JAX-ReaxFF: A Gradient Based Framework for Extremely Fast Optimization of Reactive Force Fields

Mehmet Cagri Kaymak[1], Ali Rahnamoun[2], Kurt A. O'Hearn[1], Adri C. T. van Duin[3], Kenneth M. Merz, Jr.[2], and Hasan Metin Aktulga[1]

[1]*Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA*
[2]*Department of Chemistry, Michigan State University, 578 S. Shaw Lane, East Lansing, MI 48824, USA*
[3]*Department of Mechanical Engineering, The Pennsylvania State University, University Park, State College, Pennsylvania 16802, United States*

Abstract

Molecular dynamics (MD) simulations ease the study of the chemistry of interest. While classical models that governs the interaction of the atoms lack reactivity, the quantum mechanics based methods increase the computational cost drastically. ReaxFF fills the gap between these two ends of the spectrum by allowing bond breaking and dynamic charges. To achieve realistic simulations using ReaxFF, the model parameters need to be carefully tuned based on the training data created using more accurate but expensive methods. The current optimization methods focus on black-box optimization methods such as genetic algorithms (GAs), Monte-Carlo methods and covariance matrix adaptation evolutionary strategy (CMA-ES). Due to the stochastic behavior of these methods, the training requires hundreds of thousands of error evaluations for complex training tasks and each error evaluation usually involves energy minimization of the many molecules in the training data. In this work, we propose a novel approach which takes advantage of the modern tools developed for machine learning to improve the training efficiency of the force field development for ReaxFF. By calculating the gradients of the loss function using JAX library developed by Google, we are able to use well studied local optimization methods such as the limited Broyden–Fletcher–Goldfarb–Shanno (LBFGS) and Sequential Least Squares Programming (SLSQP) method. To further decrease the training time, we skip the energy minimization of the molecules during the local optimization since the parameters of the model are not likely to change drastically during a local search. JAX allows us to easily parallelize the error evolution by compiling the code for CPUs, GPUs or TPUs. With

the help of the modern accelerators and the gradient information, we are able to decrease the training time from days to minutes.

1 Introduction

Simulating atoms and their interactions are essential to study chemical and physical properties of many interesting systems. Simulations based on quantum mechanics (QM) allow the geometries and energies to be predicted accurately by solving the Schrödinger equation. However, the computational complexity and cost of the QM based methods make them only viable for simulating small molecules for short periods of time. Molecular dynamics (MD) simulations allow a set of atoms to be simulated by using simple approximations based on the Newtonian physics. In this approach, atoms are treated as unit particles and their interactions are governed by a model called force field (FF). The simplification greatly reduces the computational cost because the effect of the electrons are captured by the force field model but these models rely on tedious parameterization of various atomic interactions corresponding to bonds, valence angles, torsion, van der Waals, Coulomb interactions etc. based on carefully selected quantum-chemical reference data. With the help of these approximations and careful training, the classical MD methods have been proved to be successful simulating systems up to billions of atoms.

One of the core limitation of classical MD methods is lack of reactivity since they typically rely on fixed interatomic connectivity and partial charges. They usually model chemical bond using simple parameterized harmonic oscillators and it renders them useless for reactive systems. ReaxFF is developed by van Duin et al. to fill the gap between QM based methods and classical MD approaches [31]. It allows bonds to form and break throughout the simulation and dynamically assign charges to the atoms using the electronegativity equalization method (EEM) [21].

ReaxFF divides the total potential energy of the system into various parts similar to the nonreactive force fields as it is shown by (1). The model requires Cartesian coordinates of the atoms and the model parameters as input to calculate various potentials and respective forces. The analytical derivative of the total potential energy with respect to the coordinates gives the atomic forces which are fundamental to the MD simulation. Therefore, the analytical forces are carefully implemented as a part of various software packages for ReaxFF such as the standalone ReaxFF, PuReMD [1, 2, 17], GULP [11] and LAMMPS [25].

$$\begin{aligned} E_{\text{system}} &= E_{\text{bond}} + E_{\text{over}} + E_{\text{under}} + E_{\text{val}} + E_{\text{pen}} + E_{\text{tors}} + \\ &\quad E_{\text{conj}} + E_{\text{vdWaals}} + E_{\text{Coulomb}} + E_{\text{specific}} \end{aligned} \tag{1}$$

The ReaxFF parameters are grouped by the number of atoms involved in the interaction such as single-body, two-body, three-body and four-body parameters besides the global parameters. Based on the distances and angles between the atoms and corresponding model parameters, the interaction lists are created.

Type	Training Item	Target	Description
Charge	ID1 1	0.5	Charge for atom 1
Energy	ID1 - ID2/2 - ID3/3	50 kcal/mol	Energy difference
	ID1	-150 kcal/mol	
	ID3/2 - ID1/3	30 kcal/mol	
Geometry	ID1 1 2	1.25 Å	Distance between atom 1 and 2
	ID2 1 2 3	120°	Valence angle between atom 1, 2 and 3
	ID3 1 2 3 4	170°	Torsion angle between atom 1, 2, 3 and 4
Force	ID1 1	0.5 0.5 0.5	Forces on atom 1
	ID2	1.0	RMSG

Table 1: Training item types. Identifiers (Ex. ID1) are used as a reference to the molecules.

For every interaction, the respective parameters are selected by the group and the types of the atoms in the interaction to calculate E_{system}. As the number of the atom types increases, the training the force field gets harder due to the increased number of parameters. Even though there are already tuned parameter sets for various systems, they might require further tuning if an existing parameter set is not performing well for chemistry of interest. In some cases, the model needs to be trained from scratch which highly increases the complexity of the process. As it is originally designed for the standalone ReaxFF, the training procedure requires three different inputs: a file containing various atom clusters related to the system of interest, a file containing the training data which uses the properties of these atom clusters such as geometric properties (angles and distances), forces on atoms, root mean square of the forces (RMSG), relative potential energies or charges with their target values and lastly selected model parameters to tune to minimize the error on the training data. While calculating model predictions, some molecules might require energy minimization before the final calculation to prevent overfitting and also to tune the parameters based on more likely states of the atom clusters since the lower energy states are more likely to be observed. The error function combines different types of training items in the following format

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

where m is the model, x_i is the model prediction, y_i is the ground truth and σ_i^{-1} is the weight of the corresponding training item. The supported item types are shown in Table 1. An energy based item could be formed using a linear relationship of up to 5 molecules using their identifiers. It helps the model to capture the relative potential energies of the molecules rather than the exact potential energies since the MD is driven by the former ones. Also there can be charge and geometry based items. For geometry based items, the molecules are expected to be energy minimized since it is essential to observe the effect of the force field on the geometry. For the other types, energy minimization is optional but usually preferred for the reasons mentioned previously.

As it is further discussed in Section 2, the current force field optimization

methods for ReaxFF employ gradient free black-box optimization methods such as Genetic Algorithms (GA) and Evolutionary Algorithms (EA). These methods could enable a global search with the cost of more error evaluations since they assume the error function is a black box and only the final output can be observed. The proposed method takes a peek inside the block box and learns more about the local surface to increase the training efficiency. By calculating the derivative of the error on the training set with respect to the selected force-field parameters using JAX [3], well tested local optimization methods such as the Limited Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm [33] and Sequential Least Squares Programming (SLSQP) [16] optimizer, both are implemented in the SciPy library [32], could be employed. The local derivative information might cause the optimizers to get stuck in a local minima but when combined with a multi-start approach, we can greatly improve the training efficiency and reduce the time spent on training a force field. By leveraging modern tools, the analytical derivative of the Error(m) with respect to the model m can be taken without explicitly implementing it and the overall training time could be greatly reduced by up to three of magnitude without sacrificing much accuracy as it is demonstrated in Section 4.

Beyond speeding up force field optimization, it should be noted that JAX-ReaxFF provides the ideal sandbox environment for domain scientists, as they can go beyond parameter optimization and start experimenting with the functional forms of the ReaxFF interactions, or add/remove interactions as desired. Through automatic differentiation, parameter optimization for the new set of functional forms can be performed without any additional effort by the domain scientist. Also, since evaluating the gradient with respect to atom positions gives forces, scientists are freed from the burden of coding the lengthy and bug-prone force calculation parts, and they can easily test the macro-scale properties predicted by the modified set of functional forms as a further validation test before production-scale simulations.

2 Related Work

The earliest method used to train a force field for ReaxFF is sequential one-parameter parabolic interpolation method (SOPPI) [7]. The algorithm uses a well known one parameter-at-a-time approach where the consecutive single parameter searches are done until a certain convergence criteria is met. The algorithm is preferred for its simplicity but as the total number of parameters increases, the number of one-parameter optimization steps needed for convergence increases drastically since only a small portion of the search space is explored in each round. Also the performance of this method is very dependent on the initial guess and the order of the parameters to be optimized in each step. Due to the drawbacks of this approach and need for a more efficient, global and automated algorithms, various global optimization methods such as genetic algorithms (GAs) [6, 15, 19], simulated annealing (SA) [13, 14], evolutionary algorithms (EAs) [30], particle swarm optimization (PSO) [10] and

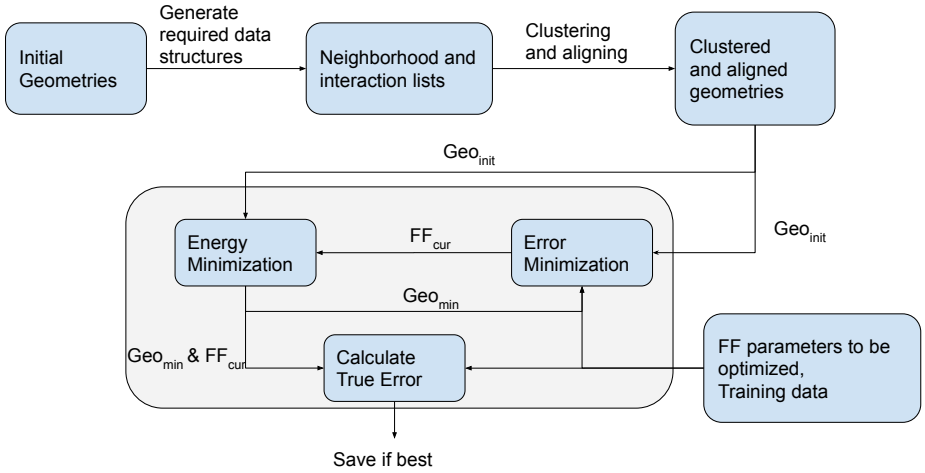


Figure 1: Proposed method

machine learning based search methods [5, 12, 24] have been investigated in the context of force field optimization. Due to the sheer amount of different methods, a thorough explanation and evaluation of these methods are out of scope of this paper. Several studies have been done to compare these methods [28, 29]. When there is no gradient information available, these methods are proven to be successful for many black box optimization problems. However, due to the nature of the global search algorithms, they require high number of error evaluations and depending on the size of the problem, they could be very costly compared to the methods using gradients to help the search.

After the emergence of the highly optimized tools for machine learning to calculate gradients of complex functions automatically, a method called Intelligent-ReaxFF has been proposed to take advantage of these tools to train a force field for ReaxFF [12]. They use the TensorFlow library to calculate the gradients and optimize a force field. 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 potentials to the reference data. The energy differences between different geometries or charges cannot be used to train the force field which limits the usability. More general framework has been developed to provide end-to-end differentiable MD simulations [27]. Unlike this work, the main focus is to provide a unifying framework to study different potential functions including machine learning potential.

3 Proposed Method

The ReaxFF energy expression is implemented in Python using JAX library which is developed by Google [3]. JAX offers hassle free gradient calculation

with automatic differentiation (auto-diff). After providing the energy calculation, we can easily calculate the gradients of the training error with respect to the parameters to be optimized using auto-diff functionality of the framework. The same auto-diff functionality is also used to calculate the forces. Besides that, JAX uses Accelerated Linear Algebra (XLA), a domain specific compiler for linear algebra, to accelerate the computation. The Python code using JAX can be run on CPUs, GPUs or TPUs without any changes on the code which is another big advantage. The Python code gets compiled only once using XLA if the input array shapes do not change.

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 to run multiple small scale simulations introduces a lot of overhead. The optimizations such as iterative preconditioned and sparse solvers to calculate the atomic charges, fast neighbor list generation algorithms, distributed computation etc. could potentially increase the run-time and complicate the implementation when the number of atoms per geometry is low. Even though vanilla Python code tends to be slower than efficient Fortran or C code, when the just-in-compiled XLA support, auto-diff functionality and benefits of targeting small geometries are considered, it is beneficial to implement the ReaxFF model in Python using JAX.

3.1 ReaxFF Model Implementation

The design goal is that instead of running one big geometry in parallel, many small geometries could be run parallel on many-core architectures. JAX offers vectorization (*vmap*) and parallelization (*pmap*) to take advantage of the underlying architecture fully. To better utilize processing units with high level of Single Instruction Multiple Data (SIMD) parallelism, JAX offers *vmap* so that multiple small systems can be merged into a batch to yield high device utilization. While the target parallelism for *vmap* is SIMD, *pmap* targets Multiple Instruction Multiple Data (MIMD) parallelism. The current implementation takes advantage of *vmap* to further accelerate the energy and gradient calculations. To be able to use vectorization, the interaction lists are precalculated and aligned.

$$BO_{ij}^{'} = BO_{ij}^{\sigma} + BO_{ij}^{\pi} + BO_{ij}^{\pi\pi} = \exp\left[p_{bo1}\left(\frac{r_{ij}}{r_o^{\sigma}}\right)^{p_{bo2}}\right] + \exp\left[p_{bo3}\left(\frac{r_{ij}}{r_o^{\pi}}\right)^{p_{bo4}}\right] + \exp\left[p_{bo5}\left(\frac{r_{ij}}{r_o^{\pi\pi}}\right)^{p_{bo6}}\right] \tag{3}$$

In regular ReaxFF implementations, the interaction and neighborhood lists are created based on the initial atom positions and the fixed force field parameters. As it can be seen in Figure 2, the bonded interactions depend on the corrected bond order term. Initially, If the distance between two atoms is less

than the given cutoff, typically 5 A, the bond order (BO) term is calculated based on Equation 3. In the given equation, r_{ij} is the distance between the atoms and p_{bo1}, p_{bo2}, p_{bo3}, p_{bo4}, p_{bo5}, p_{bo6}, r_o^{σ}, r_o^{π}, and $r_o^{\pi\pi}$ are the model parameters selected based on the atom types. If the calculated term is greater than a potential based cutoff, the interaction is added to the interaction list and the bond order term is corrected based on the neighborhood of the atoms forming the bond. 3-body and 4-body interaction lists are built using the corrected BO term. To limit the number of interactions, another cutoff parameter is used based on the bonds forming multi-bond interactions while creating 3-body and 4-body interaction lists.

Differently from the regular ReaxFF MD simulations, the force field is dynamic during the optimization. Based on the updated FF parameters, the interactions change because of the potential based cutoffs even if the atom positions stay constant. Also when a given molecular structure in the training data requires energy minimization, the positions change as well. Because JAX requires recompilation if the input array shapes change and also recreating interaction lists is costly, we have decided to create the interaction lists once before the optimization starts and use masks to ignore the unwanted elements throughout the optimization and/or energy minimization. To achieve that purpose, for every unique atom type pair, the maximum possible distance where a given pair can have a valid bonded interaction is found. If some BO related parameters need to be optimized, the values which maximize the BO term is selected from the given ranges. Then by simply scanning the possible distance values, the maximum possible distance is determined as the cutoff for the interactions between that pair. Similar logic is applied for the other types of interactions. For the geometries that require energy minimization, the maximum calculated distance is extended by a fixed value to create room for atom positions to change.

It is assumed that there is a non-bonded interaction between every atom in the system since the non-bonded interaction cutoff, typically 10 A, is a lot larger the bonded interaction cutoff and the molecules used for training are quite small. Therefore, non-bonded interactions form an $N * N$ matrix. If the system has periodic boundary condition and the box dimensions are a, b and c Angstrom (A) and cutoff is 10 A, then the size of the tensor for non-bonded interactions will become $N * N * \lceil \frac{10}{a} \rceil * \lceil \frac{10}{b} \rceil * \lceil \frac{10}{c} \rceil$.

Once the arrays for the interactions are created, their sizes do not change. The unwanted elements in the interaction arrays are masked. Although this approach wastes some of the computational power, it does not require reneighboring, recreation of the interaction lists and recompilation. It helps us to separate the interaction list generation part from the force field optimization to simplify the process. The interaction lists are generated on the CPU using multiprocessing.

To calculate the potential energy, a similar approach to the standalone ReaxFF code is followed with an only exception of the charge calculation. The Electronegativity-equalization method (EEM) based charge model assigns charges to the atoms after solving a linear equation of $Ax = b$ where A is an NxN matrix, the details can be found in [21]. Typically, the linear system is

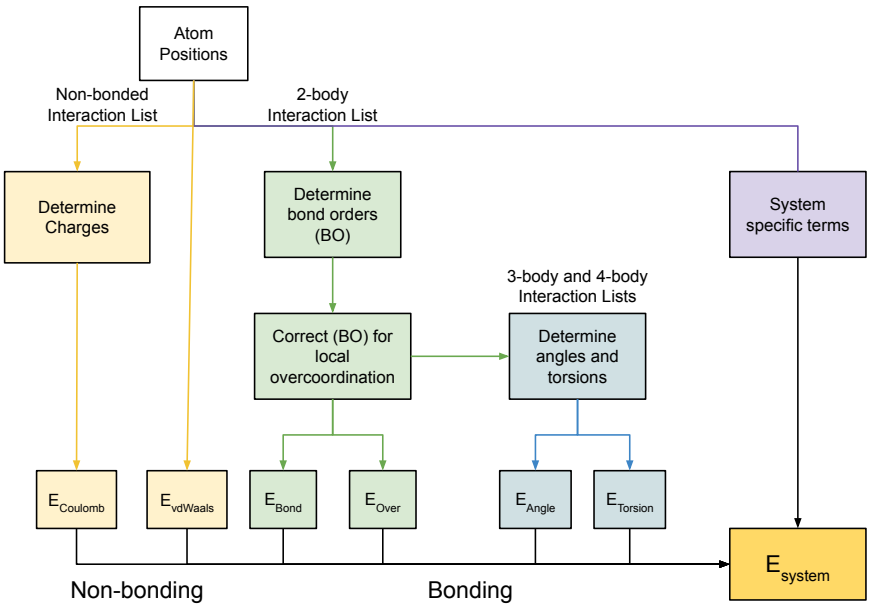


Figure 2: ReaxFF Model

solved using a pre-conditioned and iterative solving scheme. Since, N, the number of atoms, is low for the training items, we use a direct method based on LU factorization which is easier to implement and differentiate.

3.2 Data Preprocessing

Since one geometry is not enough to use all of the available cores, multiple geometries are clustered together, aligned and fed into the processing unit using the vectorization (vmap) functionality of the framework. For each cluster, the interaction lists of the geometries are aligned by padding and the newly added elements are masked later during the energy and gradient calculations so that they do not affect the output. To cluster the geometries for easy vectorization, a modified version of K-Means algorithm [20] is used. The distance formula for geometry x and cluster center y with size s_y is shown below

$$\begin{aligned}\text{Dist}(x,y) = s_y(c_1\max(n_{\{2,x\}}, n_{\{2,y\}}) + c_2\max(n_{\{3,x\}}, n_{\{3,y\}}) + \\ c_3\max(n_{\{4,x\}}, n_{\{4,y\}}) + c_4\max(n_{\{5,x\}}n_{\{1,x\}}^2, n_{\{5,x\}}n_{\{1,y\}}^2))\end{aligned} \tag{4}$$

where n_1, n_2, n_3, n_4 and n_5 are the number of atoms, the number of 2-body interactions, the number of 3-body interactions, the number of 4-body interactions and the number of periodic boxes within long range cutoff respectively. c_1 through c_4 are the constants. Each term in the equation is a indicator of computational cost of a kernel. The constants are currently chosen to be 1, but their relative weights could be learned empirically to better represent the

computational costs in a given training set. Since the cost of each cluster is determined by the largest instruction list size within the cluster, the max value between the the current cluster center y and geometry x is chosen.

After initializing k cluster centers randomly, each geometry is assigned to these clusters based on the unique distance metric where the distance is an indicator of the change in computational load after assigning geometry x to cluster center y as it is shown in (4). The new center for each cluster is determined by the largest interaction lists within the cluster, since they determine the amount of padding needed for alignment of the geometries in that cluster. After centers are updated, a new iteration is performed where each geometry is reassigned to the closest cluster. Unlike the original K-Means algorithm, the order of the geometries affects the result, so they are shuffled after each iteration to randomize the algorithm. The process continues until the cluster centers do not change anymore. Also to increase the performance of the algorithm for a given number of clusters, the clustering algorithm is run multiple times starting from different random initial cluster centers and the one where the total wasted computation is minimal is chosen as the final clustering of the geometries. Although the algorithm does not guarantee optimality, empirical results are satisfactory. Finally, another set of clusters are created after removing the geometries that do not require energy minimization. This cluster set is used to vectorize the energy minimization procedure.

Algorithm 1 Clustering Algorithm

```
C_best ← Keep track of the best so far
for r = 1, 2, ... R do
    C_cur ← Initialize the cluster centers by selecting a random geometry as the center for each cluster
    for i = 1, 2, ... I do
        C_prev ← C_cur
        Shuffle G
        for each g ∈ G do
            Assign g to c_i where Dist(g, c_i) is minimum
            Update the cluster centers
        end for
        if C_cur == C_prev then
            Break
        end if
    end for
    if Cost(C_cur) < Cost(C_best) then
        C_best ← C_cur
    end if
end for
```

The total compilation time of JAX increases exponentially with the number of clusters. Also if the wasted computation does not change drastically, small

number of clusters is more favorable for GPUs because of easy vectorization. For these reasons, the number of clusters is selected automatically based on algorithm 2. If the computational gain is not significant, smaller number of clusters is preferred.

Algorithm 2 Clustering Algorithm 2

$k_{max} \leftarrow$ Maximum number of clusters
$R \leftarrow$ Number of repetitions for the clustering algorithm
$I \leftarrow$ Number of iterations for the clustering algorithm
$C_{selected} \leftarrow$ Selected clustering of the geometries
for $k = 1, 2, \ldots k_{max}$ **do**
    $Cost_k$, $C_k \leftarrow Clustering(G, k, I, R)$
    if $|Cost_k - Cost_{k-1}|/Cost_{k-1} < 0.10$ or $k == k_{max}$ **then**
        $C_{selected} \leftarrow C_k$
        Break
    end if
end for

3.3 Force Field Training

After the final clusters are formed, the training is done using the logic described in Figure 1. Training is done using gradient based local optimizers with multi-start. For energy minimization, a simple gradient descent algorithm is used because of its simplicity. Previously mentioned vectorization method is used for both energy minimization and force field optimization to allow calculations to be parallelized on a GPU.

3.3.1 Gradient Based Local Optimization

For each iteration of the training loop, two different local optimizations are performed, one to locally minimize the energy by updating the atom positions using steepest descent and the other one to minimize the fitness error on the energy minimized geometries by updating the selected force field parameters using various local optimization methods such as L-BFGS-B [33, 4] and SLSQP [16]. The true error is calculated after the energy minimization step, if there are any geometries that require energy minimization. The local optimization step uses the single point charge, energy and force calculations using the minimized geometries from the previous step as a surrogate model to accelerate the training. The error on the surrogate gets closer to the true error as the parameters converge because the changes in the parameters become minimal. Besides the computational cost of energy minimizing the structures, it is also more error prone to calculate derivatives in that way using the auto-diff functionality because of the complex functional form of the ReaxFF. One disadvantage of separating the energy minimization from the local optimization is that the fitness score for the geometry items will be be ignored by the local optimization

since the atom positions will not change. It 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 (PES), the discrepancy could be minimized. As it is demonstrated in the later sections, the surrogate approach works well in practice for a variety of training tasks which have geometry based items.

Algorithm 3 Gradient Based Local Optimization

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

4 Evaluation

4.1 Datasets

To evaluate the performance of the proposed method, the following training sets are evaluated and the results are compared against CMA-ES, MCFF and GA, which are studied in [28]. The selected training sets are relatively different in terms of the number of parameters and the training item types. Also while the geometries in the Cobalt and Silica training sets mostly require energy minimization, the geometries in the Disulfide one are mostly single step. As shown in Table 1, the proposed method does not handle cell parameter based training items. During the training, these items are ignored. It only affects the Silica dataset which have 5 of them. When all of these are considered, they form a well rounded test-bench for the novel method we have developed.

Training Data	N_{par}	N_{geo}	N_{minim}	C	G	F	P	E
Cobalt [18]	12	146	130	0	0	0	0	144
Silica [8]	67	302	221	5	26	0	6	265
Disulfide [23]	87	231	10	0	255	4401	0	219

Table 2: Datasets. N_{par} is the number of parameters to optimize, N_{geo} is the number of geometries and N_{minim} is the number of geometries to be energy minimized. C, G, F, P and E are the number of charge based training items, geometry based items, force based items, cell parameter based items and energy based items, respectively.

4.2 Runtime and Training Evaluation

Usually when a new force field needs to be developed for a certain domain or task, multiple passes over the training data is needed. Based on the performance of the force field, the training data might be changed. If there is a pre-existing force field, it might just require a simple fine-tuning for the target domain. If a force field is being trained from scratch, multiple updates could be done to the training data. For both of these scenarios, the training efficiency of the optimizers play an important role and need to be analyzed.

The initial guess is another important factor which could change the results drastically. It is especially important for gradient based method since it cannot move through the space freely since we need to follow the direction of the gradients. To show the capabilities of the proposed approach better, different initialization methods are used. For each experiment, the initialization methods from the Shchygol et al [28] are repeated. The initial guess types are divided into three categorises including random, educated and literature based initial guesses. For the random initial guesses, the initial parameters are sampled from a uniform distribution based on the given parameter ranges. To produce educated guesses, prior information comes from the previous related force fields is used as it is described in further details in [28]. For the literature based initial guesses, the force fields developed previously using the same training data are used. To give more reliable results, each initialization method is repeated ten times. For the educated and literature based initial guesses, 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 $[\frac{-1k}{10}, \frac{1k}{10}]$ where k is the length of the given range for parameter p. For the random initial guesses, uniform sampling is done ten times to produce the guesses.

For the training, 2 different gradient based optimization algorithms, L-BFGS-B and SLSQP, are used and the results are compared against black-box optimizers including CMA-ES, GA and Monte Carlo based force field optimizer (MCFF). The settings for the black-box optimizers are given in [28]. For both L-BFGS-B and SLSQP, the maximum number iterations is set to 100. 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 is set to 20 for Algorithm 3 where the local error minimization and the energy minimization is done iteratively. Therefore, for all of the experiments, the full energy minimization and the true error calculation is done 20 times. The local error minimization only uses single step calculations.

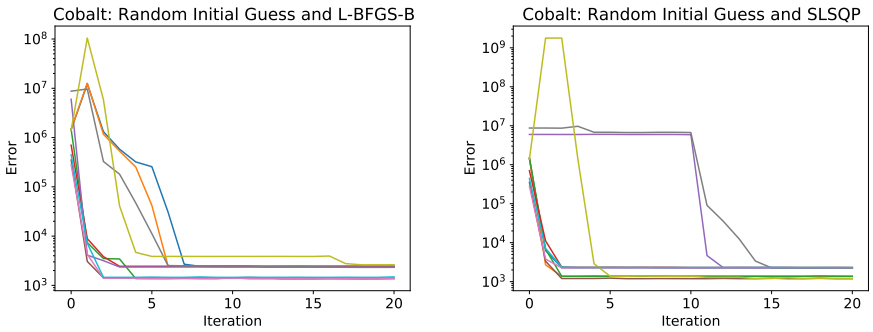
For the hardware, we have used single 1080-TI GPU and Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz (28 cores) CPU. To approximate the execution times for the black-box methods, we calculated the per iteration time on the mentioned CPU and multiplied it by the number of iterations provided for each method in [28]. This approximation is a lower bound for CMA-ES and MCFF since they have lower level of parallelism unlike genetic algorithm where each evaluation is independent from each other.

4.2.1 Cobalt

As shown in Table 5, the proposed method converges faster than the black box approaches while producing similar error on the Cobalt training data which only have energy based training items. Although close to 90% of the geometries require energy minimization, the error does not fluctuate as seen in Figure 3. It shows that the surrogate error is close to the true error for this dataset. 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 error surface which cause high gradients, the optimizer fails to escape (Figure 6b). When the progress stops, small noise is added to stimulate the progress.

Method	Initial Guess	Best Score	Median Score	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: Cobalt training results.



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

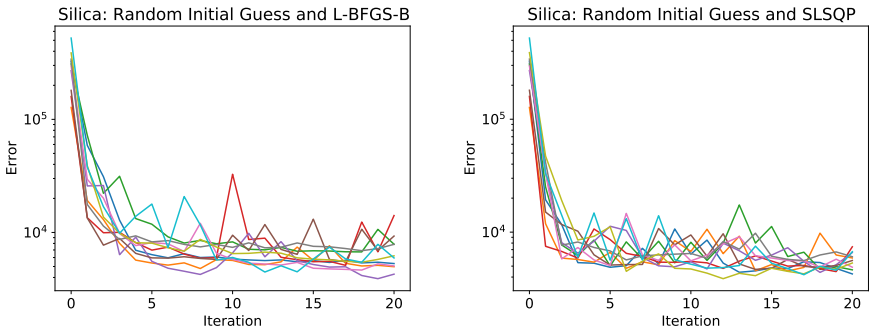
Figure 3: Convergence of the local optimizers for the Cobalt dataset

4.2.2 Silica

The silica training set includes energy, charge and geometry based items. 73% of the geometries require energy minimization. Although the single point evaluation based surrogate model ignores the geometry based items, the proposed method is able to minimize the error comparable to the black box method while taking fraction of the execution time. For the true error which is calculated after energy minimization, the geometry based items are calculated. For this dataset, the cell based items are fully ignored for both the true error and surrogate error calculations. Unlike the Cobalt case, the error fluctuates more between iterations possibly because of unstable geometries and geometry based items.

Method	Initial Guess	Best Score	Median Score	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 4: Silica training results.



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

Figure 4: Convergence of the local optimizers for the Silica dataset

4.2.3 Disulfide

The disulfide training data is drastically different from the previous one since it uses force matching to fit the ReaxFF model to the data. As it is shown below, the forces are calculated by taking the derivative of the potential energy with respect to the positions and the local optimizers require the derivative of the forces with respect to the force field parameters.

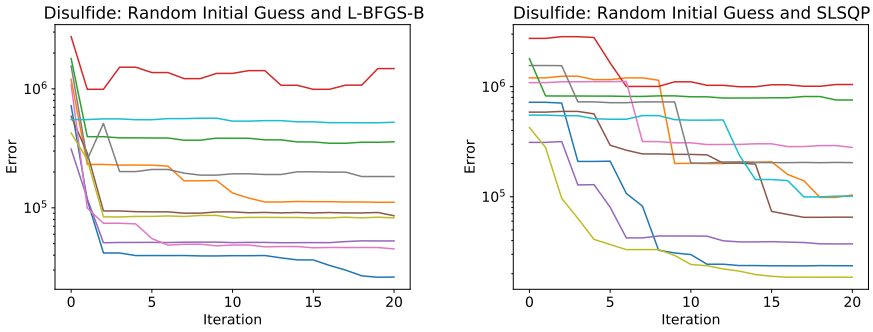
$$F_x = \frac{\partial E_p}{\partial x} \tag{5}$$

$$\frac{\partial (F_x - F_t)^2}{\partial p} = \frac{\partial (\frac{\partial E_p}{\partial x} - F_t)^2}{\partial p} \tag{6}$$

where F_x is the 3-dimensional force vector on atom x, F_t is the target force vector and p is the model parameters to be optimized. $\frac{\partial (F_t - F_x)^2}{\partial p}$ gives the gradients for the force based items required to minimize the error. The auto-differentiation functionality of the JAX library handles this easily without providing any derivative calculations by hand. However, the final gradients for the parameters from Equation 3 result in high values, $\sim 10^{17}$, while the other gradients are much lower. The high gradients stop the local optimizers from doing any progress as seen in Figure 5. Therefore, we excluded these parameters and fixed their values to the literature ones. Among 87 parameters, 18 parameters are removed and the optimization is performed again with the remaining 69 parameters. As shown in Figure 6, the results are improved drastically. The high gradients might be related to the bond order issues described in [9] but it requires further investigation.

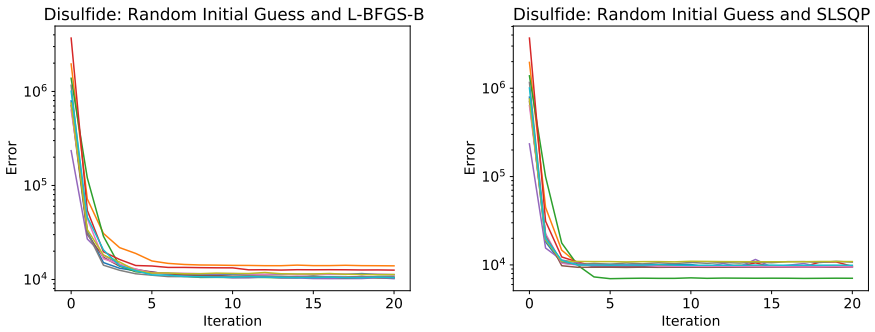
Method	Initial Guess	Best Score	Median Score	Avg. # Single Step Eval.	# True Eval.	Avg. CPU Exec. Time (min)	Avg. GPU Exec. Time (min)
L-BFGS-B*	rand	10198	10920	1660	20	9.7	0.9
	edu	10313	10631	1600			
	lit	10438	10803	1503			
SLSQP*	rand	6986	9488	1187	20	8.9	0.8
	edu	9306	9635	1234			
	lit	10304	10494	1901			
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 5: Disulfide training results. *The results are from the modified version of the training.



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

Figure 5: Convergence of the local optimizers for the Silica dataset before the modification



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

Figure 6: Convergence of the local optimizers for the Silica dataset after the modification

4.3 Force Field Evaluation

To validate the fitted Si/O and Co ReaxFF parameters, the optimized force fields were evaluated by various molecular dynamics (MD) simulations. MD simulations in this work are performed using Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) that is a molecular dynamics program from Sandia National Laboratories [25]. A relatively short time step of 0.5 fs was used in all the simulations. This is the recommended setting for ReaxFF simulations of systems that don't include light atoms like Hydrogen. All NVT ensemble (constant number of atoms, volume and temperature) simulations were performed using Nosé-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 Nosé-Hoover thermostat to control the temperature with a temperature damping parameter of 100 fs and Nosé-Hoover barostat to control the pressure with a temperature damping parameter of 1000 fs.

4.3.1 Molecular Dynamics Simulations of Pure Silica Structure

To evaluate the quality of the optimized force field for the Si/O parameters, the amorphous silica structure introduced in the Fogarty et al [8] was reconstructed. The amorphous silica system included 2000 SiO_2 molecules with an initial density of 2.2 g/cm^3 (Figure 7).

The amorphous silica system was energy-minimized to eliminate initial bad contacts. The system was then annealed twice between 300K and 4000K. The first annealing loop was performed using NVT ensemble with heating and cooling rate of 37 K/ps. The second annealing loop was performed in NPT ensemble between 300K-4000K using Nose–Hoover thermostat and barostat 1.01325 bar pressure. Similar to the NVT annealing, the heating and cooling rate was

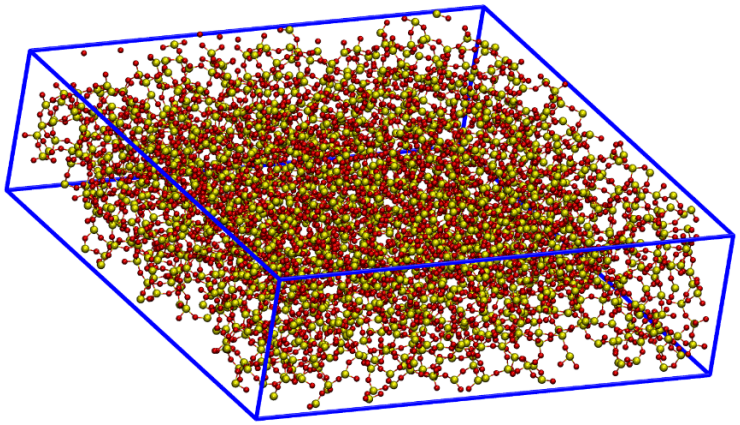


Figure 7: The amorphous silica structure including 2000 SiO_2 molecules and total of 6000 atoms. Silicon atoms are shown with yellow color and Oxygen atoms are shown with red color.

37 K/ps. Finally, the silica system was equilibrated in NPT ensemble using T=300K and P=1.01325 bar for additional 200 ps as the production run. These calculations were performed using our fitted force field and two previous ReaxFF force fields introduced for such systems. The properties of the final configuration of these silica structures are compared in Table 6. The radial distributions of the final configuration of silica structure equilibrated using our fitted force field for Si-O, O-O and Si-Si are shown in Figure 8. These results show acceptable force field training quality for silica structure using our proposed method for force field optimization.

Property	2010 FF [8]	2019 FF [28]	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 6: Silica properties calculated using three different force fields. The experimental value reported for silica density is 2.2 g/cm^3 [22]

4.3.2 Molecular Dynamics Simulations of Pure Cobalt Structure

The LAMMPS molecular dynamics code was employed to investigate crystal lattice constant correlation with cohesive energy in crystals of fcc Cobalt. The lattice constant was changed from 3Å to 5Å and the associated lattice cohesive energies were recorded (Figure 9). The results of the fitted force field were

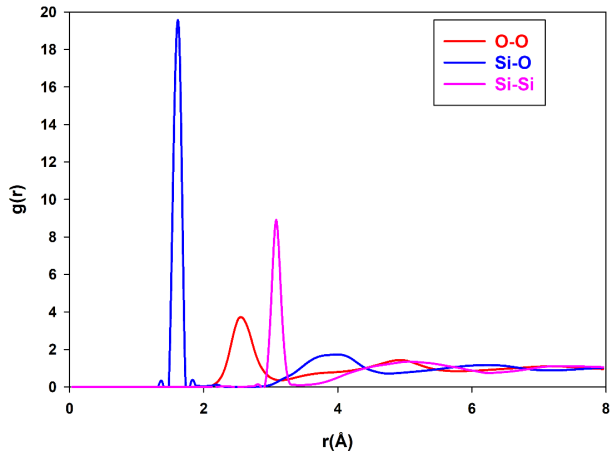


Figure 8: Radial distribution function of silicon-oxygen, oxygen-oxygen, and silicon-silicon for silica structure at the end of annealing and equilibration.

compared to two previously trained ReaxFF force fields for Cobalt [18, 28] and embedded atom method (EAM) force field [26].

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 5x5x5 ideal fcc Cobalt unit cells was generated for annealing simulations using NPT ensemble between 1000K-3000K. After the NPT equilibration of the pure Cobalt crystal at 1000K, the system was subjected to NPT ensemble annealing between 1000K-3000K 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 Figure 10. Three ReaxFF force fields showed similar dynamic evolution behavior for the pure Cobalt structure while the EAM force field showed a different dynamic evolution (Figure 11).

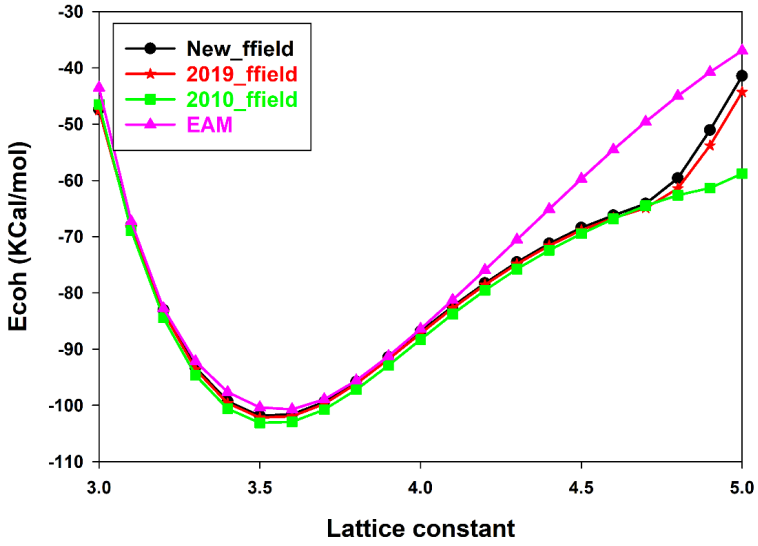


Figure 9: Variations in pure Cobalt single fcc crystal cohesive energy by variations of the lattice constant.

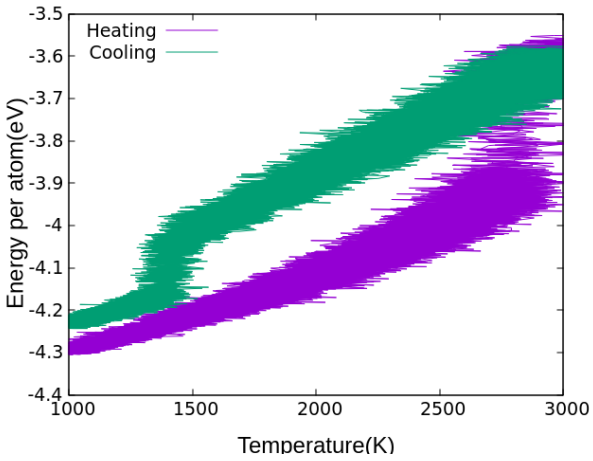


Figure 10: Annealing loop of a 5x5x5 fcc Cobalt crystal including 500 atoms using fitted ReaxFF force field with heating and cooling rate of 10 K/ps.

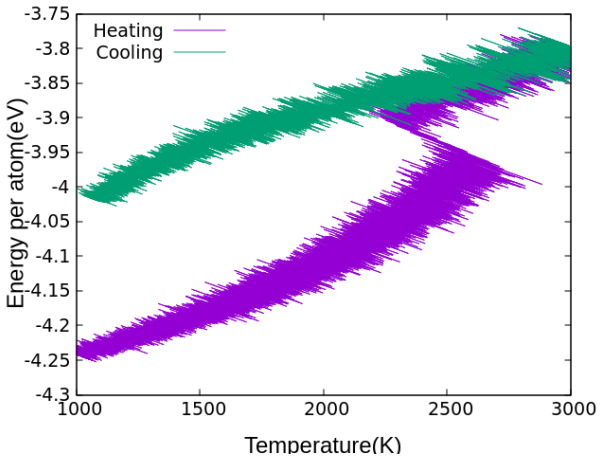


Figure 11: Annealing loop of a 5x5x5 fcc Cobalt crystal including 500 atoms using EAM force field with heating and cooling rate of 10 K/ps.

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 (Figure 12).

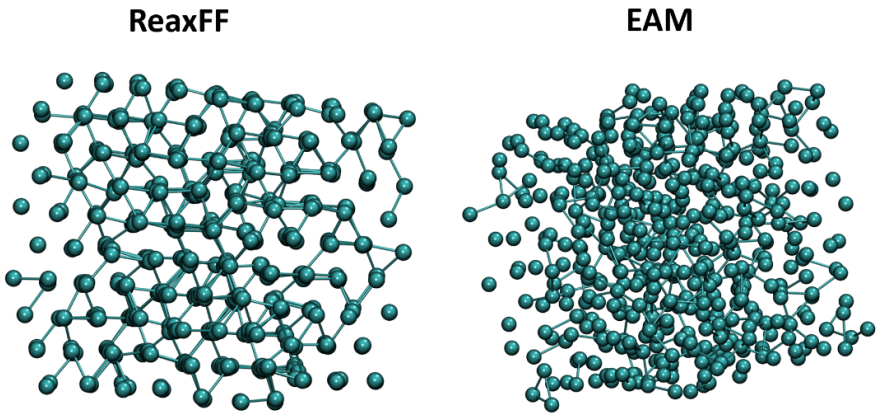


Figure 12: Final configurations of pure Co fcc crystals after annealing loop with 1000K-3000K temperature range.

4.3.3 Molecular dynamics simulations of molecules with Sulfur bonds

To test the validity of the force field containing Sulfur parameters updated using our proposed training method, we performed minimum energy structure search for single molecules with different restraints. The results of the fitted force

field were compared to two previously trained ReaxFF force fields[4, 7]. The restraints are applied on S-S bond of dimethyl disulfide (DMDS), S-C bond of dimethyl thioether (DMTE), H-S-H angle of Hydrogen sulfide (H2S) and H-S-S-H torsion angle of Hydrogen disulfide (H2S2). These potential energy graphs are shown in Figure 13.

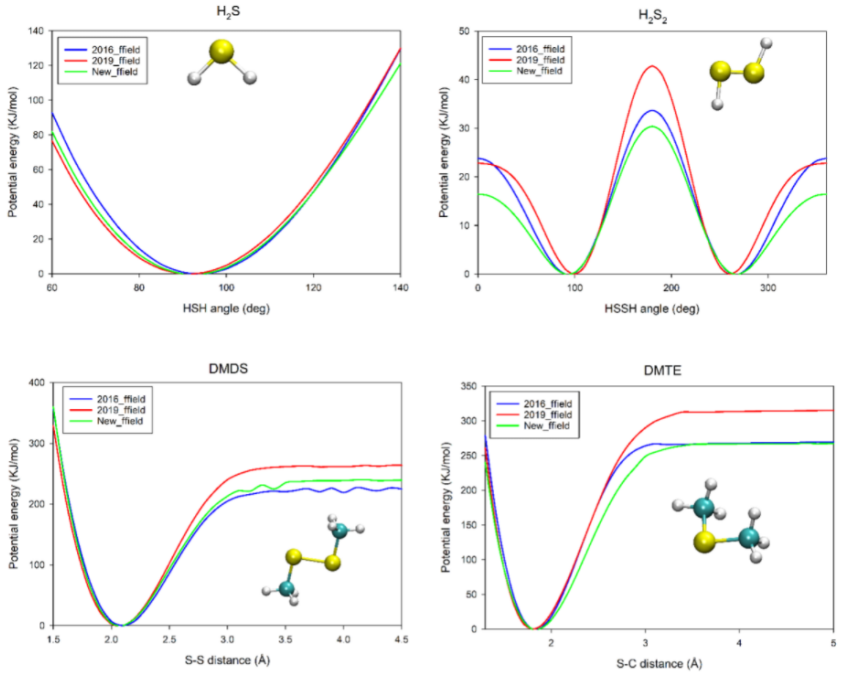


Figure 13: Potential energy graphs of energy minimized molecules including Sulfur bonds with different restraints, calculated with the updated force field and previously trained force fields.

5 Conclusion

Based on the experiments, we have seen that even if the starting force field is bad, gradient based local optimizers are able to increase the fitness of the force field drastically. As it is described in Algorithm 3, 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. This allows users to generate the force fields in minutes. Auto-diff functionality enables the study of the new functional forms for the various parts of the ReaxFF model without explicitly implementing the force calculations. JAX-ReaxFF provides a sandbox environment for domain scientists to tinker with the ReaxFF model and also enables accelerated force field optimization.

References

[1] Hasan Metin Aktulga et al. "Parallel reactive molecular dynamics: Numerical methods and algorithmic techniques". In: *Parallel Computing* 38.4-5 (2012), pp. 245–259.

[2] Hasan Metin Aktulga et al. "Reactive molecular dynamics: Numerical methods and algorithmic techniques". In: *SIAM Journal on Scientific Computing* 34.1 (2012), pp. C1–C23.

[3] James Bradbury et al. "JAX: composable transformations of Python+ NumPy programs, 2018". In: *URL http://github. com/google/jax* (2020), p. 18.

[4] Richard H Byrd et al. "A limited memory algorithm for bound constrained optimization". In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[5] Chaitanya M Daksha et al. "Automated ReaxFF parametrization using machine learning". In: *Computational Materials Science* 187 (), p. 110107.

[6] Mark Dittner et al. "Efficient global optimization of reactive force-field parameters". In: *Journal of computational chemistry* 36.20 (2015), pp. 1550–1561.

[7] 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". In: *Journal of the Chemical Society, Faraday Transactions* 90.19 (1994), pp. 2881–2895.

[8] Joseph C Fogarty et al. "A reactive molecular dynamics simulation of the silica-water interface". In: *The Journal of chemical physics* 132.17 (2010), p. 174704.

[9] David Furman and David J Wales. "Transforming the accuracy and numerical stability of ReaxFF reactive force fields". In: *The journal of physical chemistry letters* 10.22 (2019), pp. 7215–7223.

[10] David Furman et al. "Enhanced particle swarm optimization algorithm: Efficient training of reaxff reactive force fields". In: *Journal of chemical theory and computation* 14.6 (2018), pp. 3100–3112.

[11] Julian D Gale, Paolo Raiteri, and Adri CT van Duin. "A reactive force field for aqueous-calcium carbonate systems". In: *Physical Chemistry Chemical Physics* 13.37 (2011), pp. 16666–16679.

[12] Feng Guo et al. "Intelligent-ReaxFF: Evaluating the reactive force field parameters with machine learning". In: *Computational Materials Science* 172 (2020), p. 109393.

[13] Pierre O Hubin et al. "Parameterization of the ReaxFF reactive force field for a proline-catalyzed aldol reaction". In: *Journal of Computational Chemistry* 37.29 (2016), pp. 2564–2572.

[14] Eldhose Iype et al. “Parameterization of a reactive force field using a Monte Carlo algorithm”. In: *Journal of computational chemistry* 34.13 (2013), pp. 1143–1154.

[15] 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”. In: *Journal of Chemical Theory and Computation* 10.4 (2014), pp. 1426–1439.

[16] Dieter Kraft et al. “A software package for sequential quadratic programming”. In: (1988).

[17] Sudhir B Kylasa, Hasan Metin Aktulga, and Ananth Y Grama. “PuReMD-GPU: A reactive molecular dynamics simulation package for GPUs”. In: *Journal of Computational Physics* 272 (2014), pp. 343–359.

[18] Matthew R LaBrosse, J Karl Johnson, and Adri CT van Duin. “Development of a transferable reactive force field for cobalt”. In: *The Journal of Physical Chemistry A* 114.18 (2010), pp. 5855–5861.

[19] Henrik R Larsson, Adri CT van Duin, and Bernd Hartke. “Global optimization of parameters in the reactive force field ReaxFF for SiOH”. In: *Journal of computational chemistry* 34.25 (2013), pp. 2178–2189.

[20] Stuart Lloyd. “Least squares quantization in PCM”. In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[21] Wilfried J Mortier, Swapan K Ghosh, and S Shankar. “Electronegativity-equalization method for the calculation of atomic charges in molecules”. In: *Journal of the American Chemical Society* 108.15 (1986), pp. 4315–4320.

[22] RL Mozzi and n BE Warren. “The structure of vitreous silica”. In: *Journal of Applied Crystallography* 2.4 (1969), pp. 164–172.

[23] Julian Müller and Bernd Hartke. “ReaxFF reactive force field for disulfide mechanochemistry, fitted to multireference ab initio data”. In: *Journal of chemical theory and computation* 12.8 (2016), pp. 3913–3925.

[24] Hiroya Nakata and Shandan Bai. “Development of a new parameter optimization scheme for a reactive force field based on a machine learning approach”. In: *Journal of computational chemistry* 40.23 (2019), pp. 2000–2012.

[25] Steve Plimpton. “Fast parallel algorithms for short-range molecular dynamics”. In: *Journal of computational physics* 117.1 (1995), pp. 1–19.

[26] GP Purja Pun and Y Mishin. “Embedded-atom potential for hcp and fcc cobalt”. In: *Physical Review B* 86.13 (2012), p. 134116.

[27] Samuel S Schoenholz and Ekin D Cubuk. “Jax, md: End-to-end differentiable, hardware accelerated, molecular dynamics in pure python”. In: *arXiv preprint arXiv:1912.04232* (2019).

[28] Ganna Shchygol et al. “ReaxFF Parameter Optimization with Monte-Carlo and Evolutionary Algorithms: Guidelines and Insights”. In: *Journal of Chemical Theory and Computation* 15.12 (2019), pp. 6799–6812.

[29] Ganna Shchygol et al. “Systematic comparison of Monte Carlo Annealing and Covariance Matrix Adaptation for the optimization of ReaxFF parameters”. In: *ChemRxiv* (2018).

[30] Tomas Trnka, Igor Tvaroska, and Jaroslav Koca. “Automated training of ReaxFF reactive force fields for Energetics of Enzymatic reactions”. In: *Journal of chemical theory and computation* 14.1 (2018), pp. 291–302.

[31] Adri CT Van Duin et al. “ReaxFF: a reactive force field for hydrocarbons”. In: *The Journal of Physical Chemistry A* 105.41 (2001), pp. 9396–9409.

[32] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[33] Ciyou Zhu et al. “Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization”. In: *ACM Transactions on Mathematical Software (TOMS)* 23.4 (1997), pp. 550–560.