	
	ID3/2 - ID1/3	30	
Geometry	ID1 1 2	1.25	Distance between atoms 1 and 2 in geometry ID1 (in Å)
	ID2 1 2 3	120	Valence angle between atoms 1, 2 and 3 in geometry ID2 (in degrees)
	ID3 1 2 3 4	170	Torsion angle between atoms 1, 2, 3 and 4 in geometry ID3 (in degrees)
Force	ID1 1	0.5 0.5 0.5	Forces on atom 1 in geometry ID1 (in kcal/mol Å)
	ID2	1.0	Root mean squared gradients in geometry ID2 (in kcal/mol Å)

Table 3.1 Commonly used training item types in ReaxFF training. Identifiers (e.g., ID1, ID2 and ID3) denote geometries, i.e., atomic structures or molecules. A charge-based training item is declared by specifying the atom number in an input geometry along with the target charge value. Since energy-based training is performed using relative energies, these training items are specified through a simple linear arithmetic expression by using the identifiers of the geometries being compared. More than 2 geometries can make up an energy expression. In geometry-based training, target distance, valence angle or torsion angle values can be specified for particular atoms over the energy-minimized structure of an input geometry. For force matching, target force values between particular atoms for a given geometry are specified.

3.1.2 Related Work

Existing force field optimization methods for ReaxFF employ gradient-free black-box optimization methods such as GAs and EAs. These methods perform a global search in a high dimensional space and come with a high computational cost because they do not utilize any gradient information, but rather rely solely on error evaluations at different points in the search space for guiding improvement.

The earliest ReaxFF optimization tool is the sequential parabolic parameter interpolation method (SOPPI) [107]. SOPPI uses a one-parameter-at-a-time approach where consecutive single parameter searches are performed until a certain convergence criteria is met. The algorithm is simple, but as the number of parameters increases, the number of one-parameter optimization steps needed for convergence increases drastically. Also, the success of this method is very dependent on the initial guess and the order of the parameters to be optimized. Due to these drawbacks of SOPPI, various global methods such as GAs [21, 46, 56], SA [42, 44], EAs [105], PSO [27], and search methods based on machine learning [19, 39, 68, 89] have been investigated for ReaxFF optimization. For an explanation and evaluation of the most promising of these methods, we refer readers to the prior work by Shchygol et al.[94].

These methods have been proven to be successful for ReaxFF optimization. However, due to the absence of any gradient information, these global search methods require a high number of potential energy evaluations, as such they can be very costly.

With the emergence of optimized tools for machine learning to calculate gradients of complex functions automatically, a method called Intelligent-ReaxFF has been proposed to leverage these tools to train Reax force fields [39]. In this work, the TensorFlow library was used to calculate gradients for force field optimization. However, the method does not have the flexibility of the previously mentioned methods in terms of the training data. The force field only can be trained to match the ReaxFF energies to the absolute energies in the reference data; relative energies, charges, or forces cannot be used in the training, essentially limiting its usability. Also since it does not filter out the unnecessary 2-body, 3-body, and 4-body interactions before the optimization step, it is significantly slower than JAX-ReaxFF.

3.2 Proposed Method

In the following sub-sections, we discuss how JAX enables efficient and robust gradient calculation and how JAX is coupled with an efficient re-implementation of ReaxFF with these details in mind. We also highlight additional features with provide JAX-ReaxFF with performance portability to several computing architectures, including geometry clustering for batch processing on GPUs. Finally, we discuss how these details influence the local optimizer design.

3.2.1 Overview

The JAX library performs auto-differentiation on mathematical functions expressed using Python code. As such, implementation of the ReaxFF energy expressions (see Eq. (1.1)) in Python forms the core of JAX-ReaxFF. Once the individual energy expressions and the training error function are provided, JAX can easily calculate the gradient of the training error function with respect to the ReaxFF parameters under optimization. As mentioned earlier, atomic forces can also be part of the training data set – these can be calculated using the gradients of ReaxFF energy expressions with respect to atom positions, too.

Molecular systems used for force field training tend to have a small number of atoms compared to regular MD runs. Using a software designed for running simulations with thousands of atoms (such as PuReMD or LAMMPS/ReaxFF) to run several small scale simulations introduces overheads. Optimizations in these software (such as optimized sparse solvers for atomic charges, fast neighbor list generation algorithms, and distributed computation) would actually increase the overall run-time for small systems and result in unnecessarily complex code. Even though vanilla Python code tends to be slower than optimized Fortran or C code, when the auto-diff functionality, the use of small geometries, and the just-in-compiled XLA support (discussed in Section 3.2.3 are considered, the advantages of JAX-ReaxFF outweighs the performance loss from not using Fortran or C.

While gradient-based optimization functionality is straight-forward to achieve using JAX as described above, there are a number of important considerations needed to realize an efficient (from a run-time point-of-view) and scalable (from a memory utilization perspective) parameter optimization framework. Fig. 3.1 gives an overview of the task-flow in JAX-ReaxFF. After the neighbor list and interactions lists are calculated for the input geometries (Section 3.2.2, we cluster the inputs based on the size of their interaction lists and align them properly in memory to ensure efficient single instruction multiple data (SIMD) parallelization (Section 3.2.3). After these preparation steps, the main optimization loop is executed until convergence or the maximum number of optimization steps are reached (which typically takes only tens of iterations). During the parameter optimization loop, some molecules might require energy minimization. Hence, the main optimization loop contains a "gradient-based optimization" step followed by a "geometry optimization" step. We discuss each step in more detail in the ensuing subsections.

3.2.2 ReaxFF Model Implementation

In ReaxFF implementations for MD simulations, neighbor and interaction lists are created based on the atom positions and the fixed force field parameters. Due to the dynamic nature of interactions in ReaxFF, accurate and fast calculation of energy terms (especially

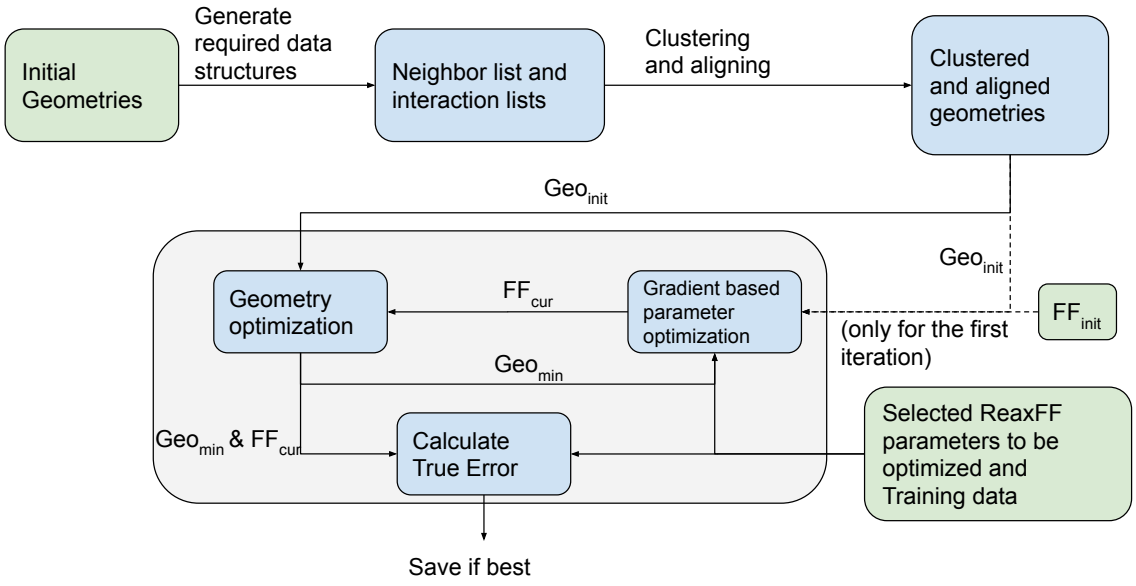


Figure 3.1 JAX-ReaxFF execution flow graph. Inputs are shown in green (initial geometries, training data, and the list of parameters to be optimized), while the core steps of JAX-ReaxFF are shown in blue. The gray box shows the main optimization loop (Algorithm 3.1).

the higher order ones such as valence angle and torsion) is critical. Differently from regular ReaxFF MD simulations, the force field is also dynamic during parameter optimization, hence adding to the challenges of developing an efficient implementation.

Pair-wise bonded interactions: We illustrate the challenges using bond order calculations as an example. As shown in Fig. 1.2, all bonded interactions depend on the corrected bond order term. Initially, if the distance between two atoms is less than a given cutoff, typically 5 Å, the uncorrected bond orders (BO) are calculated according to Eq. (3.2), where r_{ij} is the actual distance between the atom pair i-j, and r_o^{σ}, r_o^{π}, and $r_o^{\pi\pi}$ are the idea bond lengths for σ-σ, σ-π and π-π bonds, respectively. The parameters p_{bo_1}, p_{bo_3}, and p_{bo_5} are typically small negative values, while p_{bo_2}, p_{bo_4}, and p_{bo_6} are relatively large positive values. Together, they ensure that the bond order function attains the value 1 at or below the ideal

bond distance, and it smoothly decreases to 0 when r_{ij} is greater than the ideal bond length.

$$\begin{aligned}
\mathrm{BO}'_{ij} &= \mathrm{BO}^{\sigma}_{ij} + \mathrm{BO}^{\pi}_{ij} + \mathrm{BO}^{\pi\pi}_{ij} \\
&= \exp\left[p_{\mathrm{bo}_1}\left(\frac{r_{ij}}{r^{\sigma}_{o}}\right)^{p_{\mathrm{bo}_2}}\right] \\
&\quad + \exp\left[p_{\mathrm{bo}_3}\left(\frac{r_{ij}}{r^{\pi}_{o}}\right)^{p_{\mathrm{bo}_4}}\right] \\
&\quad + \exp\left[p_{\mathrm{bo}_5}\left(\frac{r_{ij}}{r^{\pi\pi}_{o}}\right)^{p_{\mathrm{bo}_6}}\right]
\end{aligned} \tag{3.2}$$

Normally, if the uncorrected bond order is greater than a certain threshold, it is added to the initial bond list and subsequently bond order corrections are applied based on the neighborhood of the atoms forming the bond. In the context of parameter optimization though, whether the pair *i*-*j* will form a bond above the given threshold also depends on the values of those parameters. Furthermore, if a given molecular structure in the training data set requires geometry optimization (as is needed for most structural properties), atom positions change as well. To speedup the computation, we utilize just-in-time compilation via XLA to create a static computational graph and map it to the desired computational device. Since the shapes of the input arrays are part of the graph by design, when their sizes change, JAX automatically triggers recompilation which is quite expensive considering the complexity of the ReaxFF model. Therefore we create the interaction lists once before the optimization starts and use masks to ignore the unwanted elements throughout the parameter and/or geometry optimization steps. For this purpose, for every unique element pair, the maximum possible distance (r_{max}) where each atom pair can have a bond order above a given threshold is determined using the equations below:

$$f_{ij}(r) = \max_{\substack{p_{\mathrm{bo}_{1\text{-}6}},\\ r^{\sigma}_{o},r^{\sigma}_{o},r^{\pi\pi}_{o}}} \quad BO^{'}_{ij} \tag{3.3}$$

$$\begin{aligned}
&r_{max} = \max_{r} \quad r \\
&\text{subject to} \qquad f_{ij}(r) \geq \text{threshold}.
\end{aligned} \tag{3.4}$$

If some BO related parameters are included in the optimization, the parameter values which maximize the BO term (Eq. (3.3)) are selected from the specified parameter ranges. Then through a distance scan, the maximum possible distance is determined as the cutoff for inclusion of bond orders between that pair of elements (Eq. (3.4)). For geometries that require minimization, the maximum calculated distance is extended by a buffer distance to be able to accommodate potential atom position changes. This enables us to eliminate the expensive recompilation step as the atoms move during geometry optimization and/or force field parameters change during the parameter optimization process.

Higher Order Bonded Interactions: Similar logic is applied for other types of interactions. In a given molecule with N atoms, when there is no trimming, there will be $O(N^3)$ 3-body and $O(N^4)$ 4-body interactions. Trimming these interaction lists is required to decrease the computational and memory costs. 3-body and 4-body interaction lists are built using the corrected BO term. Another threshold is applied to the bonds forming the 3-body and 4-body interaction lists. Since the higher order bonded interactions are built using the corrected BO terms, the thresholds are also based on the previously described maximum possible BO terms. Further trimming of the lists is possible by scanning multiple distances and angles, but due to the increased computational complexity, trimming based solely on the BO term is employed.

Non-bonded Interactions: It is assumed that there is a non-bonded interaction between every atom in the system since the non-bonded interaction cutoff (which is typically 10 Å) is much larger than the size of molecular/crystal structures used for training. Therefore, non-bonded interactions form an N x N matrix. If the system has periodic boundary conditions with box dimensions a, b, and c and with non-bonded interaction cutoff of r, then the size of the tensor for non-bonded interactions will become

$$N^2 \cdot \left(2 \cdot \left\lceil \frac{r}{a} \right\rceil + 1\right) \cdot \left(2 \cdot \left\lceil \frac{r}{b} \right\rceil + 1\right) \cdot \left(2 \cdot \left\lceil \frac{r}{c} \right\rceil + 1\right).$$

Note that the terms after N^2 accounts for the periodic boundaries.

Evaluation of the Potential Energy: Once interaction lists are created as described above, they stay constant throughout the optimization with unwanted interactions simply being masked out. Although masking wastes some computational power, it avoids the expensive reneighboring, interaction list recreation, and recompilation steps as force field parameters evolve. It also leads to a simplified codebase because the interaction list generation part can be separated from the force field optimization process. The interaction list creation is always performed on the CPU using multiprocessing, regardless of whether a hardware accelerator is used for the optimization part or not.

To calculate the potential energy, a similar approach to the standalone ReaxFF code is followed with the exception of charge equilibration. The use of EEM for distributing partial charges requires the solution of a system of linear equations (for details see Mortier et al.[65]) which is solved using preconditioned iterative solvers for large systems [69]. However, since the number of atoms is small for the training set structures, a direct LU factorization was easier to implement and auto-differentiate.

3.2.3 Clustering and Alignment for SIMD Parallelization

JAX uses XLA, a domain specific compiler for linear algebra, to achieve hardware portability. Using XLA, JAX-ReaxFF can easily run on multi-core CPUs, GPUs, or TPUs without any code changes. JAX offers vectorization (v_{map}) and parallelization (p_{map}) support to take full advantage of the underlying architecture. While v_{map} is aimed at single instruction multiple data (SIMD) parallelism which merges multiple small computations into batches to achieve high device utilization, p_{map} targets multiple instruction multiple data parallelism.

Our target architecture has been GPUs as they provide significant performance advantages over multi-core CPUs and have become the mainstream hardware accelerators. However, attaining high performance on GPUs requires some important considerations. Since parameter optimization requires efficient execution of several small atomic structures as opposed to running one big MD simulation in parallel, JAX-ReaxFF leverages the v_{map} support to accelerate the energy and gradient calculations. The keys for efficient vectorization

in JAX-ReaxFF are the pre-calculation of interaction lists that remain static throughout optimization (as described in the previous subsection), the clustering of input geometries with similar computational demands together (explained below), and the alignment of the interaction lists of geometries in the same cluster (by padding as necessary) for efficient memory accesses. As mentioned before, unwanted/unnecessary interactions in these static lists are masked during the energy and gradient calculations so that they do not affect the results.

To cluster the input geometries for efficient vectorization, a modified version of the k-means algorithm [59] was used. The distance between geometry g_i and cluster C_j is given as

$$\text{Dist}\left(C_j, g_i\right) = \text{Cost}\left(C_j \cup \{g_i\}\right) - \text{Cost}\left(C_j\right). \tag{3.5}$$

The distance is an indicator of the change in computational load after assigning geometry g_i to cluster C_j. The cost function calculates the approximate computational cost of aligning the geometries in a given set as follows:

$$\begin{aligned}\text{Cost}\left(C\right) = s \cdot \big[&w_1 n_1 + w_2 n_2 + w_3 n_3 \\ &+ w_4 n_4 + w_5 n_5 n_1^2\big].\end{aligned} \tag{3.6}$$

In Eq. (3.6), s is the number of geometries within cluster C, and n_1, n_2, n_3, n_4, and n_5 are the max numbers of atoms, 2-body interactions, 3-body interactions, 4-body interactions, and periodic boxes within cluster C, respectively. Since n_{1-5} are the main components in the cost calculation, collectively they can be thought as the cluster center. The coefficients w_1 through w_5 are indicators of the relative computational cost for single body, two-body, three-body, four-body, and non-bonded interactions, respectively. They can be determined empirically to accurately represent the computational costs in a given training set for a particular architecture. Since the interaction lists of the geometries within each cluster need to be aligned properly, the cost is calculated by multiplying the cluster size with the weighted sum of the maximum interaction list sizes within the cluster. This enables the clustering

algorithm to group geometries that are computationally similar and consequently increase the computational as well as memory efficiencies.

After initializing the k cluster centers randomly, each geometry is assigned to these clusters based on the unique distance metric shown in Eq. (3.5). After clusters and their corresponding centers are updated, a new iteration is performed where each geometry is reassigned to the cluster that is closest to it. After every reassignment, cluster centers are updated since the newly assigned geometry might shift a cluster center drastically because we use the maximum interaction sizes within each cluster as the cluster center. If we were to delay the update till the end of the reassignment step, the final clusters would become sub-optimal due to the delay in center updates. Hence, unlike the original k-means algorithm, the order of geometries affects the result. Therefore, input geometries are shuffled after each iteration for randomization. The process continues until the cluster centers do not change anymore. Also, to ensure high performance, the clustering algorithm is executed multiple times starting from different random initial cluster centers and the one where the total wasted computation (which can be determined by the total amount of padding) is minimal is chosen as the final clustering of the geometries. Although the algorithm does not guarantee optimality, empirical results show that the presented clustering algorithm outperforms the original k-means algorithm for this task. Since the results are satisfactory, other clustering methods were not explored.

The compilation time of JAX increases drastically with the number of clusters because JAX unrolls the loop that iterates through the clusters during compilation. Also, if the wasted computation does not increase significantly, a smaller number of clusters is more preferable for GPUs since improving SIMD parallelism is easier within clusters. For these reasons, the number of clusters is selected automatically. Unless the computational gain from a higher number of cluster centers is not significant, smaller number of clusters is preferred. The pseudocode for the clustering algorithm is provided in Section B.1.

3.2.4 Gradient-Based Local Optimization

After the final clusters are formed, parameter optimization is performed using gradient-based local optimizers with multi-start as depicted in Fig. 3.1. Vectorization-based parallelism is employed for both energy minimization and parameter optimization steps shown in this figure.

For gradient-based optimization to work, JAX traces the error function from Eq. (3.1) and computes the gradients of the parameters. However, since typically many geometries require geometry optimization for energy minimization, tracing the gradients through the optimization step is more error prone due to the complex functional form of the ReaxFF. To remedy this problem, we separate the geometry optimization from the error minimization. The error function in Eq. (3.1) requires geometries to be optimized before the final predictions. The error function without the geometry optimization can be thought as a surrogate model since it is a fast way to approximate the true error where the geometry optimization is done as well. Only single step energy calculations are used for surrogate error which deviates from the true error depending on how much geometries change during the true error calculation. The approach accelerates the training significantly since only cheap single step calculations are done for the local parameter optimization and it does not require tracing the gradients through the geometry optimization step as it is not part of the surrogate error calculation.

The optimization algorithm starts from the initial geometries (Geo_{init}) and the initial force field (FF_{init}) then the force field is iteratively improved. For each iteration of the training loop shown in Algorithm 3.1, two different local optimizations are performed, one being local geometry optimization using the steepest descent method and the main one being minimization of the fitness error on the energy minimized geometries by updating the force field parameters using various local optimization method such as L-BFGS-B and SLSQP. Both of these methods are classified as quasi-Newton methods where the Hessian matrix is approximated by successive gradient calculations [52, 118]. Error minimization

Algorithm 3.1 Gradient-based local optimization.

$\mathrm{FF_{cur}} \leftarrow \mathrm{FF_{init}}$
for *iteration* $= 1, 2, \ldots$ **do**
    $\mathrm{FF_{cur}} \leftarrow$ Locally minimize the error through the selected gradient-based algorithm using $\mathrm{Geo_{min}}$ and starting from $\mathrm{FF_{cur}}$. $\mathrm{Geo_{min}}$ is fixed.
    $\mathrm{Geo_{min}} \leftarrow$ Geometry optimize the structures starting from the initial geometries $\mathrm{Geo_{init}}$ with the current model $\mathrm{FF_{cur}}$
    $E_{\mathrm{cur}} \leftarrow$ Calculate the current error using $\mathrm{Geo_{min}}$ and $\mathrm{FF_{cur}}$
    if $E_{\mathrm{cur}} < E_{\mathrm{best}}$ **then**
        $E_{\mathrm{best}} \leftarrow E_{\mathrm{cur}}$
        $\mathrm{FF_{best}} \leftarrow \mathrm{FF_{cur}}$
    end if
    if $|E_{\mathrm{cur}} - E_{\mathrm{prev}}|/E_{\mathrm{prev}} < 0.001$ **then**
        $FF_{\mathrm{cur}} \leftarrow$ Add small uniform noise to FF_{best}
    end if
    $E_{\mathrm{prev}} \leftarrow E_{\mathrm{cur}}$
end for

step uses the Geo_{min} and the optimized force field ($\mathrm{FF_{cur}}$) from the last iteration and after applying the selected gradient-based algorithm, $\mathrm{FF_{cur}}$ gets updated with the newly trained force field. This step uses the surrogate model where the error is calculated with only the single step calculations. After that the geometry optimization step starts from $\mathrm{Geo_{init}}$ and yields optimized geometries (Geo_{min}) using $\mathrm{FF_{cur}}$. The true error is calculated right after the geometry optimization, if there are any geometries that require it. After each iteration, the true error (E_{cur}) for $\mathrm{FF_{cur}}$ on the training data is calculated. If E_{cur} is lower than the lowest error so far (E_{best}), $\mathrm{FF_{cur}}$ is saved as the best force field ($\mathrm{FF_{best}}$). If the optimizer gets stuck in a local minima, noise is added to $\mathrm{FF_{best}}$ to escape the minima. The error on the surrogate gets closer to the true error as parameters converge because changes in parameters become minimal. One disadvantage of separating the energy minimization from the local optimization is that the error for the geometry items will be be ignored by the local optimization since the atom positions will not change. This introduces a discrepancy between the true error and the surrogate one. However, if the training data has multiple items related to the geometry-based items as a result of potential energy surface scans, the discrepancy could be minimized. As it is demonstrated through numerical experiments, the surrogate

approach works well in practice for a variety of training tasks which have geometry-based items.

3.3 Evaluation

We evaluate the performance of JAX-ReaxFF, as well as the quality of the resulting force fields using published data sets with different characteristics.

3.3.1 Experimental Setup

Training Tasks: We identified three training tasks[1] that form a well-rounded test bench with their varying degrees of complexity. These tasks include different system types (cobalt, a metal; silica, an amorphous material; disulfide, a molecular system), different types and numbers of items in the training data set, and different number of parameters with their respective ranges to be optimized. We also note that these training tasks have been used by others for computational studies as well as for comparison of different force field optimization methods. Hence there exist several data points in the literature that serve as a benchmark for our JAX-ReaxFF tests. While structures in the cobalt and silica data sets mostly require energy minimization, those in the disulfide case mostly require single-step energy evaluations. Section 3.3.1 summarizes the specifications of the selected training tasks. JAX-ReaxFF currently does not support training items with simulation cell optimization, as such these were ignored. This only affects the silica data set which has only 5 of them (out of a total of 296 cases).

Training Data	N_{par}	N_{strc}	N_{minim}	C	G	F	P	E
Cobalt [54]	12	146	130	0	0	0	0	144
Silica [23]	67	302	221	5	26	0	6	265
Disulfide [67]	87	231	10	0	255	4401	0	219

Table 3.2 Training data sets where N_{par} is the number of parameters to optimize, N_{strc} is the number of structures in the training data set and N_{minim} is the number of geometries to be energy minimized. Columns C, G, F, P, and E denote the number of training items used for atomic charges, geometries, atomic forces, cell parameters, and energies, respectively.

[1]The data sets which are provided in the Supplementary Information of Ref.[94] and can be downloaded from https://ndownloader.figstatic.com/files/18698201.

Baseline Results: We compare the performance and training accuracy of JAX-ReaxFF to those of methods by Shchygol et al. [94], namely the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), the Monte Carlo Force Field (MCFF) optimizer, and the GA techniques described therein. The MCFF optimizer utilizes the simulated annealing approach to slowly modify the parameters and act based on the change in the error value. The remaining two approaches are population-based and are inspired by the basic principles of biological evolution. In GAs, a population of candidate solutions for a given optimization problem is evolved towards better solutions. Typically, evolution happens through random mutations and cross-overs between selected candidate solutions. In CMA-ES, new solutions are sampled from a multivariate normal distribution. The pairwise dependencies between the parameters are captured by the covariance matrix, and as the search progresses, CMA-ES updates the covariance matrix. All three approaches use ReaxFF model as a black box and find the direction solely from the function evaluations. Shchygol et al. [94] has compared these methods on different training tasks without focusing on tuning them and repeated the experiments multiple times with different starting conditions. Since they have provided an important test bench to compare different optimizers for ReaxFF, we follow the same approach to evaluate JAX-ReaxFF.

Since the exact software and hardware from [94] are not accessible, execution times for the baseline methods are approximated on the hardware described in Section S2 of Supporting Information. We have calculated the time per true error evaluation for each training task using OGOLEM with sPuReMD backend and multiplied this by the total number of error evaluations presented in [94]. This approximation is a lower bound for CMA-ES and MCFF since they have a lower level of parallelism unlike genetic algorithms where each evaluation is independent from each other.

The initial guess is an important factor which could change the results. It is especially important for gradient-based optimizations because these methods cannot move through the space freely as they need to follow the direction of the gradients. To show the capabilities

of JAX-ReaxFF, we experimented with all three initialization methods used by Shchygol et al. [94], namely initial guesses which are random, educated, and based on the literature. For random initial guesses, values are sampled from a uniform distribution-based on the given parameter ranges. To produce educated guesses, prior information from the previous related force fields is used as it is described further in Shchygol et al. [94]. For the literature-based initial guesses, force fields developed previously using the same training data sets are used. To obtain reliable results, each optimization method is repeated ten times using different initial guesses. For the educated- and literature-based initial guesses, a small amount of uniform random noise is added to the parameters without violating the range restrictions. For each parameter p, the noise value is sampled from $\left[\frac{-1k}{10}, \frac{1k}{10}\right]$ where k is the length of the given range for parameter p. By increasing the range of the random noise, the optimizer could be forced to move further away from the starting parameters, but the described noise range is used for the results presented herein.

3.3.2 Run-time and Training Performance Evaluation

In JAX-ReaxFF, as mentioned above, two different gradient-based optimization algorithms are available, L-BFGS-B and SLSQP. For both L-BFGS-B and SLSQP, the maximum number iterations is set to 100 which is for the step 3 of Algorithm 3.1. For L-BFGS-B, the maximum number of iterations for the line search is set to 20 and the maximum number of variable matrix corrections to approximate the Hessian matrix is set to 20. For the rest of the control parameters, the default values from the SciPy library are used. The iteration count for the main optimization loop of Algorithm 3.1 is set to 20 where the local error minimization and the geometry optimization steps are iteratively repeated this many times. Therefore, for all JAX-ReaxFF experiments, the true error calculation with geometry optimization is done 20 times as the local error minimization only uses single step calculations. For experiments reported here, the percentage of the noise used to help escaping the local minima (line 11, Algorithm 3.1) is set to be 0.01%. Increasing the noise can potentially improve the results, but this typically results in higher training times. JAX-ReaxFF was

compiled to run in single precision for all experiments except for the disulfide case due to the numerical issues described later. Each training experiment is repeated 10 times and the lowest and median training errors are reported.

3.3.2.1 Cobalt Force Field Optimization

Cobalt test case has only energy-based training items targeting the bulk cobalt properties. About 90% of these items require energy minimization, yet the training error does not fluctuate as shown in Fig. B.1. This shows that the surrogate error is close to the true error for this data set. Otherwise, the error would fluctuate between iterations since the surrogate error is used for the error minimization in each iteration. For some of the random runs, SLSQP does not show any progress initially. One possible explanation is that when the initial parameters are from a non-smooth part of the optimization space that cause high gradients, the optimizer fails to escape (Fig. B.1b). Small noise that is added when a stall in progress is detected stimulates progress as expected.

In Section 3.3.2.1, we compare the convergence of JAX-ReaxFF against the black box approaches for the cobalt test case. For optimizations with JAX-ReaxFF, we observe that the SLSQP method almost always outperforms L-BFGS-B in terms of both lowest and median errors; this is also true for the silica and disulfide test cases, too. Hence, we compare JAX-ReaxFF to other techniques based on the SLSQP results. Among black box methods, the CMA-ES method gives the best force fields in terms of the lowest errors, but the GA method is the best one when it comes to median errors. Compared to CMA-ES, we observe significantly faster convergence for JAX-ReaxFF in terms of both the number of evaluations required (≈600 vs. 45,000) as well as the time taken (≈25 vs. 880 minutes), while obtaining similar or better force fields as measured by the lowest error criterion. Compared to the GA method, SLSQP is even more efficient computationally (≈25 vs. 3913 minutes), while yielding significantly better results in terms of the median error criterion except for the random initial guess case. These comparisons are CPU times only, but as mentioned above, one advantage of JAX-ReaxFF is that it can be ported to GPUs simply by choosing GPUs

Method	Initial Guess	Lowest Error	Median Error	Avg. # Single Step Eval.	# True Eval.	Avg. CPU Exec. Time (min)	Avg. GPU Exec. Time (min)
L-BFGS-B	rand	1368	2334	480	20	23.5	1.2
	edu	1352	1499	418			
	lit	1366	1446	450			
SLSQP	rand	1191	2253	513	20	24.8	1.3
	edu	1168	1188	618			
	lit	1187	1189	637			
Genetic Algorithm	rand	1346	1645	-	200k	3913	-
	edu	1349	1424				
	lit	1345	1483				
CMA-ES	rand	1150	1894	-	45k	880	-
	edu	1159	1491				
	lit	1180	2320				
MCFF	rand	1422	2104	-	45k	880	-
	edu	1532	2092				
	lit	1360	1405				

Table 3.3 Cobalt training results.

as the compilation target. On the GPU, JAX-ReaxFF's execution times decrease by a factor of 20, making it even more advantageous compared to the CPU-only black-box codes.

3.3.2.2 Silica Force Field Optimization

The silica training set includes energy-based, charge-based, and geometry-based items to describe the behavior of amorphous and crystalline silica, as well as its reactions. 73% of the items in this training set require energy minimization. As shown in Fig. S2 in SI, unlike the cobalt case, the error fluctuates more between iterations, possibly due to the presence of geometry matching items in the training set, the relatively unstable nature of energy minimization during parameter optimization and the fact that the number of parameters to be optimized are significantly higher in this case than the cobalt case. Across all initial guess schemes and error criteria, the best performing black box method for the silica test case is the GA method. Although the surrogate model based on single point evaluation for JAX-ReaxFF ignores the simulation cell optimization items, it is still able to minimize the training errors comparable to those of the GA method ($\approx$3900 vs. $\approx$3650 for the lowest error and $\approx$4500 vs. $\approx$3750 for median error). The execution times for JAX-ReaxFF (32

Method	Initial Guess	Lowest Error	Median Error	Avg. # Single Step Eval.	# True Eval.	Avg. CPU Exec. Time (min)	Avg. GPU Exec. Time (min)
L-BFGS-B	rand	3901	5214	1865	20	25.0	1.6
	edu	4143	4467	1385			
	lit	4315	5068	1929			
SLSQP	rand	3870	4498	2962	20	31.9	2.0
	edu	3977	4540	2839			
	lit	3857	4534	2938			
Genetic Algorithm	rand	3577	3738	-	200k	1632	-
	edu	3705	3817				
	lit	3593	3721				
CMA-ES	rand	3739	4753	-	45k	367	-
	edu	3747	4122				
	lit	3793	4298				
MCFF	rand	5059	6584	-	45k	367	-
	edu	5632	7127				
	lit	4885	6126				

Table 3.4 Silica training results.

mins on the CPU and 2 mins on the GPU) are significantly shorter than that of the GA method (1632 mins). The CMA-ES method which gives force fields with errors similar to JAX-ReaxFF's is still significantly slower (367 mins) than JAX-ReaxFF.

3.3.2.3 Disulfide Force Field Optimization

The disulfide training data is significantly different from the previous ones as it mainly relies on force matching for model optimization. In JAX-ReaxFF, forces are calculated by taking the derivative of the potential energy expressions with respect to atom positions

$$F_x = \frac{\partial E_p}{\partial x}$$

$$\frac{\partial (F_x - F_t)^2}{\partial p} = \frac{\partial (\frac{\partial E_p}{\partial x} - F_t)^2}{\partial p} \tag{3.7}$$

where F_x is the 3-dimensional force vector for atom x, F_t is the target force vector from the training data set and p is the model parameter to be optimized. The term $\frac{\partial (F_t - F_x)^2}{\partial p}$ gives the gradients for the force matching items in the objective function. We observed that when the optimization is performed in single precision as we did with the previous two cases, the gradients of the force-based items with respect to the parameters from Eq. (3.2) result in

Method	Initial Guess	Lowest Error	Median Error	Avg. # Single Step Eval.	# True Eval.	Avg. CPU Exec. Time (min)	Avg. GPU Exec. Time (min)
L-BFGS-B*	rand	6513	12906	2017	20	13.5	1.2
	edu	8162	9649	1942			
	lit	6943	7578	1837			
SLSQP*	rand	4744	43712	1059	20	22.2	2.0
	edu	4774	4913	3523			
	lit	4970	5836	4270			
Genetic Algorithm	rand	19285	20384	-	340k	878	-
	edu	18054	20150				
	lit	18524	21206				
CMA-ES	rand	8052	11371	-	45k	116	-
	edu	8727	11105				
	lit	9284	11120				
MCFF	rand	8507	11893	-	45k	116	-
	edu	9608	13393				
	lit	10605	13625				

Table 3.5 Disulfide training results (using double precision).

extremely high values, $\sim 10^{17}$. These high gradients prevent the local optimizers from doing any progress as seen in Fig. B.3. Bond order parameters form the core of the dynamic bond concept in ReaxFF and therefore affect all types of bonded interactions. Calculation of the numerically sensitive bond order functionals [28] and the second order gradients necessary for force-matching (Eq. (3.7)) in single precision is likely the culprit for this problem. This issue can be remedied easily by performing the disulfide training in double precision at the expense of roughly doubling the execution time.

As shown in Fig. B.4 and Section 3.3.2.3, this significantly improves the results. JAX-ReaxFF can attain training errors much lower than the black-box methods (4744 vs. 8052 for lowest errors and 4913 vs. 11105 for median errors) in much shorter time (22 vs. 116 minutes on the CPU) despite the performance hit from using double precision. On the GPU, the execution time of JAX-ReaxFF with double precision again takes only a couple minutes. This situation shows that gradient-based optimization is prone to numerical failures when higher order gradients are needed, but the use of double precision provides an easy remedy and the resulting execution time penalty does not represent any problem in terms of the total times taken.

3.3.3 Discussion of the Results

These results show that the training errors of the force fields optimized using JAX-ReaxFF are on par with or better than those from the literature [94], while the training time decreases by one to three orders of magnitude. The relatively good median training errors obtained from multiple runs also indicate that JAX-ReaxFF could produce various force fields with comparable performance. The main benefit is that the overall training times are significantly lower which enables researchers to quickly iterate over ideas, try different weighting of the training items, or use different data for their indented tasks.

The training error of optimized parameters can be seen as proxies, but the quality of the resulting force field parameter sets ultimately need to be validated through actual MD simulations and comparisons against experimental and/or QM data. The next section focuses on validating the quality of the force fields trained using JAX-ReaxFF.

3.4 Force Field Validation

MD simulations in this work are performed using the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS), a molecular dynamics program from Sandia National Laboratories [103]. A relatively short time step of 0.5 fs was used in all simulations. This is the recommended setting for ReaxFF simulations of systems that do not include light atoms like hydrogen. All NVT ensemble (constant number of atoms, volume and temperature) simulations were performed using the Nose-Hoover thermostat to control the temperature with a temperature damping parameter of 100 fs which determines how rapidly the temperature is relaxed. All NPT ensemble (constant number of atoms, pressure and temperature) simulations were performed using the Nose-Hoover thermostat to control the temperature with a temperature damping parameter of 100 fs and the Nose-Hoover barostat to control the pressure with a temperature damping parameter of 1000 fs.

3.4.1 MD Simulations of Pure Cobalt

We investigated the crystal lattice constant correlation with cohesive energy in crystals of fcc cobalt for validation. The lattice constant was changed from 3 Å to 5 Å and the

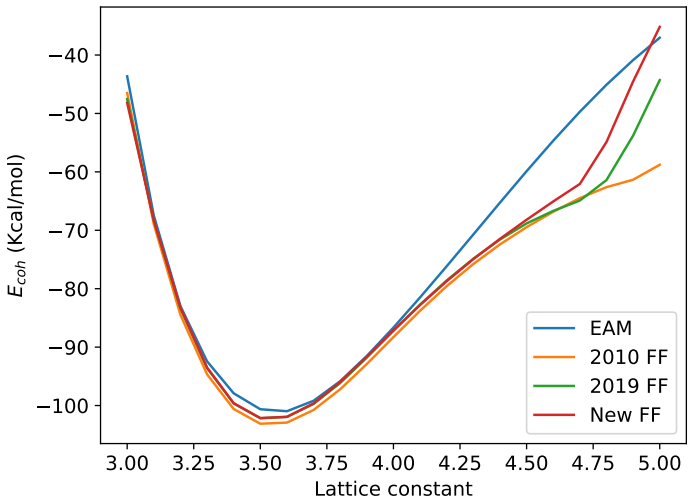


Figure 3.2 Variations in pure cobalt single fcc crystal cohesive energy by variations of the lattice constant.

associated lattice cohesive energies were recorded (Fig. 3.2). Simulation results from the force field with the lowest error fitted using JAX-ReaxFF (see Section 3.3.2.1) were compared to two previously published ReaxFF force fields for cobalt [54, 94] and embedded atom method (EAM) force field [75]. To validate the quality of the force field in capturing the dynamics behavior, the annealing loop was generated for a pure cobalt crystal structure and was compared to the available force fields. A cubic simulation box of 5 x 5 x 5 ideal fcc cobalt unit cells was generated for annealing simulations using the NPT ensemble between 1000 K and 3000 K. After the NPT equilibration of the pure cobalt crystal at 1000 K, the system was subjected to NPT ensemble annealing between 1000 K and 3000 K by 10 K/ps heating and cooling rate. A time step of 0.5 fs was used for the simulations. The changes in the system energy during this annealing loop is shown in Fig. 3.3. Three ReaxFF force fields showed similar dynamic evolution behavior for the pure cobalt structure while the EAM force field showed a different dynamic evolution (Fig. 3.3). After completion of the annealing loop, structural evaluation showed that using the ReaxFF force fields resulted a considerable recrystallization in the pure cobalt structure, while recrystallization was not observed when EAM force field was utilized (Fig. 3.4). These results demonstrate that Reax

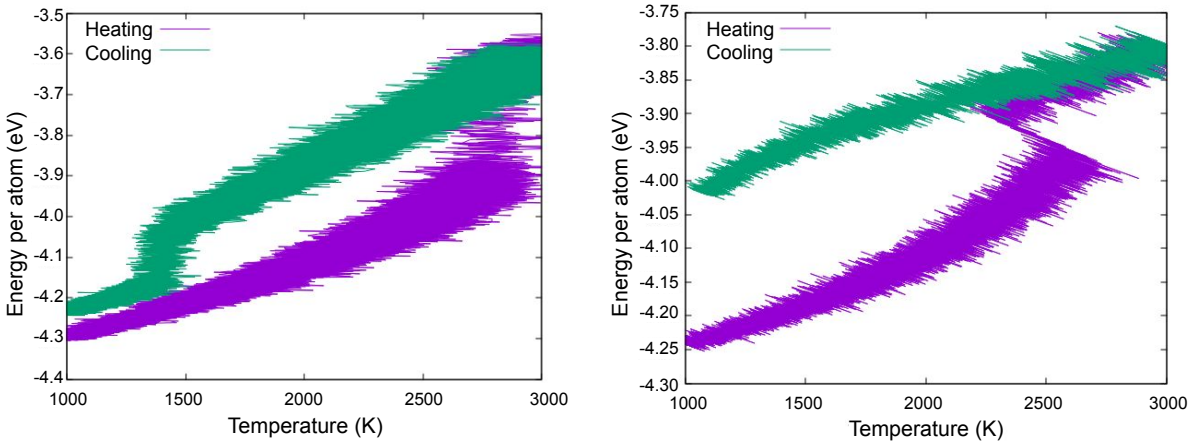


Figure 3.3 Annealing loop of a 5 x 5 x 5 fcc cobalt crystal including 500 atoms using the newly fitted ReaxFF force field (left) and the EAM force field (right) with heating and cooling rate of 10 K/ps.

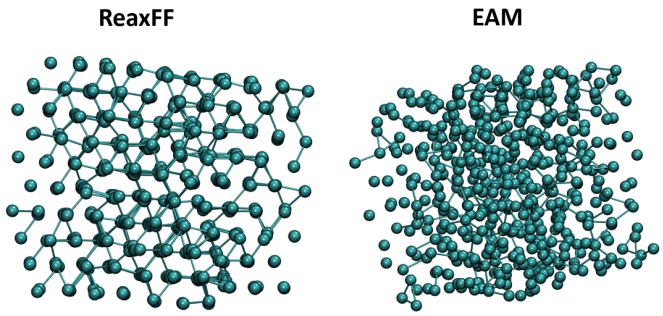


Figure 3.4 Final configurations of pure Co fcc crystals after the annealing loop in the 1000 K and 3000 K temperature range.

force fields yield more physically meaningful simulations, and the fact that the newly fitted force field can produce simulation results similar to those reported in the literature provides validation for the JAX-ReaxFF based optimization method.

3.4.2 MD Simulations of Amorphous Silica

To evaluate the quality of the silica force field optimized using JAX-ReaxFF, the amorphous silica structure introduced in Fogarty et al. [23] was reconstructed. The amorphous silica system included 2000 SiO_2 molecules with an initial density of 2.2 g/cm^3 (Fig. 3.5); it was energy-minimized first to eliminate initial bad contacts. The system was then annealed twice between 300 K and 4000 K. The first annealing loop was performed using NVT ensemble with heating and cooling rate of 37 K/ps. The second annealing loop was performed

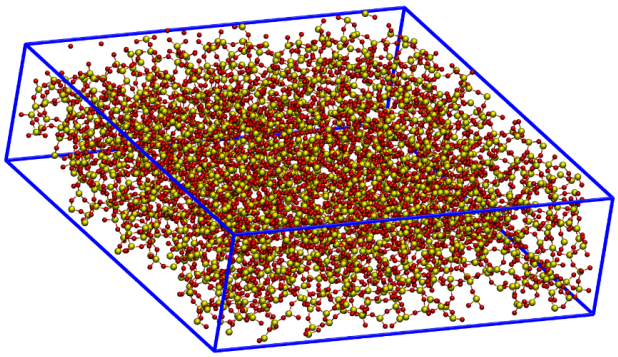


Figure 3.5 The amorphous silica structure including 2000 SiO_2 molecules and total of 6000 atoms. Silicon atoms are shown with yellow color and Oxygen atoms are shown with red color.

in NPT ensemble between 300 K and 4000 K using a Nose–Hoover thermostat and barostat at 1.01325 bar pressure. Similar to the NVT annealing, the heating and cooling rate was 37 K/ps.

In the production run, the silica system was equilibrated in NPT ensemble using $T =$ 300 K and $P = 1.01325$ bar for an additional 200 ps. These calculations were performed using the force field with the lowest training error from JAX-ReaxFF, as well as two previously published Reax force fields [23, 94]. The properties of the final configuration of these silica structures are compared in Table 3.6. While the new force field produced by JAX-ReaxFF results in a different parameter set than the two previously published force fields, it attains density and coordination numbers that are almost identical to the experimentally determined values. The radial distribution functions of the final configuration of silica structure equilibrated using our fitted force field are shown in Fig. 3.6). The locations of the peaks and the overall shapes for Si-O, O-O, and Si-Si closely match the expected values too. These results provide further evidence that JAX-ReaxFF can produce high quality force fields.

3.4.3 MD Simulations of Sulfur Bond Containing Molecules

To test the validity of the force fields containing Sulfur parameters optimized using JAX-ReaxFF, we performed potential energy scans for single molecules (dimethyl disulfide (DMDS), dimethyl thioether (DMTE), hydrogen sulfide (H_2S), and hydrogen disulfide

Property	2010 FF[23]	2019 FF[94]	New FF
Density (g/cm^3)	2.19	2.31	2.23
Si coordination	3.99	3.94	3.97
O coordination	1.99	1.97	1.99

Table 3.6 Silica properties calculated using three different force fields. The experimental value reported for silica density is 2.2 g/cm^3 [66].

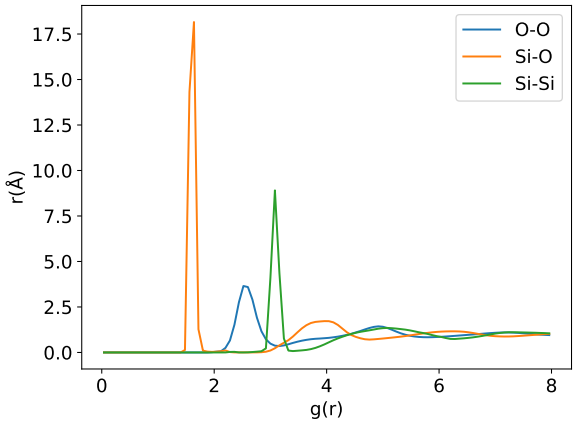


Figure 3.6 Radial distribution function of silicon-oxygen, oxygen-oxygen, and silicon-silicon for the silica structure at the end of annealing and equilibration.

(H_2S_2) with different restraints. The restraints are applied on the S-S bond of DMDS, S-C bond of DMTE, H-S-H angle of H_2S, and H-S-S-H torsion angle of H_2S_2.

In Fig. 3.7, results from newly fitted force fields are compared against two previously published ReaxFF force fields [94, 67]. The force field "New_FF2" has the lowest training error among all JAX-ReaxFF runs. While the force fields from the literature are able to nicely match the potential energy scans from CASPT2 ab initio data, New_FF2 exhibits a discontinuity for the potential energy surface of H_2S_2, pointing to a potential overfitting problem. JAX-ReaxFF actually seems to be able to find a better force field parameter set for the provided training data (in terms of the error) as it exploits the gradient information. Besides the training data, the validation data is also provided for this task [67]. Therefore, we picked another force field, called "New_FF1" from the parameter sets produced by JAX-ReaxFF. New_FF1 has the lowest cumulative error on both the training and the validation data sets. As can be seen in the H_2S_2 plot, New_FF1 fixes the overfitting issue, while being

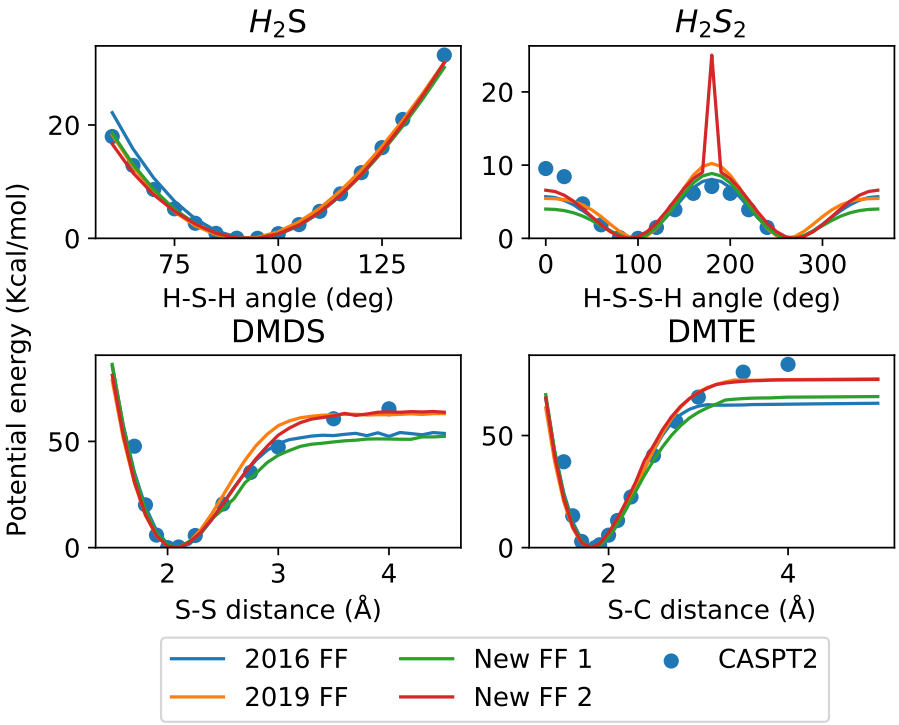


Figure 3.7 Potential energy scans of molecules containing sulfur bonds with different restraints, calculated using the newly trained and previously reported force fields. *New_FF1* has the lowest cumulative error and *New_FF2* has the lowest training error.

able to closely match the CASPT2 data overall.

3.5 Concluding Remarks

We presented a new software called JAX-ReaxFF that enables fast optimization of Reax force field parameters by leveraging recent advances in machine learning. JAX-ReaxFF uses several innovative techniques for high performance on architectures with GPUs. Clustering similar geometries together to maximize the SIMD parallelism while limiting the padding for alignment yields high parallelism, especially for single step evaluations. As it is described in Algorithm 3.1, by using single step energy evaluation-based approximations to the error function and gradient information about the search space, we are able to decrease the convergence time significantly with the help of GPU acceleration. We have empirically showed that even though the local optimizer is not fully aware of the geometry optimization, the overall algorithm converges with changes in parameter space becoming minimal as the algorithm progresses. Based on extensive experiments, we demonstrated that even if the starting

initial guess is a poor one, gradient-based local optimizers are able to improve the fitness of the force field drastically. Combined together, these innovations allow users to optimize Reax force fields in mere minutes. This is notable because existing methods require several days for obtaining essentially same quality force fields. Finally, JAX-ReaxFF provides a utility not available in other similar tools – its auto-diff functionality enables the study of the new functional forms for the interactions of the ReaxFF model without explicitly implementing the force calculations and the optimizer, since both forces and parameter gradients can automatically be calculated by JAX.

CHAPTER 4

END-TO-END DIFFERENTIABLE REACTIVE MOLECULAR DYNAMICS SIMULATIONS USING JAX

The contents of this chapter first appeared as [51]. This work is reproduced with the permission of Springer Nature.

While JAX-ReaxFF improves the speed of parameter optimization for ReaxFF, it is not suitable for running large scale MD simulations. JAX-ReaxFF precomputes the interaction lists to compute the potential energy and keeps them fixed throughout the parameter optimization. Also, since the molecules used for parameterization tend to have small number of atoms, the way the interaction lists are generated is not suitable for bigger molecules.

In this section, we introduce a new implementation of ReaxFF using JAX that is suitable for both running large scale simulations and parameter optimization. We leverage the existing JAX-MD library [84] to better support larger molecules without sacrificing performance. The new approach allows running end-to-end differentiable simulations. Moreover, it enables easy integration of ReaxFF with different potentials that already exist within JAX-MD.

4.1 Background and Motivation

4.1.1 Related Work

To enable large-scale and long duration simulations, several ReaxFF implementations with different features and architectural support have been developed over the past couple decades. PuReMD has shared and distributed-memory versions for both CPUs and GPUs (CUDA-based), all of which are maintained separately [3, 2, 53], and several of these versions have been integrated into LAMMPS and AMBER [77]. More recently, to ensure hardware portability and simplify code maintenance and performance optimizations, a Kokkos-based implementation of ReaxFF has been developed in LAMMPS [106]. Kokkos is a performance portable programming model and allows the same codebase implemented using its primitives to be compiled for different backends. The current ReaxFF/Kokkos software also supports distributed-memory parallelism. In addition to the above open-source software, SCM pro-

vides a commercial software that includes ReaxFF support [81].

As mentioned in Chapter 1, a new class of force fields called machine learning potentials (MLP) such as SNAP [104], the Behler/Parrinello potential [9], SchNet [86], OrbNet [76], and NequIP [8] has emerged. ML libraries such as Tensorflow [1], PyTorch [70], and JAX [11] have gained widespread adoption not only for ML approaches but as a general purpose programming model even when using conventional techniques. This can be attributed to the convenience of advanced tools developed around these programming models and libraries such as auto-differentiation, auto-vectorization, and just-in-time compilation. Such tools have enabled fast prototyping of new ideas as well as hardware portability without sacrificing much computational efficiency.

Intelligent-ReaxFF [39] and JAX-ReaxFF [50] implementations both leverage modern machine learning frameworks. However, they are both primarily designed for force field fitting, and as such they are designed to work with molecular systems typically containing tens of atoms, and they cannot scale beyond systems with more than a couple hundred atoms. More importantly, they both lack molecular dynamics capabilities.

4.1.2 JAX and JAX-MD Overview

Since the new ReaxFF implementation is developed in JAX-MD, important design and implementation decisions were based on how JAX and JAX-MD work. As such, we first briefly describe these frameworks.

JAX [11] is a machine learning framework for transforming numerical functions. It implements the Numpy API using its own primitives and provides high order transformation functions for any Python function written using JAX primitives. The most notable of these transformation functions are automatic differentiation (*grad*), vectorization on a single device to leverage SIMD parallelism (*vmap*), parallelization across multiple devices (*pmap*), and just-in-time compilation (*JIT*). These transformations can be composed together to enable more complex ones. JAX uses XLA, a domain specific compiler for linear algebra, under the hood to achieve hardware portability. This allows any Python code written in terms of

JAX primitives to be seamlessly compiled for CPUs, GPUs, or TPUs. Since XLA is also used extensively to accelerate Tensorflow models, XLA is supported for almost all modern processors, including GPUs by Nvidia and AMD. With JIT, XLA could apply performance optimizations targeted specifically for the selected device. The main limitation of JAX is that it expects the input data to the transformed functions to have fixed sizes. This allows XLA to adopt more aggressive performance optimizations during compilation, but when the size of the input data changes, the code needs to be recompiled.

JAX-MD [84] is an MD package built in Python using JAX. It is designed for performing differentiable physics simulations with a focus on MD. It supports periodic and non-periodic simulation environments. JAX-MD employs a scalable $3D$ grid-cell binning based algorithm to construct the neighbor list for atoms in a given system. It includes integrators for various kinds of ensembles as well as Fast Inertial Relaxation Engine (FIRE) Descent [10] and Gradient Descent based energy minimizers. Various machine learning potentials such as the Behler-Perrinello architecture [9] and graph neural networks including the Neural Equivariant Interatomic Potentials (NequIP) [8], based on the GraphNet library [7], are also readily available. When combined with the capabilities of JAX, this rich ecosystem enables researchers to easily develop and train hybrid approaches for various chemistry and physics applications.

4.1.3 Our Contribution

The aforementioned features of ML cyber-infrastructure are highly attractive from the perspective of MD software, considering the fact that existing force field implementations are mostly written in low-level languages and tuned to the target hardware for high performance. As such, we introduce a portable, performant, and easy-to-maintain ReaxFF implementation in Python built on top of JAX-MD [84]. This new implementation of ReaxFF is

- **easy-to-maintain** because it only requires expressing the functional form of the potential energy for different atomic interactions in Python. MD simulations require calculation of forces which are calculated by taking the gradient of the potential en-

ergy with respect to atom positions at each time step. This can simply be accomplished with a call to the `grad( )` function in JAX,

- **hardware portable** because for its functional transformations, JAX uses XLA (Accelerated Linear Algebra) [82], which is a domain specific compiler for vector and matrix operations. Since XLA has high performance implementations across different CPUs (x86_64 and ARM) as well as GPUs (Nvidia and AMD), porting our ReaxFF implementation does not require any additional coding,

- **performant** because we ensure that our underlying ReaxFF interaction lists are suitable for vectorization, and we leverage just-in-time compilation effectively through a carefully designed update/reallocation scheme,

- **versatile** because we designed our implementation such that the same interaction kernels can be re-used in either a single high performance run (needed for long MD simulations) or multiple small single-step runs (needed for parameter optimization) settings. This allows our implementation to be suitable for force field training as well. Also, it simplifies the study of new functional forms for various interactions in the ReaxFF model.

4.2 Design and Implementation

In this section, we describe the overall design considerations and present the final design for our ReaxFF implementation in JAX-MD. To simplify the design and ensure modularity, generation of the interaction lists have been separated from the computation of partial energy terms. For overall efficiency and scalability, special consideration has been given to memory management.

4.2.1 Memory Management

To avoid frequent re-compilations, sizes of input to JAX's transforming functions must be known and fixed. As such, we separate the logic for handling the interaction list generation

Algorithm 4.1 General structure of computations in an MD simulation.

```
interLists ← Create the interaction lists using the allocate function
for timestep = 1, 2, ... do
    Calculate forces
    Update positions using the calculated forces
    overflow ← Update the interaction lists
    if overflow then
        interLists ← Reallocate based on the most recent utilizations
    end if
end for
```

into `allocate` and `update` parts. The allocate function estimates the sizes of all interaction lists (see Fig. 4.1) and allocates the needed memory with some buffer space (default 20%). Due to its dynamic nature, JAX transformations such as *vmap* and *jit* cannot be applied to the allocate function. The update function works with the already-allocated interaction lists, and fills them based on atom positions while preserving their sizes. Since the update function works on arrays with static sizes, JAX transformations such as *vmap* and *jit* can be and are applied to this function. For effective use of *vmap*, the update function also applies padding when necessary. Finally, while filling in the interaction lists, it also checks whether the utilization of the space allocated for each list falls below a threshold mark (default 50%) where the utilization is the ratio of the true size to the total size. If it does, a call to the allocate function is triggered to shrink the interaction lists as shown in Algorithm 4.1, which in turn causes JAX to recompile the rest of the code since array dimensions change.

Another important aspect of our memory management scheme is the *filtering of interaction lists*. In ReaxFF, while bonds are calculated dynamically, not all bonds are strong enough to be chemically meaningful, and therefore they are ignored (a typical bond strength threshold is 0.01). This has ramifications for higher-order interactions such as 3-body, 4-body, and H-bond interactions as well because they are built on top of the dynamically generated bond lists. As we discuss in more detail below, the acceptance criteria for each interaction is different. For 3-body and 4-body interactions, acceptance criteria depend on the strength of bonds among the involved atoms as well as force field parameters specific

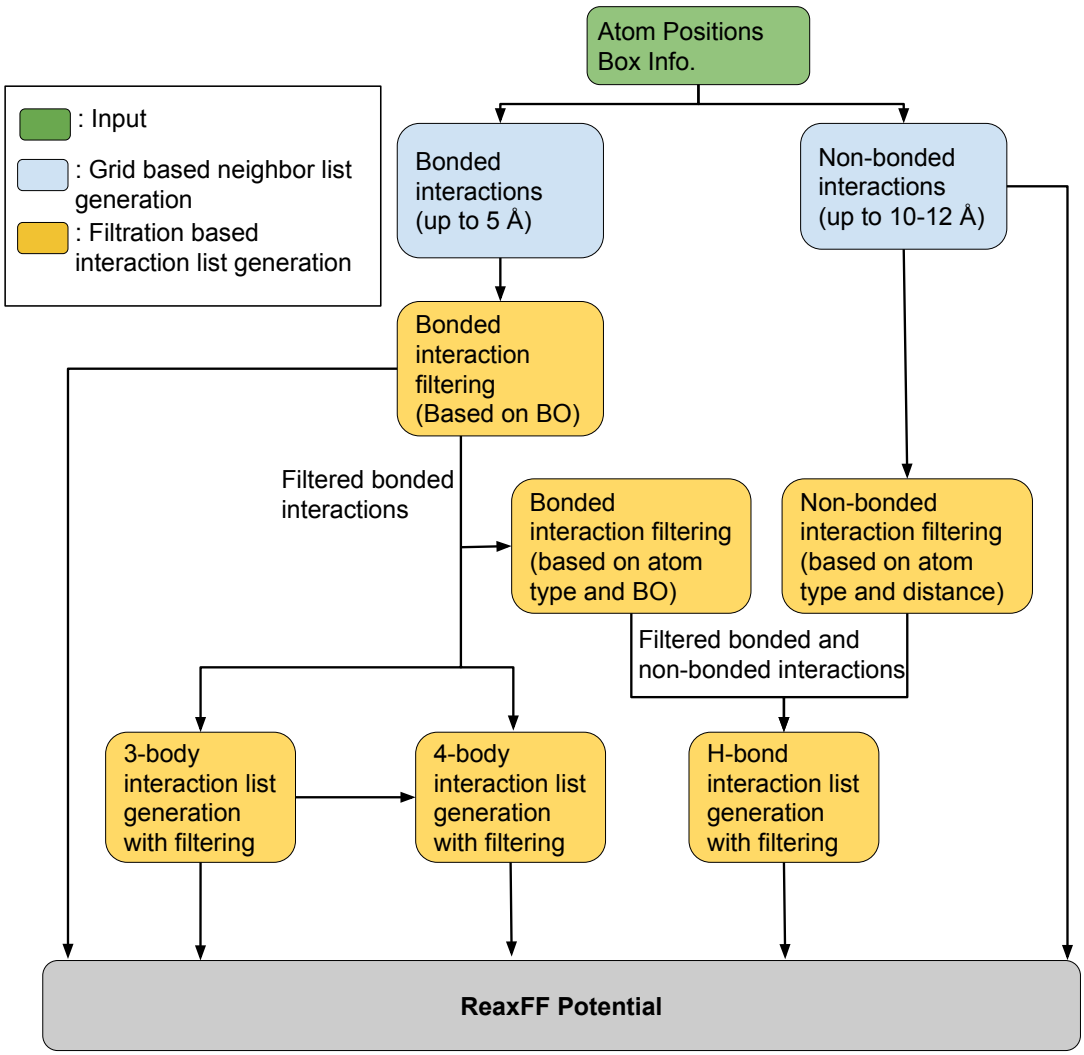


Figure 4.1 Flow graph describing the generation of the interaction lists.

to that group of atoms; for H-bonds, it is a combination of acceptor-donor atom types and bond strengths. However, the steps for filtering all interaction lists are similar and can be implemented as a generic routine with a candidate interaction list and an interaction-specific acceptance criterion. The interactions that require filtering and their relevant input data are shown as yellow nodes in Fig. 4.1. First, the candidate interaction list is populated. Then, candidates get masked based on the predefined acceptance criterion. Finally, the candidate list is pruned and passed onto its corresponding potential energy computation function. While actually pruning the candidate list might be seen as an overhead, we note that the number of unaccepted 3-body and 4-body interactions are so high that simply ignoring them

during the potential energy computations introduce a significant computational overhead. Also, the memory required to keep the unfiltered 3-body and 4-body interaction lists would limit the scalability of our implementation for GPUs due to their limited memory resources.

The filtering logic discussed above is JAX-friendly because the shapes of the intermediate (candidate) and final (pruned) data structures are fixed. As such, *vmap* and *jit* transformations can be applied to the filtering procedure, too. As with un-pruned lists, filtered interaction list generation also keeps track of utilization of the relevant lists and sets the overflow flag, when necessary.

4.2.2 Generation of Interaction Lists

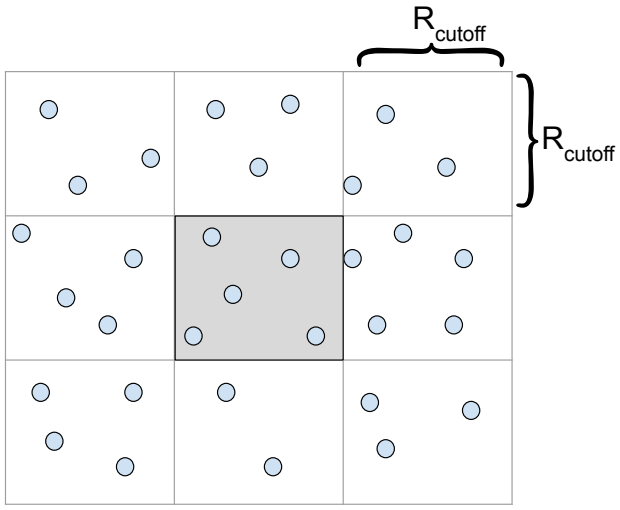


Figure 4.2 Illustration of grid-cell neighbor search used to generate neighbor lists.

Pair-wise Bonded Interactions: In ReaxFF, bond order (BO) between atom pairs are at the heart of all bonded potential energy computations. The BOs are computed in two steps. First, uncorrected BOs are computed according to Eq. (4.1), where r_{ij} is the distance between the atom pair i-j, and r_o^{σ}, r_o^{π}, and $r_o^{\pi\pi}$ are the ideal bond lengths for σ-σ, σ-π and

π-π bonds, respectively.

$$\begin{aligned} \mathrm{BO}'_{ij} &= \mathrm{BO}^{\sigma}_{ij} + \mathrm{BO}^{\pi}_{ij} + \mathrm{BO}^{\pi\pi}_{ij} \\ &= \exp\left[p_{\mathrm{bo}_1} \cdot \left(\frac{r_{ij}}{r^{\sigma}_o}\right)^{p_{\mathrm{bo}_2}}\right] + \exp\left[p_{\mathrm{bo}_3} \cdot \left(\frac{r_{ij}}{r^{\pi}_o}\right)^{p_{\mathrm{bo}_4}}\right] \\ &\quad + \exp\left[p_{\mathrm{bo}_5} \cdot \left(\frac{r_{ij}}{r^{\pi\pi}_o}\right)^{p_{\mathrm{bo}_6}}\right] \end{aligned} \tag{4.1}$$

After uncorrected bond orders are computed, the strength of BO'_{ij} is corrected based on the local neighborhood of atoms i and j. The corrected BO (BO_{ij}) represents the coordination number (i.e., number of bonds) between two atoms. Corrected bonds below a certain threshold get discarded as they do not correspond to chemical bonds. Hence, they do not contribute to the total energy.

To calculate uncorrected BO, for each atom in a given system, their neighbors are found using a grid-cell binning based neighbor search algorithm (Fig. 4.2). This allows us to generate the bonded neighbor lists in $O(Nk)$ where N is the number of atoms and k is the average number neighbors per atom. The side length of the grid cell is set to 5.5 Å, as a buffer space of 0.5 Å is added to the 5 Å actual bonded interaction cutoff to avoid frequent updates to the neighbors list. Since the cell size is almost the same as the bonded interaction cutoff, neighbor search only requires checking the nearby 3^3 grid cells. Neighbor information is stored in a $2D$ format where the neighbors of atom i are located on ith row with padding and alignment, as necessary. This format which is very similar to the ELLPACK format [110] is highly amenable for vectorization and memory coalescing on modern GPUs. It also simplifies bond order corrections because the neighbor indices for a given atom are stored consecutively. As will be discussed later, it also helps creating 3-body (for valency) and 4-body (for torsion) interactions since they use BOs as the main input. After creating the $2D$ neighbor array, BO terms are calculated and pairs with small BOs are filtered out as described above.

Higher Order Bonded Interactions: After pruning the bonded interactions, 3-body and 4-body interaction lists are generated (Fig. 4.3). For each atom, every two neighbor

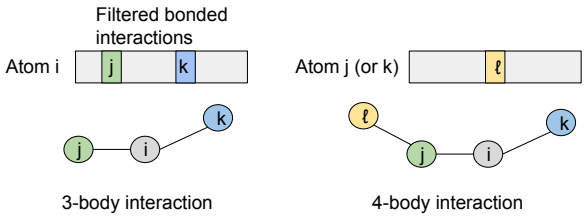


Figure 4.3 Atoms and their interactions involved in formation of the 3-body and 4-body interactions.

pairs are selected to form the candidate list for 3-body interactions. In a system with N atoms and k neighbors per atom, there will be $O\left(Nk^2\right)$ candidates. Then the candidates are masked and filtered based the involved BO terms to form the final array with shape $M \times 3$ where M is the total number of interactions and columns are atom indices. After that, the finalized 3-body interaction list is used to generate the candidates for the 4-body interactions. For each 3 body interaction i-j-k, neighbors of both j and k are explored to form the 4-body candidate list and then the candidates get filtered based on the 4-body specific mask.

When the molecule involves hydrogen bonds, the hydrogen interaction list is built using the filtered bonded and non-bonded interactions. A hydrogen bond can only be present if there are hydrogen donors and acceptors. While the acceptor and the hydrogen are covalently bonded (short range), the acceptor bonds to the hydrogen through a dipole-dipole interaction, therefore it is long ranged (up to 7.5 Å). Hence, to find all possible hydrogen bonds involving a given hydrogen atom, both its bonded neighbors and non-bonded neighbors are scanned. Using the appropriate masking criterion, the final interaction list is formed to be used for potential energy calculations.

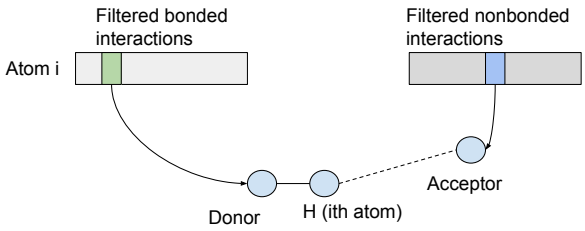


Figure 4.4 Atoms and their interactions involved in formation of hydrogen bonds.

Non-Bonded Interactions: In ReaxFF, non-bonded interactions are effective up to 10–12 Å , and they are smoothly tapered down to 0 beyond the cutoff. Similar to the pair-wise bonded interactions, the long range neighbor lists are also built using the grid-cell binning approach, this time using a buffer distance of 1 Å to avoid frequent neighbor updates. The neighbors are again stored in a $2D$ array similar to the ELLPACK format. This simplifies accessing the long range neighbors of a given atom while building the Hydrogen bond interactions list (as shown in Fig. 4.4). Also, the sparse matrix-vector multiplication kernel (SpMV) required for the dynamic charge calculation becomes simpler and more suitable for GPUs [109].

The non-bonded interaction list is used to compute van der Waals and Coulomb energy terms. While E_{vdWaals} computation is relatively simple as it only involves the summation of the pair-wise interaction energies, E_{Coulomb} requires charges to be dynamically computed based on a suitable charge model such as the charge equilibration (QEq) [80], electronegativity equalization (EE) [65], or atom-condensed Kohn-Sham density functional theory approximated to second order (ACKS2) method [111]. Our current JAX-based implementation relies on the EE method.

The EE method involves assigning partial charges to individual atoms while satisfying constraints for both the net system charge and the equalized atom electronegativities. The details of the charge equilibration are provided in Section 2.2.2. Since the size of the linear system is $(n + 1) \times (n + 1)$, it is prohibitively expensive to solve it with direct methods when n is becomes large (beyond a few hundred). Hence, we employ an iterative sparse linear solver as the matrix is sparse for large systems due the 10-12 Å cutoff for the non-bonded interactions. The iterative solvers available in JAX only expect a linear operator as a function pointer that can perform the matrix-vector multiplication. This allows us to define the SpMV operation directly using the non-bonded neighbor lists provided in an ELLPACK-like format described earlier without applying any transformations. Another optimization to accelerate the charge equilibration is to use initial guesses to warm start the iterative solver.

Algorithm 4.2 Gradient-based parameter optimization.

$\theta \leftarrow$ Initialize the model parameters
training set $\leftarrow$ Align the training set by padding with dummy atoms
lossFunction $\leftarrow$ Create a loss function by utilizing vmap(energyFunction)
calculateGradients $\leftarrow$ jit(grad(lossFunction))
while stopping criterion not met **do**
    $X_i, Y_i \leftarrow$ Sample a minibatch of data from the training set
    Create the interaction lists for X_i
    g $\leftarrow$ calculateGradients(θ, interLists, Y_i)
    $\theta \leftarrow$ Update the model parameters using g
end while

Since the charges fluctuate smoothly as the simulation progresses, we use the cubic spline extrapolation to produce the initial guesses based on past history [2].

4.2.3 Force Field Training

Predictive capabilities of empirical force fields are arguably more important than their performance. For this, it is crucial for force field parameters to be optimized using high-fidelity quantum mechanical training data. In contrast to MD simulations involving a single system iterated over long durations, this optimization process typically involves executing several (on the order of hundreds to thousands, depending on the model and target systems) small molecular systems for a single step using different parameter sets in a high-throughput fashion. While evolutionary algorithms have traditionally been used for Reax force field optimizations, as JAX-ReaxFF [50] and Intelligent-ReaxFF [39] have recently demonstrated, using gradient-based optimization techniques can accelerate the training process by two to three orders of magnitude. However, the gradient information needed for force field optimization is much more complex than that of MD simulation – one needs to calculate the derivative of the fitness function which is typically formulated as a weighted sum of the difference between predicted and reference quantities over all systems in the training dataset with respect to parameters to be optimized (which is usually on the order of tens of parameters for ReaxFF). While this would be a formidable task using analytical or numerical techniques, the auto-differentiation capabilities of JAX enable us to easily repurpose the

above described ReaxFF MD implementation for parameter optimization. By composing different transformations, a simple loss function defined for a single sample can extended to work for a batch of training data as shown in Algorithm 4.2. To fully take advantage of SIMD parallelism, especially on GPUs, we ensure that different molecules in the training dataset are properly divided into small batches. To reduce the number of dummy atoms and the amount of padding within each batch, the training set could be clustered based on how much computation they require. Given the allocate/update mechanism described in Section 4.2.1, the different sizes of interaction lists for different molecular systems in a batch data does not cause additional challenges.

4.3 Experimental Results

4.3.1 Software and Hardware Setup

To verify the accuracy of the presented JAX-based ReaxFF implementation, simulations were performed using molecular systems shown in Table 4.1. The Kokkos-based LAMMPS implementation of ReaxFF was chosen for validation and benchmarking comparisons due to its maturity and maintenance. For this purpose, we used the most recent stable release of LAMMPS (git tag stable_23Jun2022_update3), and experimented on both Nvidia and AMD GPUs. LAMMPS was built using GCC v10.3.0, OpenMPI v4.1.1, and CUDA v11.4.2 for the Nvidia GPUs, and with ROCm v5.3.0, aomp v16.0, and OpenMPI v4.1.4 for the AMD GPUs (using device-specific compiler optimization flags for both). For the JAX experiments, Python v3.8, JAX v0.4.1, and JAX-MD v0.2.24 were paired with CUDA v11.4.2 for the Nvidia GPUs, and ROCm v5.3.0 for the AMD GPUs. Hardware details are presented in Table 4.2. The compute nodes at the Michigan State University High-Performance Computing Center (MSU-HPCC) and the AMD Cloud Platform are used for the experiments.

Name	Chem. Rep.	N	Sim. Box (Å)	Force Field
Water	H_2O	2400	$29.0 \times 28.9 \times 29.3$	[23]
Silica	SiO_2	6000	$36.9 \times 50.7 \times 52.5$	[23]

Table 4.1 Molecular systems used in the performance evaluation section, with the third column (N) indicating the number of atoms, the fourth one denoting the dimensions of the rectangular simulation box, and the last column showing the force field used to simulate the system.

GPU	CPU	Cluster
A100	Intel Xeon 8358 (64 cores)	MSU-HPCC
V100	Intel Xeon Platinum 8260 (48 cores)	MSU-HPCC
MI210	AMD EPYC 7742 (64 cores)	AMD Cloud Platform
MI100	AMD EPYC 7742 (64 cores)	AMD Cloud Platform

Table 4.2 Hardware details of the platforms used for performance experiments.

4.3.2 Validation of MD Capabilities

Fig. 4.5 shows that the JAX-based ReaxFF energies almost perfectly match those from LAMMPS in actual MD simulations. The deviation only becomes visible after 2000 MD steps which is inevitable due to machine precision limitations. The relative energy difference is around 10^{-7} for both the water and silica systems.

4.3.2.1 Performance and Scalability

We compare the performance of JAX-based ReaxFF to Kokkos/ReaxFF package in LAMMPS on both Nvidia and AMD GPUs. While Kokkos/ReaxFF supports MPI parallelism, we use a single GPU for all tests. Kokkos/ReaxFF incurs minimal communication overheads when there is a single MPI process. The performance comparison on AMD GPUs is possible through Kokkos' ROCm backend support, as well as the availability of JAX/XLA on AMD GPUs.

To create systems with varying size, the molecular systems shown in Table 4.1 have been periodically replicated along the x, y, and z dimensions. The number of atoms vary from 2400 to 19200 for the water systems and from 6000 to 24000 for the silica systems.

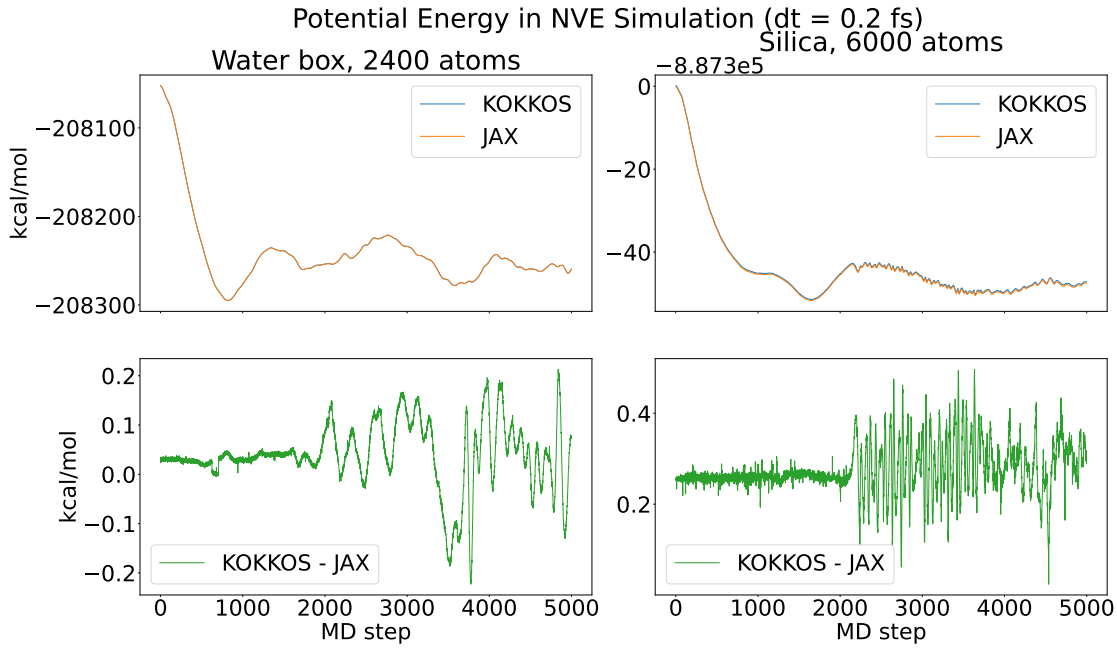


Figure 4.5 Comparison of absolute (top plots) and relative difference (bottom plots) in potential energies for NVE simulations with a time step of 0.2 fs and a CG solver with 1e-6 tolerance for the charge calculation.

For each experiment, NVE simulations with a time step of 0.2 fs were run for 5000 steps, and the average time per step in ms was reported. For both the Kokkos and JAX-based implementations, the buffer distance for the non-bonded interactions was set to 1 Å. While reneighboring is done every 25 MD steps for Kokkos, the JAX implementation keeps track of how much atoms move since the last neighborhood update and only reneighbors when atoms move more than the buffer distance. As suggested by the Kokkos documentation, the half-neighbor list option is used.

While written in Python using JAX primitives, the proposed implementation is faster when the system size is small on all GPUs. As the number of atoms increases, while the time increases linearly for the JAX implementation, the Kokkos one increases sublinearly. The sublinear scaling for Kokkos indicates that it cannot fully utilize the resources when the problem size is small unlike JAX. As the problem size increases, Kokkos starts to utilize the GPU better and yield better performance. The Kokkos implementation achieves up to 3.2x speedup for the largest water systems on AMD GPUs (MI100 and MI210). On Nvidia GPUs

(V100 and A100), it is around 2.3x faster for the same water system with 19200 atoms. For the silica systems where there are no hydrogen bonds, Kokkos is around 2x faster on the AMD GPUs and 1.5x on the Nvidia GPUs. On the other hand, when the problem size is small, JAX achieves up to 1.8x speedup on an A100 GPU.

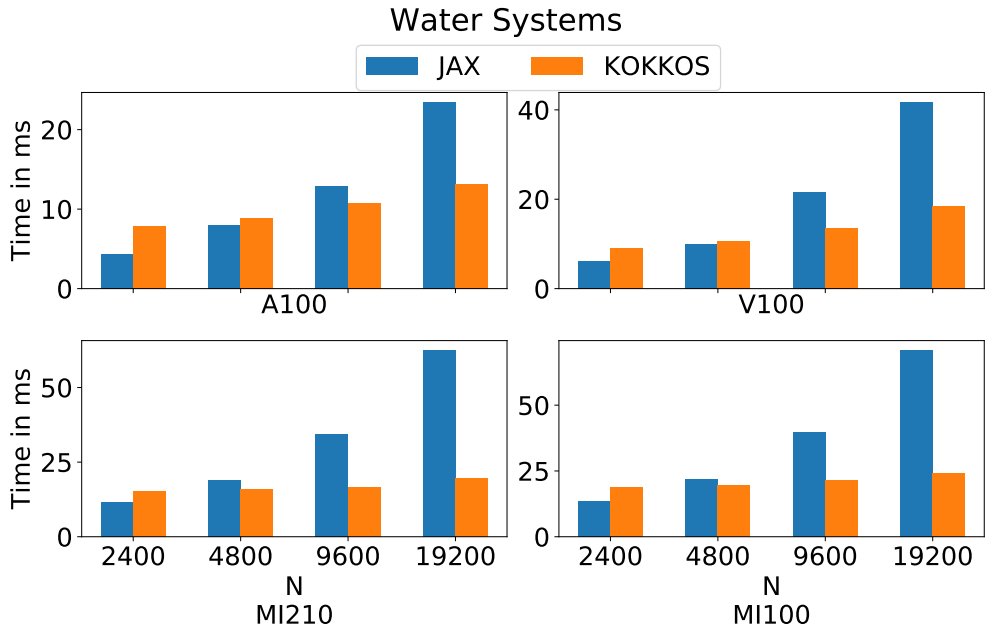


Figure 4.6 Average time per MD step (in ms) for the water systems with varying sizes.

4.3.3 Training

To demonstrate the training performance of the described implementation, we trained the ReaxFF parameters on the public QM9 dataset of about 134k relaxed organic molecules made up of H, C, N, O, and F atoms, with each molecule containing up to nine non-hydrogen atoms [79]. All systems are calculated at the B3LYP/6-31G(2df,p) level of theory. To simplify the dataset, we removed the molecules that contain F atoms which resulted in around 130k molecules. During optimization, 80% of the data is used for training and the remaining 20% for testing. The training is done using the AdamW optimizer [60] from the Optax library [5] with a batch size of 512 and the learning rate is set to 0.001.

The ReaxFF model is typically fit to the training data containing relative energy differences between molecules with the same type of atoms (different conformations and configu-

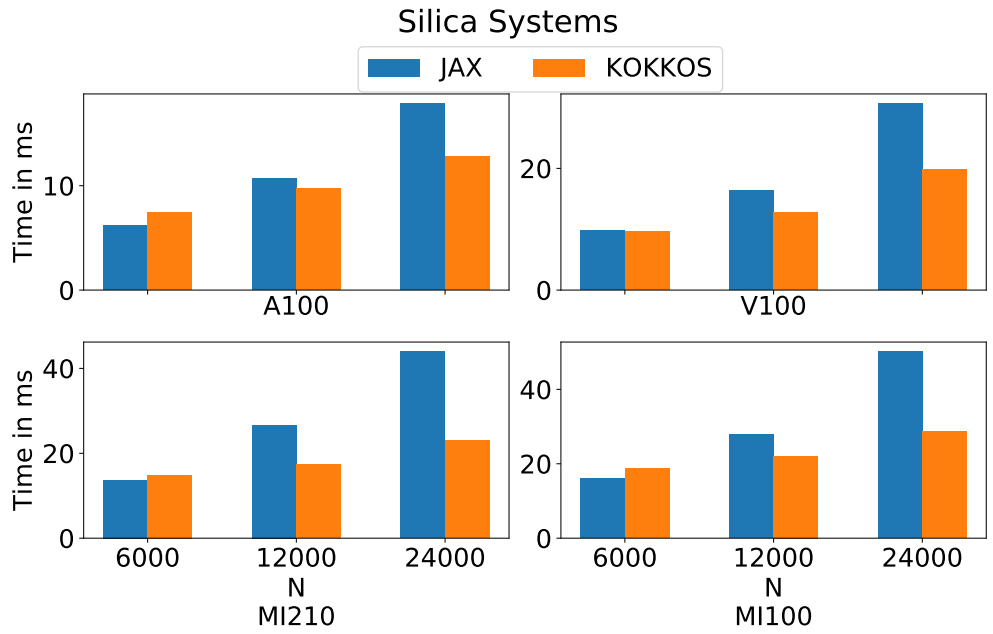


Figure 4.7 Average time per MD step (in ms) for the silica systems with varying sizes.

rations) and the energies of the individual atoms get canceled out. Since the QM9 dataset only contains the absolute energies, we added a new term to the ReaxFF potential to remedy the energy shifts caused by the self-energies of the individual atoms.

$$\begin{aligned} E_{\text{system}} &= E_{\text{ReaxFF}} + E_{\text{self-energy}} \\ E_{\text{self-energy}} &= \sum_{i=1}^{N} s_i \end{aligned} \tag{4.2}$$

In Eq. (4.2), E_{ReaxFF} is the original ReaxFF potential designed to capture the interaction related terms and $E_{\text{self-energy}}$ is the newly added parameterized self-energy term to capture the energy shifts, and s_i is the self energy of atom i solely determined by the atom type. Hence, the new term only contains 4 parameters as there are 4 atom types in the modified QM9 dataset. In total, around 1100 ReaxFF parameters are optimized during the training. The training is performed on an A100 GPU with each epoch taking approximately 8 seconds. Fig. 4.8 shows the mean absolute error (MAE) per epoch. Since the ReaxFF model has a relatively small number of parameters compared to most modern ML methods, the training and test MAE perfectly overlap throughout the training. The final MAE of the model on the test data is 3.6 kcal/mol. While this is higher than the ideal target of 1 kcal/mol error,

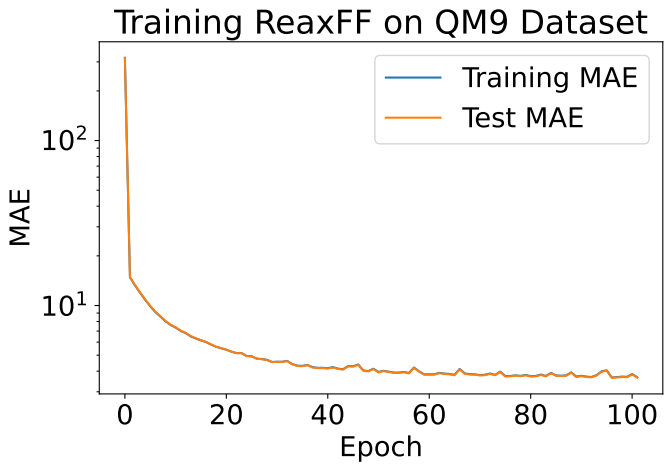


Figure 4.8 Training progress of the ReaxFF model on the QM9 dataset, with the final MAE on the test data being 3.6 kcal/mol.

we note that this is a straight optimization without any fine-tuning to demonstrate the capabilities of the new ReaxFF implementation.

4.4 Concluding Remarks

With the accelerator landscape changing rapidly and becoming more complex, cross platform compilers gain more importance as they enable the same codebase to be used on different architectures. By leveraging modern machine learning cyber-infrastructure, we developed a new JAX-based ReaxFF implementation that is easy-to-maintain, hardware portable, performant, and versatile. Using auto-differentiation, forces in MD simulations are computed directly from energy functions implemented in Python without requiring any extra coding. It also allows the same code to be used for both MD simulations and parameter optimization which are both essential to study any system of interest with ReaxFF. While Kokkos is an another cross-plotform solution, it lacks auto-differentiation and batching optimization capabilities. Although it is more performant for bigger molecules, the JAX implementation is faster for small ones while also providing new functionalities.

CHAPTER 5

UNCERTAINTY QUANTIFICATION METHODS FOR MACHINE LEARNING POTENTIALS

As mentioned previously, machine learning potentials could achieve a high level of accuracy when trained on comprehensive datasets. Quantum mechanical (QM) methods can be employed to compute target observables such as energies, charges, or forces for specific molecules to form a dataset; however, sampling chemically relevant molecules from the vast chemical space remains a formidable challenge. When training a model to study a single chemistry of interest, the relevant data could be collected by a domain expert. Nevertheless, constructing a dataset for a more general model that could be used to study a diverse variety of systems is not feasible using this approach. Additionally, any sampling method relying solely on QM methods becomes computationally prohibitive due to their significant costs.

To address these limitations, active learning based data generation methods have been employed to train machine learning potentials [97, 98, 74, 37]. In active learning, the chemical space is explored by running a sampling method such as molecular dynamics simulations using a machine learning model. When the prediction uncertainty for a given sample exceeds a certain threshold, the target observables for the sample are collected and added to the training data. The model used for sampling is retrained when there is enough new data. The process is repeated until a certain criterion is met. The choice of UQ metric is important for this procedure as it determines where the model fails in the sampled space. In this section, we propose and compare different uncertainty quantification (UQ) methods to be used for active learning. We utilized the previously introduced ANI model as the MLP model for demonstration but the investigated UQ methods could be easily extended to work with other MLPs. Our main motivation is to explore cheaper alternatives to the ensemble based UQ as it is computationally expensive.

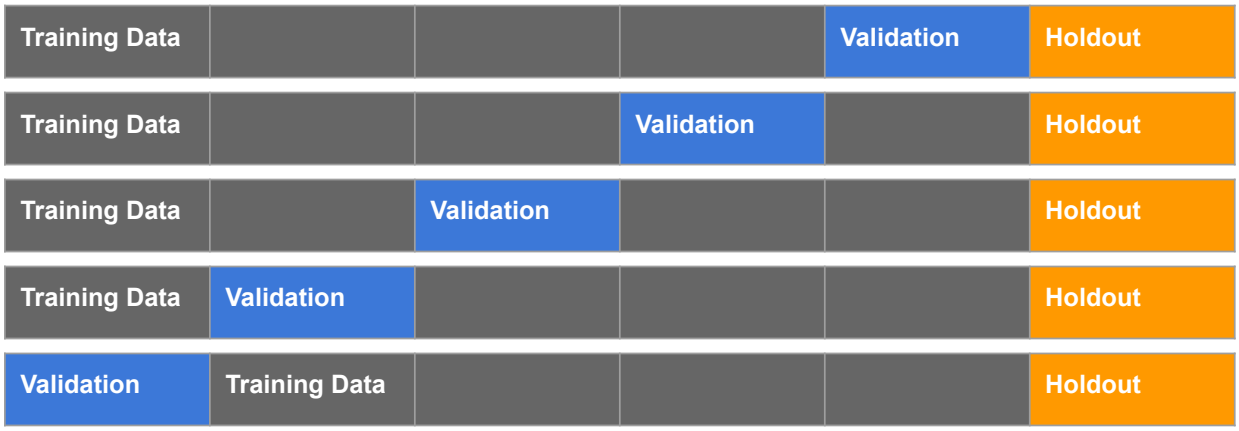


Figure 5.1 Ensemble training with cross-validation.

5.1 Methods

5.1.1 Ensemble Based Uncertainty Quantification

Combining multiple predictors to improve performance is a widely used approach in literature. A popular example is the random forest method, which combines multiple decision trees (weak learners) to make predictions [12]. Another way to train ensembles is by using bootstrapping. In this method, each member of the ensemble is trained on a different set of samples from the original training data, with resampling being performed with replacement. However, in cross-validation based ensemble training, the resampling is done without replacement as illustrated in Fig. 5.1. This lowers the amount of data each model receives but increases overall diversity which is preferred for the uncertainty estimation.

To estimate the uncertainty, multiple models are trained using cross-validation, and the standard deviation of their predictions for a given sample is used as the uncertainty metric. Despite its simplicity, ensemble-based uncertainty estimation has proven to be effective for various classification and regression tasks [55]. Smith et al. successfully employed this approach to iteratively extend the initial training data using active learning [97].

For a given molecule with N atoms, the ensemble disagreement based uncertainty metric (UQ_{ens}) is defined as

$$UQ_{ens} = \frac{\sigma}{\sqrt{N}} \tag{5.1}$$

where σ is the standard deviation (STD) of the predictions of the models in the ensemble.

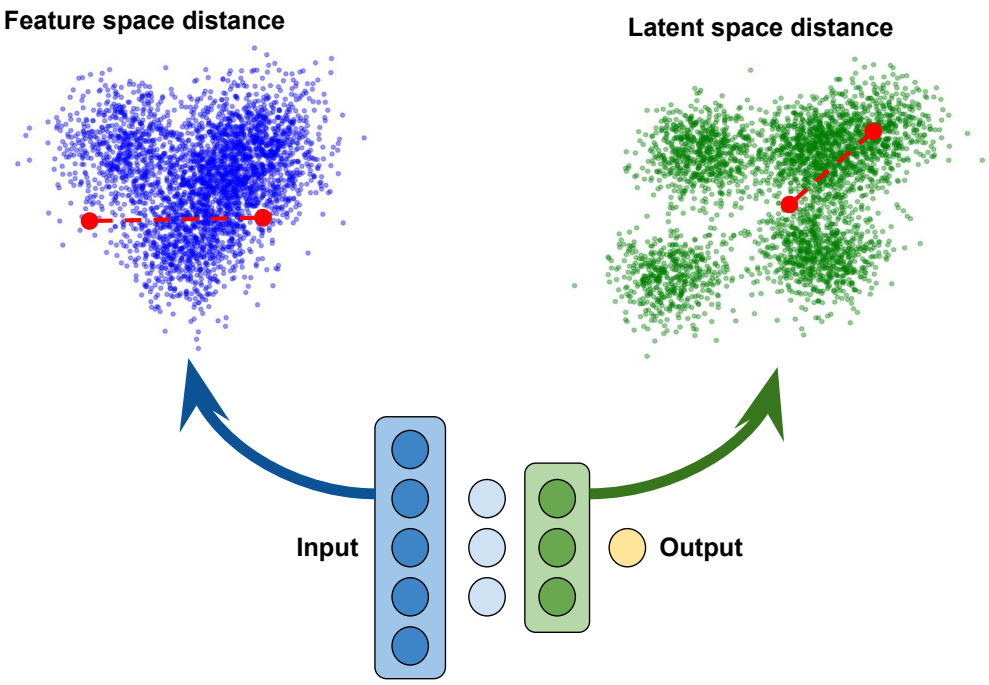


Figure 5.2 Feature and latent space distance.

STD is divided by the square root of the number of atoms to normalize the metric so that it is not biased towards larger molecules as further explained in [97].

Despite its simplicity and performance as a UQ metric, it is computationally expensive in both training and inference time since it employs multiple models. The time cost could be hidden by using a separate GPU for each model, but it increases the total amount of resources needed.

5.1.2 Nearest Neighbor Based Uncertainty Quantification

An alternative to ensemble based UQ, distance to the training data is proven to be an effective UQ metric [95, 58]. Intuitively, predictions for samples that are dissimilar to the training data are expected to be inaccurate. Hence, it is expected that the distance to the training data is highly correlated with the prediction uncertainty. If there is single descriptive vector (fingerprint) per molecule and a given molecule has descriptor x, the uncertainty of a model M trained on X_{train} can be defined as

$$\text{Dist}(x, X_{train}) = \underset{x' \in X_{train}}{\arg\min}\, d(x, x')$$

where d is a distance metric, typically Euclidean distance. To improve its robustness, K-Nearest Neighbors (K-NN) algorithm could be used which calculates the average distance to the closest k elements from the training data. A greater distance suggests that the corresponding chemical space is less represented in the training data. Moreover, since this method requires only a single model to gather latent representations and calculate the K-NN distance, it is computationally less intensive compared to the ensemble-based approach.

Unlike handcrafted descriptors, machine learning models typically learn target specific representations from data. Janet et al. proposed using these latent representations to calculate distances, as these representations are specifically tailored for the given task [45]. Typically, deep learning models have many layers to improve the learning capacity. The vector before the output layer is shown to be an effective representation to use for distance based UQ [45].

MLPs typically employ distinct neural networks for each atom type, with these networks predicting the corresponding atomic energies. The total energy for a molecule is calculated by summing up the predicted atomic energies. Hence, there is no single latent vector representing the entire molecule unlike the molecular representations explored in [45]. Our proposed approach involves calculating the latent space distance to the training data at the atomic level. This is accomplished by first gathering type-specific latent vectors from the training data for each atom type. Subsequently, a K-Nearest Neighbors (K-NN) model is constructed for each atom type using the collected data. To compute the distance for a given molecule, each atom in the molecule is processed, and its atom type is identified. Using the corresponding K-NN model, the distance for each atom is calculated. Finally, the UQ metric is determined by averaging the distances calculated for all the atoms in the molecule. Algorithm 5.1 provides a concise summary of this approach. If a molecule contains atoms with latent representations that are dissimilar to the training data, it indicates a level of uncertainty which is expected to lead to inaccurate predictions.

Algorithm 5.1 Latent space distance using K-NN.

$N_{type} \leftarrow$ Number of available atom types
$M \leftarrow$ Set of atomic NNs (in total, N_{type} models)
$X_{train} \leftarrow$ Training data
$k_{neigh} \leftarrow$ Number of neighbors for K-NN
for each $i = 1, 2, \ldots N_{type}$ **do**
    $V_i \leftarrow$ Collect latent vectors for type i from X_{train} using M_i
    $\text{KNN}_i \leftarrow$ Create K-NN model from V_i
end for
    ▷ After creating the K-NN models from the training data, the UQ metric could be calculated in the following way:
$x_{test} \leftarrow$ Given molecule to calculate the UQ metric for
$d_x \leftarrow 0$
for each $a \in x$ **do**
    $t \leftarrow$ Atom type of a
    $v_a \leftarrow$ Latent vector for atom a using M_t
    $d_a \leftarrow$ Calculate the distance using KNN_t and v_a
    $d_x \leftarrow d_x + d_a$
end for
$d_x \leftarrow d_x/\text{len}(x)$ (Calculate the average distance)

5.1.3 Autoencoder Based Uncertainty Quantification

As previously explained, the K-NN approach requires storing latent vectors after training to calculate the distance to the training data. During inference, it is necessary to find the nearest neighbors for each atom. However, if the training data is substantial, the latent vectors might not fit into memory, potentially drastically increasing the neighbor search time. To remedy this, we propose an alternative approach using autoencoders to estimate uncertainty.

Employing variations of autoencoders is a well-known method for detecting outliers or estimating uncertainty [4, 17]. Typically, autoencoders are trained in an unsupervised manner where the labels are not available. The model is trained using reconstruction loss which is typically L2 norm of the difference between the input vector and the reconstructed vector. Reconstruction minimization has also been explored as an auxiliary task to improve generalization during supervised model training [57]. Le et al. show that trying to reconstruct the input vector from the latent space regularizes the latent space and improves generalization

performance [57].

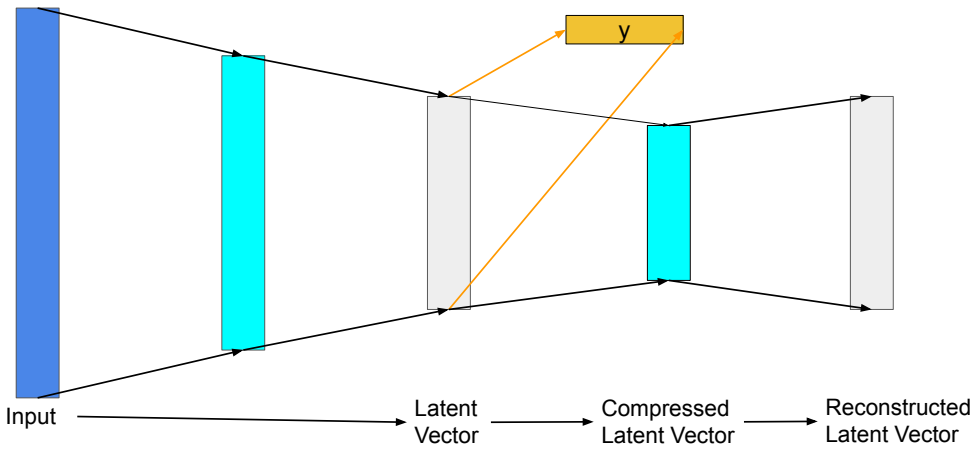


Figure 5.3 Multi-task autoencoder training.

As illustrated in Fig. 5.3, instead of reconstructing the input vector, we pass the latent vector through a bottleneck to further compress it and then reconstruct the latent vector from this compressed version. We favored this method since there may not be an explicit input vector for some MLPs such as graph neural networks (GNNs) [34]. While we primarily studied ANI models that create explicit input vectors using predefined functions like Behler and Parrinello symmetry functions [9], our approach is easily adaptable for different MLPs as well.

To train a model using reconstruction loss, we employ a loss function that combines both energy and reconstruction loss:

$$L(W) = \sum_i \left[\left\| E_i - \sum_j \hat{E_{ij}} \right\|^2 + \alpha \sum_j \|\hat{x_{ij}} - x_{ij}\|^2 \right] \tag{5.2}$$

where W is the model weights, i is the molecule index and j is the atom index. For the energy loss part, E_i is the reference energy and $\hat{E_{ij}}$ is the atomic energy prediction. For the reconstruction loss part, x_{ij} is the latent vector for atom j and $\hat{x_{ij}}$ is the reconstructed vector. Hyperparameter α controls the weight of the reconstruction loss.

For a molecule with N atoms, the equation below provides the molecular reconstruction

error to serve as a UQ metric:

$$\mathrm{UQ}_{\mathrm{reconst}} = \frac{1}{N} \sum_{j} \left\| \hat{x_j} - x_j \right\|^2 . \tag{5.3}$$

We should note that just as there is a distinct neural network for each atom type, the autoencoder component is also specific to each type.

5.2 Evaluation

5.2.1 Datasets

We have picked 2 different datasets to evaluate and compare the methods described in the earlier section.

5.2.1.1 QM9 dataset

The QM9 dataset comprises approximately 130k energy-equilibrated molecules made up of C, H, O, N, and F atoms. Each molecule contains up to 9 heavy atoms. There are 13 properties per molecule calculated at the B3LYP/6-31G(2df,p) level of theory [79]. We only used potential energy during training and evaluation. Although it lacks diversity, it is a well-studied benchmark dataset.

5.2.1.2 ANI-1ccx dataset

The ANI-1ccx dataset contains nearly 500k diverse molecular conformations consisting of C, H, O and N atoms. It is an intelligently selected 10% sub-sample of the ANI-1x data set, but recomputed with an accurate coupled cluster (approximately CCSD(T)/CBS) level of theory [99].

5.2.2 Experiments

To compare the UQ metrics, we designed two types of experiments. For the first set of experiments, we split each dataset into train, test and validation sets then compare how well the UQ metrics correlate with the error in the test data. Although comparing these methods in a static way provides valuable insights, in an active learning environment, the training data is iteratively extended based on model uncertainty evaluation. Hence, we designed another experiment that can provide further information about the methods in a dynamic

environment where the data is constantly updated. The details are provided in the later sections.

We used an ensemble of 5 models for the ensemble-based UQ. For K-NN, we set K to 1. Lastly, we set α to 0.005 for the autoencoder approach. Regarding the training-related hyperparameters, we set the batch size to 256, the learning rate to 0.001, and trained for 400 epochs. Additionally, we utilized a learning rate scheduler that reduces the learning rate by 25% when the validation loss does not improve for 10 consecutive epochs. The details about the model architecture are provided in Fig. C.1 of Appendix C.

The reported errors are normalized in the following way:

$$\text{Error}_{\text{norm}} = \frac{|E - \hat{E}|}{\sqrt{N}}$$

where E and $\hat{E}$ are target and predicted energies respectively, and N is the number of atoms for a given molecule.

5.2.2.1 Static Experiments

We split each dataset into the training and testing data at an 80/20 ratio. For single model UQ metrics (K-NN and autoencoder based), 20% of training data is used as validation data and for ensemble training, cross-validation is used. After the training, the UQ metrics are evaluated on the test data.

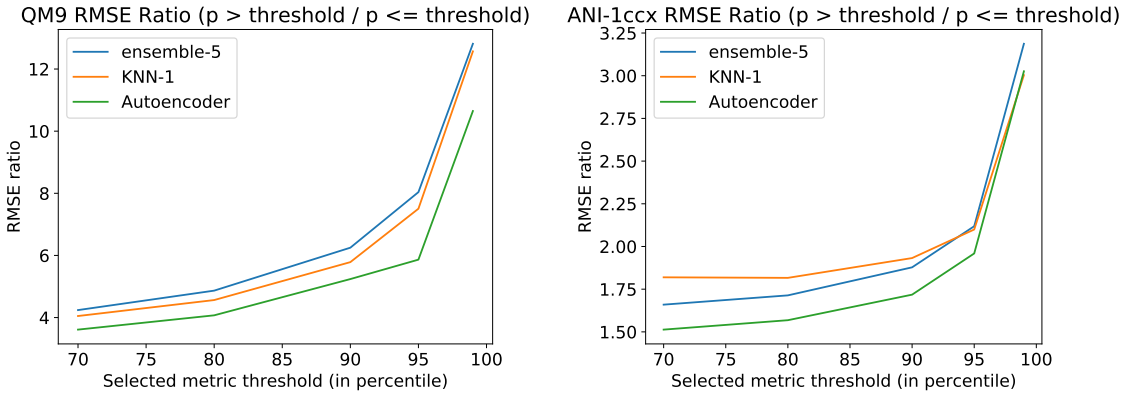


Figure 5.4 RMSE ratios for QM9 and ANI1CCX. The ratios are calculated by dividing the test data into two groups based on the magnitude of the UQ metric, higher is better.

The Root Mean Square Error (RMSE) ratios displayed in Fig. 5.4 were derived as follows: First, the samples in the test data were sorted by the magnitude of the specified UQ metric. Then, based on a chosen percentile rank threshold, the data was divided into two distinct groups. Samples with uncertainty higher than the threshold were placed in the first group, while the remaining samples were allocated to the second group. The ratio of these two groups provides the values shown in Fig. 5.4. If the UQ metric is highly correlated with the absolute error, it effectively differentiates between high-error and low-error cases, resulting in a higher RMSE ratio. In contrast, a UQ metric that is randomly distributed would lead to an RMSE ratio of approximately 1, as the division between the two groups would be essentially arbitrary, causing the RMSE values within each group to be very similar to each other and to the overall RMSE.

As seen in the figure, the ensemble-based UQ delivers the best results for both datasets, followed by K-NN, and lastly, the autoencoder-based UQ. The difference between the ensemble-based and K-NN based UQ metrics is very minimal for both datasets. While all UQ metrics perform more effectively on the QM9 datasets, the performance disparity between the metrics is less pronounced on the more diverse ANI-1ccx dataset. A possible explanation for this observation could be the extensive use of active learning in the generation of the ANI-1ccx dataset [99]. This heightened diversity might lead the learned models to be more balanced, potentially resulting in a suboptimal performance of the UQ metrics.

Method	Dataset	Normalized RMSE (kcal/mol)
Ensemble	QM9	0.2084
K-NN	QM9	0.2288
Autoencoder	QM9	0.2543
Ensemble	ANI-1ccx	0.7243
K-NN	ANI-1ccx	0.8515
Autoencoder	ANI-1ccx	0.8531

Table 5.1 Normalized RMSE in the test data for different setups.

The RMSE values on the evaluated datasets and models are presented in Table 5.1.

Since the ensemble approach aggregates predictions from multiple models, its RMSE on the test data is lower than that of the single-model predictions used in the K-NN and autoencoder approaches. Additionally, the RMSE values for the autoencoder approach are slightly higher due to the complexity of learning two different tasks simultaneously, as it employs a loss function detailed in Eq. (5.2). This dual-task learning can introduce a trade-off that potentially impairs the optimization of the RMSE.

5.2.2.2 Dynamic Experiments

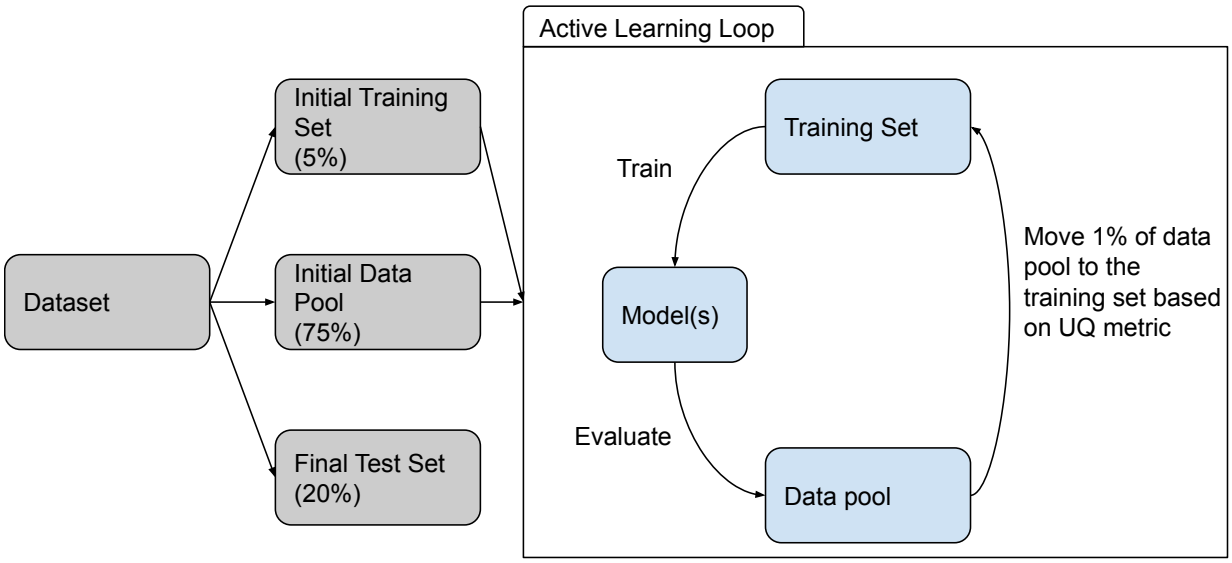


Figure 5.5 Iterative data refinement.

To more effectively compare the UQ metrics, we devised the method depicted in Fig. 5.5. For each dataset, we first split it randomly into three segments. The initial training set is used to start the active learning process, while the initial data pool augments the training data using a chosen UQ metric. The final test set monitors the performance throughout the active learning phase. After training, a segment of the data pool is selected based on assessed uncertainty and added to the training data. In every iteration, uncertainty for all samples in the data pool is evaluated. Subsequently, a subset with the highest uncertainty augments the training data. For a more comprehensive comparison, we also assessed random selection and selection based on normalized absolute error. To keep the RMSE metric more

comparable across the UQ metrics, a single model without the reconstruction loss is trained with the training data available at each cycle for every UQ metric and Fig. 5.6 is created with errors reported from these models.

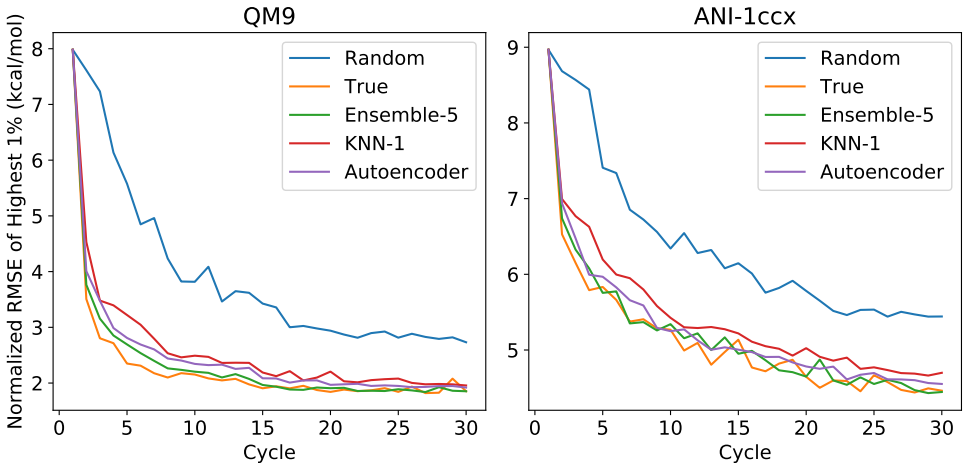


Figure 5.6 Change of the RMSE corresponding to the top 1% of test samples with the highest normalized error for each cycle.

As seen in Fig. 5.6, while random selection lags behind the investigated UQ metrics, these UQ metrics perform similarly to the selection based on normalized error. Though the differences are subtle, the ensemble approach performs slightly better, while the K-NN approach yields marginally inferior results.

5.3 Concluding Remarks

Since the quality of data is crucial for the generalizability of MLPs, active learning approaches have gained more importance in generating such data. While the existing approaches often utilize ensemble disagreement as the uncertainty metric, we have explored and compared different methods, including K-NN and autoencoder-based uncertainty quantification. Ensemble disagreement yields superior results but is computationally more demanding as multiple models must be trained and used for inference. We offer more economical alternatives and demonstrate their efficacy on the QM9 and ANI-1ccx datasets through two distinct experiments. Although our focus is primarily on the ANI architecture, we plan to conduct similar evaluations for newer MLP architectures in the future since the UQ metrics

we provide are easily adaptable to different MLPs with minimal changes.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we have sought to enhance the fidelity and usability of molecular models using hybridization and machine learning methods. Despite the multitude of modeling approaches available for atomistic simulations, each with its unique trade-offs, no single approach is universally applicable. We introduced ReaxFF/AMBER, a hybrid method that melds the reactive force field ReaxFF with the classical force field AMBER, thereby broadening the capabilities of the latter by adding dynamic bonds and charges. However, due to the complex nature of ReaxFF, the existing parameterization tools lacked the efficiency and speed needed for the expansive scope of biological simulations enabled by ReaxFF/AMBER. This led us to develop JAX-ReaxFF, a gradient-based parameterization tool for ReaxFF, by leveraging the versatile JAX framework. With this new tool, we are able to cut down the training time drastically compared to the existing approaches. While JAX-ReaxFF is designed for fast parameter optimization, it is not suitable for running MD simulations. To leverage the vast machine learning ecosystem for easier development, maintenance, and seamless integration with existing Python libraries, we have integrated an end-to-end differentiable implementation of ReaxFF into JAX-MD, which is a JAX based MD library. While offering the mentioned benefits, the performance of the new implementation is on par with the widely used KOKKOS-based ReaxFF implementation on modern NVIDIA and AMD GPUs.

The wide success and adaptation of the machine learning techniques and libraries have led to the development of a new class of force fields called machine learning potentials. Unlike their classical and reactive counterparts, MLPs hinge on high quality training data to offer QM level accuracy while being drastically faster. We compared different uncertainty quantification approaches as they are essential for the active learning methods used to generate training data with minimal supervision. Also, UQ metrics help understanding where MLPs fail when running MD simulations as they are hard to examine compared to the clas-

sical force fields. Although our preliminary experiments predominantly centered on the ANI architecture, future endeavors will branch out to newer MLP designs, including message passing and equivariant neural networks.

BIBLIOGRAPHY

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] Hasan Metin Aktulga, Joseph C Fogarty, Sagar A Pandit, and Ananth Y Grama. Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques. *Parallel Computing*, 38(4-5):245–259, 2012.

[3] Hasan Metin Aktulga, Sagar A Pandit, Adri CT van Duin, and Ananth Y Grama. Reactive molecular dynamics: Numerical methods and algorithmic techniques. *SIAM Journal on Scientific Computing*, 34(1):C1–C23, 2012.

[4] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

[5] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, John Quan, George Papamakarios, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Luyu Wang, Wojciech Stokowiec, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. See http://github.com/deepmind/jax.

[6] Jonathan T Barron. Continuously differentiable exponential linear units. *arXiv preprint arXiv:1704.07483*, 2017.

[7] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[8] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):1–11, 2022.

[9] Jörg Behler and Michele Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical review letters*, 98(14):146401, 2007.

[10] Erik Bitzek, Pekka Koskinen, Franz Gähler, Michael Moseler, and Peter Gumbsch. Structural relaxation made simple. *Physical review letters*, 97(17):170201, 2006.

[11] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. *Version 0.2*, 5:14–24, 2018.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[13] Donald W Brenner, Olga A Shenderova, Judith A Harrison, Steven J Stuart, Boris Ni, and Susan B Sinnott. A second-generation reactive empirical bond order (rebo) potential energy expression for hydrocarbons. *Journal of Physics: Condensed Matter*, 14(4):783, 2002.

[14] Patrick Bultinck, Wilfried Langenaeker, Philippe Lahorte, Frank De Proft, Paul Geerlings, Michel Waroquier, and JP Tollenaere. The electronegativity equalization method i: Parametrization and validation for atomic charge calculations. *The Journal of Physical Chemistry A*, 106(34):7887–7894, 2002.

[15] David A Case, H Metin Aktulga, Kellon Belfon, IY Ben-Shalom, Scott R Brozell, DS Cerutti, TE Cheatham III, GA Cisneros, VWD Cruzeiro, TA Darden, et al. *Amber 2021*. University of California Press, 2021.

[16] David A Case, Thomas E Cheatham III, Tom Darden, Holger Gohlke, Ray Luo, Kenneth M Merz Jr, Alexey Onufriev, Carlos Simmerling, Bing Wang, and Robert J Woods. The amber biomolecular simulation programs. *Journal of computational chemistry*, 26(16):1668–1688, 2005.

[17] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. Autoencoder-based network anomaly detection. In *2018 Wireless telecommunications symposium (WTS)*, pages 1–5. IEEE, 2018.

[18] Kimberly Chenoweth, Adri CT Van Duin, and William A Goddard. Reaxff reactive force field for molecular dynamics simulations of hydrocarbon oxidation. *The Journal of Physical Chemistry A*, 112(5):1040–1053, 2008.

[19] Chaitanya M Daksha, Jejoon Yeon, Sanjib C Chowdhury, and John W Gillespie Jr. Automated reaxff parametrization using machine learning. *Computational Materials Science*, 187:110107, 2021.

[20] Tom Darden, Darrin York, and Lee Pedersen. Particle mesh ewald: An n log (n) method for ewald sums in large systems. *The Journal of chemical physics*, 98(12):10089–10092, 1993.

[21] Mark Dittner, Julian Müller, Hasan Metin Aktulga, and Bernd Hartke. Efficient global optimization of reactive force-field parameters. *Journal of computational chemistry*, 36(20):1550–1561, 2015.

[22] Marcus Elstner, Dirk Porezag, G Jungnickel, J Elsner, M Haugk, Th Frauenheim, Sandor Suhai, and Gotthard Seifert. Self-consistent-charge density-functional tight-binding method for simulations of complex materials properties. *Physical Review B*, 58(11):7260, 1998.

[23] Joseph C Fogarty, Hasan Metin Aktulga, Ananth Y Grama, Adri CT Van Duin, and Sagar A Pandit. A reactive molecular dynamics simulation of the silica-water interface. *The Journal of chemical physics*, 132(17):174704, 2010.

[24] Richard A Friesner. Ab initio quantum chemistry: Methodology and applications. *Proceedings of the National Academy of Sciences*, 102(19):6648–6653, 2005.

[25] Richard A Friesner and Victor Guallar. Ab initio quantum chemical and mixed quantum mechanics/molecular mechanics (qm/mm) methods for studying enzymatic catalysis. *Annu. Rev. Phys. Chem.*, 56:389–427, 2005.

[26] MJ ea Frisch, GW Trucks, H Bernhard Schlegel, GE Scuseria, MA Robb, JR Cheeseman, G Scalmani, VPGA Barone, GA Petersson, HJRA Nakatsuji, et al. Gaussian 16, 2016.

[27] David Furman, Benny Carmeli, Yehuda Zeiri, and Ronnie Kosloff. Enhanced particle swarm optimization algorithm: Efficient training of reaxff reactive force fields. *Journal of chemical theory and computation*, 14(6):3100–3112, 2018.

[28] David Furman and David J Wales. A well-behaved theoretical framework for reaxff reactive force fields. *The Journal of Chemical Physics*, 153(2):021102, 2020.

[29] Joseph J Gajewski and Neal D Conrad. Variable transition state structure in 3, 3-sigmatropic shifts from. alpha.-secondary deuterium isotope effects. *Journal of the American Chemical Society*, 101(22):6693–6704, 1979.

[30] Julian D Gale, Paolo Raiteri, and Adri CT van Duin. A reactive force field for aqueous-calcium carbonate systems. *Physical Chemistry Chemical Physics*, 13(37):16666–16679, 2011.

[31] Bruce Ganem. The mechanism of the claisen rearrangement: déjà vu all over again. *Angewandte Chemie International Edition in English*, 35(9):936–945, 1996.

[32] Jiali Gao. Combined qm/mm simulation study of the claisen rearrangement of allyl vinyl ether in aqueous solution. *Journal of the American Chemical Society*, 116(4):1563–1564, 1994.

[33] Xiang Gao, Farhad Ramezanghorbani, Olexandr Isayev, Justin S Smith, and Adrian E Roitberg. Torchani: A free and open source pytorch-based deep learning implementation of the ani neural network potentials. *Journal of chemical information and modeling*, 60(7):3408–3415, 2020.

[34] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[35] Andreas W Götz, Matthew A Clark, and Ross C Walker. An extensible interface for qm/mm molecular dynamics simulations with amber. *Journal of computational chemistry*, 35(2):95–108, 2014.

[36] Victor Guallar and Frank H Wallrapp. Qm/mm methods: Looking inside heme proteins biochemisty. *Biophysical chemistry*, 149(1-2):1–11, 2010.

[37] Konstantin Gubaev, Evgeny V Podryabinkin, and Alexander V Shapeev. Machine learning of molecular properties: Locality and active learning. *The Journal of chemical physics*, 148(24), 2018.

[38] JonathanáM Guest, J áSimon Craw, MarkáA Vincent, and IanáH Hillier. The effect of water on the claisen rearrangement of allyl vinyl ether: theoretical methods including explicit solvent and electron correlation. *Journal of the Chemical Society, Perkin Transactions 2*, 1(1):71–74, 1997.

[39] Feng Guo, Yu-Shi Wen, Shi-Quan Feng, Xiao-Dong Li, Heng-Shuai Li, Shou-Xin Cui, Zhen-Rong Zhang, Hai-Quan Hu, Gui-Qing Zhang, and Xin-Lu Cheng. Intelligent-reaxff: Evaluating the reactive force field parameters with machine learning. *Computational Materials Science*, 172:109393, 2020.

[40] Berk Hess, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of chemical theory and computation*, 4(3):435–447, 2008.

[41] Martin Hiersemann and Udo Nubbemeyer. *The Claisen rearrangement: methods and applications*. John Wiley & Sons, 2007.

[42] Pierre O Hubin, Denis Jacquemin, Laurence Leherte, and Daniel P Vercauteren. Parameterization of the reaxff reactive force field for a proline-catalyzed aldol reaction. *Journal of Computational Chemistry*, 37(29):2564–2572, 2016.

[43] Christine M Isborn, Andreas W Gotz, Matthew A Clark, Ross C Walker, and Todd J Martínez. Electronic absorption spectra from mm and ab initio qm/mm molecular dynamics: environmental effects on the absorption spectrum of photoactive yellow protein. *Journal of chemical theory and computation*, 8(12):5092–5106, 2012.

[44] Eldhose Iype, Markus Hütter, APJ Jansen, Silvia V Nedea, and CCM Rindt. Parameterization of a reactive force field using a monte carlo algorithm. *Journal of computational chemistry*, 34(13):1143–1154, 2013.

[45] Jon Paul Janet, Chenru Duan, Tzuhsiung Yang, Aditya Nandy, and Heather J Kulik. A quantitative uncertainty metric controls error in neural network-driven chemical discovery. *Chemical science*, 10(34):7913–7922, 2019.

[46] Andres Jaramillo-Botero, Saber Naserifar, and William A Goddard III. General multiobjective force field optimization framework, with application to reactive force fields for silicon carbide. *Journal of Chemical Theory and Computation*, 10(4):1426–1439, 2014.

[47] William L Jorgensen, Jayaraman Chandrasekhar, Jeffry D Madura, Roger W Impey, and Michael L Klein. Comparison of simple potential functions for simulating liquid water. *The Journal of chemical physics*, 79(2):926–935, 1983.

[48] Jaewoon Jung, Wataru Nishima, Marcus Daniels, Gavin Bascom, Chigusa Kobayashi, Adetokunbo Adedoyin, Michael Wall, Anna Lappala, Dominic Phillips, William Fischer, et al. Scaling molecular dynamics beyond 100,000 processor cores for large-scale biophysical simulations. *Journal of computational chemistry*, 40(21):1919–1930, 2019.

[49] J Kästner. Umbrella sampling. wires computational molecular science 1: 932–942, 2011.

[50] Mehmet Cagri Kaymak, Ali Rahnamoun, Kurt A. O'Hearn, Adri CT Van Duin, Kenneth M Merz Jr, and Hasan Metin Aktulga. JAX-ReaxFF: A Gradient-Based Framework for Fast Optimization of Reactive Force Fields. *Journal of Chemical Theory and Computation*, 18(9):5181–5194, 2022.

[51] Mehmet Cagri Kaymak, Samuel S Schoenholz, Ekin D Cubuk, Kurt A O'Hearn, Kenneth M Merz Jr, and Hasan Metin Aktulga. End-to-end differentiable reactive molecular dynamics simulations using jax. In *International Conference on High Performance Computing*, pages 202–219. Springer, 2023.

[52] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[53] Sudhir B Kylasa, Hasan Metin Aktulga, and Ananth Y Grama. Puremd-gpu: A reactive molecular dynamics simulation package for gpus. *Journal of Computational Physics*, 272:343–359, 2014.

[54] Matthew R LaBrosse, J Karl Johnson, and Adri CT van Duin. Development of a transferable reactive force field for cobalt. *The Journal of Physical Chemistry A*, 114(18):5855–5861, 2010.

[55] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[56] Henrik R Larsson, Adri CT van Duin, and Bernd Hartke. Global optimization of parameters in the reactive force field reaxff for sioh. *Journal of computational chemistry*, 34(25):2178–2189, 2013.

[57] Lei Le, Andrew Patterson, and Martha White. Supervised autoencoders: Improving generalization performance with unsupervised regularizers. *Advances in neural information processing systems*, 31, 2018.

[58] Ruifeng Liu and Anders Wallqvist. Molecular similarity-based domain applicability metric efficiently identifies out-of-domain compounds. *Journal of Chemical Information and Modeling*, 59(1):181–189, 2018.

[59] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[60] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[61] Xiya Lu, Dong Fang, Shingo Ito, Yuko Okamoto, Victor Ovchinnikov, and Qiang Cui. Qm/mm free energy simulations: Recent progress and challenges. *Molecular simulation*, 42(13):1056–1078, 2016.

[62] Rita P Magalhães, Henriques S Fernandes, and Sérgio F Sousa. Modelling enzymatic mechanisms with qm/mm approaches: current status and future challenges. *Israel Journal of Chemistry*, 60(7):655–666, 2020.

[63] Madushanka Manathunga, Yipu Miao, Dawei Mu, Andreas W Gotz, and Kenneth M Merz Jr. Parallel implementation of density functional theory methods in the quantum interaction computational kernel program. *Journal of Chemical Theory and Computation*, 16(7):4315–4326, 2020.

[64] Luca Monticelli and D Peter Tieleman. Force fields for classical molecular dynamics. *Biomolecular simulations: Methods and protocols*, pages 197–213, 2013.

[65] Wilfried J Mortier, Swapan K Ghosh, and S Shankar. Electronegativity-equalization method for the calculation of atomic charges in molecules. *Journal of the American Chemical Society*, 108(15):4315–4320, 1986.

[66] RL Mozzi and n BE Warren. The structure of vitreous silica. *Journal of Applied Crystallography*, 2(4):164–172, 1969.

[67] Julian Mülller and Bernd Hartke. Reaxff reactive force field for disulfide mechanochemistry, fitted to multireference ab initio data. *Journal of chemical theory and computation*, 12(8):3913–3925, 2016.

[68] Hiroya Nakata and Shandan Bai. Development of a new parameter optimization scheme for a reactive force field based on a machine learning approach. *Journal of computational chemistry*, 40(23):2000–2012, 2019.

[69] Kurt A O'Hearn, Abdullah Alperen, and Hasan Metin Aktulga. Fast solvers for charge distribution models on shared memory platforms. *SIAM Journal on Scientific Computing*, 42(1):C1–C22, 2020.

[70] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[71] David A Pearlman, David A Case, James W Caldwell, Wilson S Ross, Thomas E Cheatham III, Steve DeBolt, David Ferguson, George Seibel, and Peter Kollman. Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Computer Physics Communications*, 91(1-3):1–41, 1995.

[72] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. *Journal of computational chemistry*, 26(16):1781–1802, 2005.

[73] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[74] Evgeny V Podryabinkin, Evgeny V Tikhonov, Alexander V Shapeev, and Artem R Oganov. Accelerating crystal structure prediction by machine-learning interatomic potentials with active learning. *Physical Review B*, 99(6):064114, 2019.

[75] GP Purja Pun and Y Mishin. Embedded-atom potential for hcp and fcc cobalt. *Physical Review B*, 86(13):134116, 2012.

[76] Zhuoran Qiao, Matthew Welborn, Animashree Anandkumar, Frederick R Manby, and Thomas F Miller III. OrbNet: Deep learning for quantum chemistry using symmetry-adapted atomic-orbital features. *The Journal of chemical physics*, 153(12):124111, 2020.

[77] Ali Rahnamoun, Mehmet Cagri Kaymak, Madushanka Manathunga, Andreas W Götz, Adri CT Van Duin, Kenneth M Merz Jr, and Hasan Metin Aktulga. Reaxff/amber—a framework for hybrid reactive/nonreactive force field molecular dynamics simulations. *Journal of chemical theory and computation*, 16(12):7645–7654, 2020.

[78] Ali Rahnamoun, Kurt A O'Hearn, Mehmet Cagri Kaymak, Zhen Li, Kenneth M Merz Jr, and Hasan Metin Aktulga. A polarizable cationic dummy metal ion model. *The Journal of Physical Chemistry Letters*, 13(23):5334–5340, 2022.

[79] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1(1):1–7, 2014.

[80] Anthony K Rappe and William A Goddard III. Charge equilibration for molecular dynamics simulations. *The Journal of Physical Chemistry*, 95(8):3358–3363, 1991.

[81] SCM ReaxFF. Theoretical chemistry, 2020.

[82] Amit Sabne. Xla: Compiling machine learning for peak performance. 2020.

[83] RT Sanderson. An interpretation of bond lengths and a classification of bonds. *Science*, 114(2973):670–672, 1951.

[84] Samuel Schoenholz and Ekin Dogus Cubuk. Jax md: a framework for differentiable physics. *Advances in Neural Information Processing Systems*, 33:11428–11441, 2020.

[85] Frederic W Schuler and George W Murphy. The kinetics of the rearrangement of vinyl allyl ether1. *Journal of the American Chemical Society*, 72(7):3155–3159, 1950.

[86] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. *Advances in neural information processing systems*, 30, 2017.

[87] Gustavo de M Seabra, Ross C Walker, Marcus Elstner, David A Case, and Adrian E Roitberg. Implementation of the scc-dftb method for hybrid qm/mm simulations within the amber molecular dynamics package. *The Journal of Physical Chemistry A*, 111(26):5655–5664, 2007.

[88] Thomas P Senftle, Sungwook Hong, Md Mahbubul Islam, Sudhir B Kylasa, Yuanxia Zheng, Yun Kyung Shin, Chad Junkermeier, Roman Engel-Herbert, Michael J Janik, Hasan Metin Aktulga, et al. The reaxff reactive force-field: development, applications and future directions. *npj Computational Materials*, 2(1):1–14, 2016.

[89] Mert Y Sengul, Yao Song, Nadire Nayir, Yawei Gao, Ying Hung, Tirthankar Dasgupta, and Adri CT van Duin. Indeedopt: a deep learning-based reaxff parameterization framework. *npj Computational Materials*, 7(1):1–9, 2021.

[90] Hans Martin Senn and Walter Thiel. Qm/mm methods for biomolecular systems. *Angewandte Chemie International Edition*, 48(7):1198–1229, 2009.

[91] Alexander Shapeev, Konstantin Gubaev, Evgenii Tsymbalov, and Evgeny Podryabinkin. Active learning and uncertainty estimation. *Machine Learning Meets Quantum Physics*, pages 309–329, 2020.

[92] David E Shaw, Peter J Adams, Asaph Azaria, Joseph A Bank, Brannon Batson, Alistair Bell, Michael Bergdorf, Jhanvi Bhatt, J Adam Butts, Timothy Correia, et al. Anton 3: twenty microseconds of molecular dynamics simulation before lunch. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2021.

[93] David E Shaw, Ron O Dror, John K Salmon, JP Grossman, Kenneth M Mackenzie, Joseph A Bank, Cliff Young, Martin M Deneroff, Brannon Batson, Kevin J Bowers, et al. Millisecond-scale molecular dynamics simulations on anton. In *Proceedings of the conference on high performance computing networking, storage and analysis*, pages 1–11, 2009.

[94] Ganna Shchygol, Alexei Yakovlev, Toman Trnka, Adri CT Van Duin, and Toon Verstraelen. Reaxff parameter optimization with monte-carlo and evolutionary algorithms: Guidelines and insights. *Journal of Chemical Theory and Computation*, 15(12):6799–6812, 2019.

[95] Robert P Sheridan, Bradley P Feuston, Vladimir N Maiorov, and Simon K Kearsley. Similarity to molecules in the training set is a good discriminator for prediction accuracy in qsar. *Journal of chemical information and computer sciences*, 44(6):1912–1928, 2004.

[96] Justin S Smith, Olexandr Isayev, and Adrian E Roitberg. Ani-1, a data set of 20 million calculated off-equilibrium conformations for organic molecules. *Scientific data*, 4(1):1–8, 2017.

[97] Justin S Smith, Ben Nebgen, Nicholas Lubbers, Olexandr Isayev, and Adrian E Roitberg. Less is more: Sampling chemical space with active learning. *The Journal of chemical physics*, 148(24):241733, 2018.

[98] Justin S Smith, Benjamin Nebgen, Nithin Mathew, Jie Chen, Nicholas Lubbers, Leonid Burakovsky, Sergei Tretiak, Hai Ah Nam, Timothy Germann, Saryu Fensin, et al. Automated discovery of a robust interatomic potential for aluminum. *Nature communications*, 12(1):1257, 2021.

[99] Justin S Smith, Roman Zubatyuk, Benjamin Nebgen, Nicholas Lubbers, Kipton Barros, Adrian E Roitberg, Olexandr Isayev, and Sergei Tretiak. The ani-1ccx and ani-1x data sets, coupled-cluster and density functional theory properties for molecules. *Scientific data*, 7(1):1–10, 2020.

[100] James JP Stewart. Optimization of parameters for semiempirical methods v: Modification of nddo approximations and application to 70 elements. *Journal of Molecular modeling*, 13:1173–1213, 2007.

[101] JJPRB Tersoff. Modeling solid-state chemistry: Interatomic potentials for multicomponent systems. *Physical review B*, 39(8):5566, 1989.

[102] Jos Thijssen. *Computational physics*. Cambridge university press, 2007.

[103] Aidan P Thompson, H Metin Aktulga, Richard Berger, Dan S Bolintineanu, W Michael Brown, Paul S Crozier, Pieter J in't Veld, Axel Kohlmeyer, Stan G Moore, Trung Dac Nguyen, et al. Lammps-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171, 2022.

[104] Aidan P Thompson, Laura P Swiler, Christian R Trott, Stephen M Foiles, and Garritt J Tucker. Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials. *Journal of Computational Physics*, 285:316–330, 2015.

[105] Tomas Trnka, Igor Tvaroska, and Jaroslav Koca. Automated training of reaxff reactive force fields for energetics of enzymatic reactions. *Journal of chemical theory and computation*, 14(1):291–302, 2018.

[106] Christian R Trott, Damien Lebrun-Grandié, Daniel Arndt, Jan Ciesko, Vinh Dang, Nathan Ellingwood, Rahulkumar Gayatri, Evan Harvey, Daisy S Hollman, Dan Ibanez, et al. Kokkos 3: Programming model extensions for the exascale era. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):805–817, 2021.

[107] Adri CT van Duin, Jan MA Baas, and Bastiaan Van De Graaf. Delft molecular mechanics: a new approach to hydrocarbon force fields. inclusion of a geometry-dependent charge calculation. *Journal of the Chemical Society, Faraday Transactions*, 90(19):2881–2895, 1994.

[108] Adri CT Van Duin, Siddharth Dasgupta, Francois Lorant, and William A Goddard. Reaxff: a reactive force field for hydrocarbons. *The Journal of Physical Chemistry A*, 105(41):9396–9409, 2001.

[109] F Vazquez, Ester M Garzón, JA Martinez, and JJ Fernandez. The sparse matrix vector product on GPUs. In *Proceedings of the 2009 International Conference on Computational and Mathematical Methods in Science and Engineering*, volume 2, pages 1081–1092. Computational and Mathematical Methods in Science and Engineering Gijón, Spain, 2009.

[110] Francisco Vázquez, José-Jesús Fernández, and Ester M Garzón. A new approach for sparse matrix vector product on nvidia gpus. *Concurrency and Computation: Practice and Experience*, 23(8):815–826, 2011.

[111] Toon Verstraelen, PW Ayers, Veronique Van Speybroeck, and Michel Waroquier. ACKS2: Atom-condensed Kohn-Sham DFT approximated to second order. *The Journal of chemical physics*, 138(7):074108, 2013.

[112] Ross C Walker, Michael F Crowley, and David A Case. The implementation of a fast and accurate qm/mm potential method in amber. *Journal of computational chemistry*, 29(7):1019–1031, 2008.

[113] Junmei Wang, Romain M Wolf, James W Caldwell, Peter A Kollman, and David A Case. Development and testing of a general amber force field. *Journal of computational chemistry*, 25(9):1157–1174, 2004.

[114] Xinyan Wang, Jichen Li, Lan Yang, Feiyang Chen, Yingze Wang, Junhan Chang, Junmin Chen, Wei Feng, Linfeng Zhang, and Kuang Yu. Dmff: an open-source automatic differentiable platform for molecular force field development and molecular dynamics simulation. *Journal of Chemical Theory and Computation*, 19(17):5897–5909, 2023.

[115] Olaf Wiest, Kersey A Black, and KN Houk. Density functional theory isotope effects and activation energies for the cope and claisen rearrangements. *Journal of the American Chemical Society*, 116(22):10336–10337, 1994.

[116] Jun Zhang, Y Isaac Yang, Lijiang Yang, and Yi Qin Gao. Dynamics and kinetics study of "in-water" chemical reactions by enhanced sampling of reactive trajectories. *The Journal of Physical Chemistry B*, 119(45):14505–14514, 2015.

[117] Weiwei Zhang and Adri CT Van Duin. Improvement of the reaxff description for functionalized hydrocarbon/water weak interactions in the condensed phase. *The Journal of Physical Chemistry B*, 122(14):4083–4092, 2018.

[118] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

APPENDIX A

SUPPLEMENTAL DATA FOR REAXFF/AMBER—A FRAMEWORK FOR HYBRID REACTIVE/NONREACTIVE FORCE FIELD MOLECULAR DYNAMICS SIMULATIONS

A.1 Hardware and Software Setup

All computations have been performed on Laconia, a cluster with over 400 compute nodes at Michigan State University's High-Performance Computing Center. Each of the base compute nodes on Laconia has 28 cores, located on two fourteen core Intel Xeon E5-2680v4 Broadwell 2.4 GHz processors, and has 128 GB DDR3 2133 MHz ECC memory. Each core possesses a 64 KB L1 cache (32 KB instruction, 32 KB data), a 256 KB L2 cache. Between fourteen cores of a single "Broadwell" processor, a 35 MB L3 cache is shared. At the time of the experiments, the Laconia nodes ran CentOS version 7 distribution of GNU/linux for x86_64 architectures. The standalone ReaxFF software was built using the Intel Fortran compiler (IFORT) version 2019.3.199 with the -O3 flag.

A.2 QM Claisen Rearrangement Transition Configurations

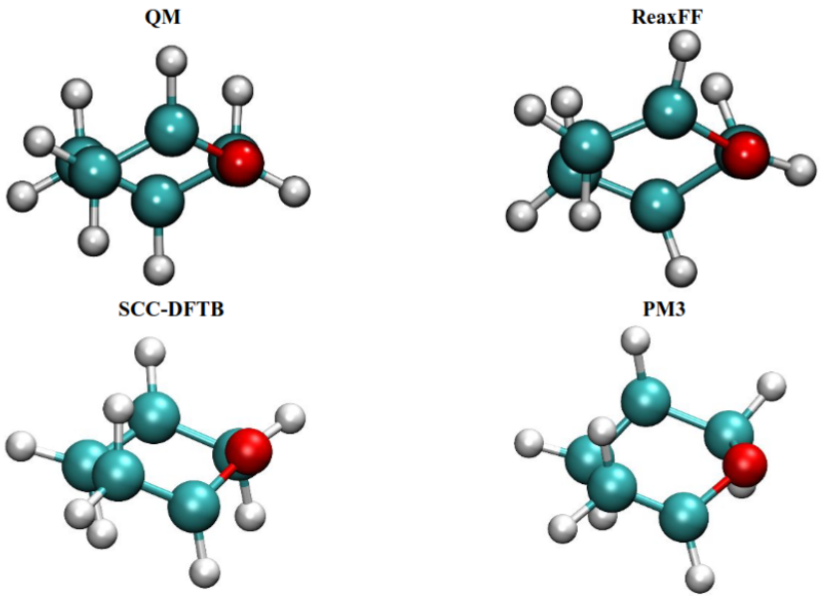


Figure A.1 Transition configuration of AVE during Claisen rearrangement for QM, ReaxFF, SCC-DFTB and PM3 simulations.

APPENDIX B

SUPPLEMENTAL DATA FOR JAX-REAXFF: A GRADIENT-BASED FRAMEWORK FOR FAST OPTIMIZATION OF REACTIVE FORCE FIELDS

B.1 Clustering Algorithms

Algorithm B.1 Clustering (G, I, R).

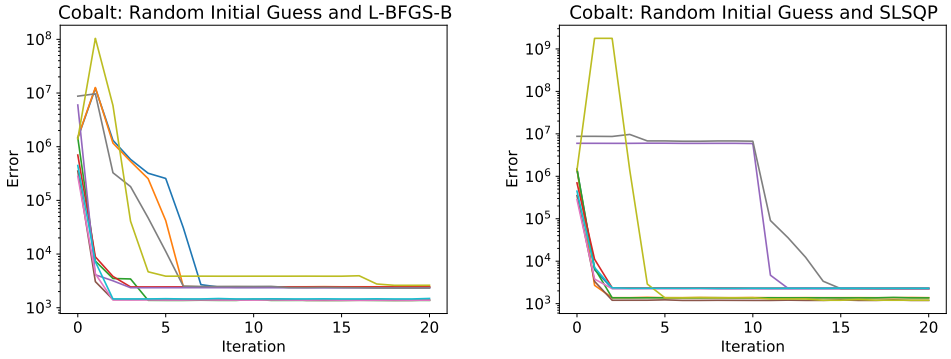
$C_{\text{best}} \leftarrow$ Keep track of the best so far
for $r = 1, 2, \ldots, R$ **do**
    $C_{\text{cur}} \leftarrow$ Initialize cluster centers by selecting random geometries
    for $i = 1, 2, \ldots, I$ **do**
        $C_{\text{prev}} \leftarrow C_{\text{cur}}$
        Shuffle G
        for each $g \in G$ **do**
            Assign g to c_i where Dist (g, c_i) is minimum
            Update the cluster centers
        end for
        if $C_{\text{cur}} == C_{\text{prev}}$ **then**
            break
        end if
    end for
    if Cost (C_{cur}) < Cost (C_{best}) **then**
        $C_{\text{best}} \leftarrow C_{\text{cur}}$
    end if
end for

Algorithm B.2 Modified k-Means for geometry clustering.

$k_{\max} \leftarrow$ Maximum number of clusters
$R \leftarrow$ Number of repetitions for the clustering algorithm
$I \leftarrow$ Number of iterations for the clustering algorithm
$C_{\text{selected}} \leftarrow$ Selected clustering of the geometries
for $k = 1, 2, \ldots, k_{\max}$ **do**
    Cost_k, $C_k \leftarrow$ Clustering (G, I, R) ▷ See Algorithm B.1
    if $|\text{Cost}_k - \text{Cost}_{k-1}| \,/\, \text{Cost}_{k-1} <$ tolerance or $k == k_{\max}$ **then**
        $C_{\text{selected}} \leftarrow C_k$
        break
    end if
end for

B.2 Convergence Plots for Training Tasks

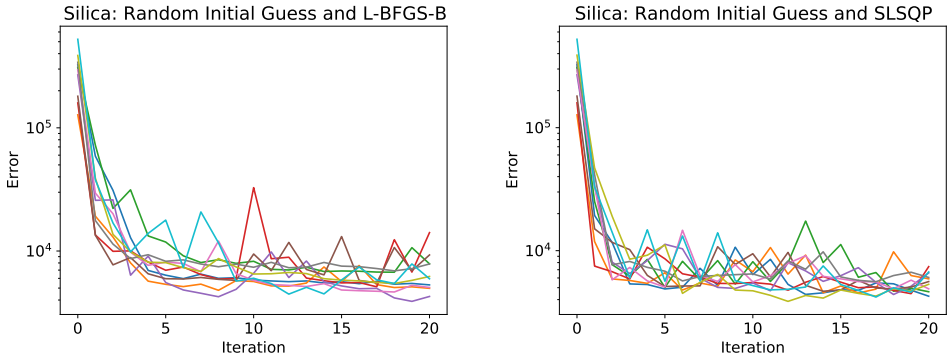
B.2.1 Cobalt



(a) L-BFGS-B (b) SLSQP

Figure B.1 Convergence of the local optimizers for the cobalt data set.

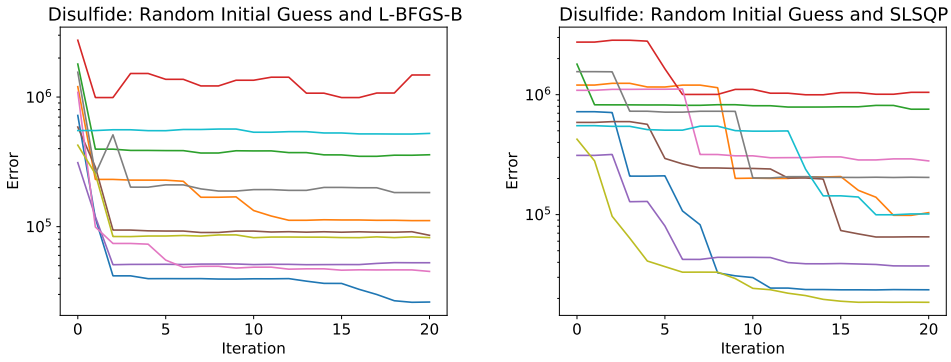
B.2.2 Silica



(a) L-BFGS-B (b) SLSQP

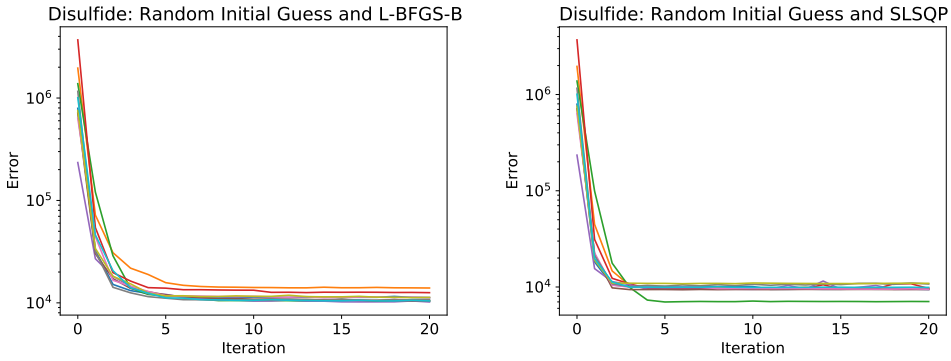
Figure B.2 Convergence of the local optimizers for the silica data set.

B.2.3 Disulfide



(a) L-BFGS-B (b) SLSQP

Figure B.3 Convergence of the local optimizers for the disulfide data set with single precision.



(a) L-BFGS-B (b) SLSQP

Figure B.4 Convergence of the local optimizers for the disulfide data set with double precision.

B.3 Hardware and Software Setup

All the CPU performance experiments reported were conducted on server with two Intel Xeon E5-2680 v4 @ 2.40GHz processors (2 processors each with 14 cores) and 128 GB 2133 MHz DDR4 RAM. The GPU experiments were conducted on server with a sole Intel Core i7-9700K CPU @ 3.60GHz processor, 16 GB 3000 MHz DDR4 RAM, and single 1080-TI PCI-e card (11 GB GDDR5X memory). For the baseline methods, the OGOLEM version 1.0 software with the PuReMD backend was used.

The proposed methods were implemented in Python 3.7 and utilized JAX version 0.1.76, NumPy version 1.16.4 and SciPy version 1.5.1.

APPENDIX C

SUPPLEMENTAL DATA FOR UNCERTAINTY QUANTIFICATION METHODS FOR MACHINE LEARNING POTENTIALS

C.1 Model Architecture

Name	Architecture
H	Linear(160)-CELU(0.1)-Linear(128)-CELU(0.1)-Linear(96)-CELU(0.1)-Linear(1)
C	Linear(144)-CELU(0.1)-Linear(112)-CELU(0.1)-Linear(96)-CELU(0.1)-Linear(1)
O, N, F	Linear(128)-CELU(0.1)-Linear(112)-CELU(0.1)-Linear(96)-CELU(0.1)-Linear(1)
Encoder for all types	Linear(16)-CELU(0.1)
Decoder for all types	CELU(0.1)-Linear(96)

Figure C.1 Model architectures for each available atom type and the autoencoder part (encoder and decoder).

As illustrated in Fig. C.1, the *CELU* [6] activation function is employed because it is twice differentiable, a critical property for using this model as a potential energy function. The term *Linear* refers to a linear transformation layer with learnable weights, with the output dimension provided as an argument. The model architecture determines the input dimension.