Proxima: Accelerating the Integration of Machine Learning in Atomistic Simulations

Yuliana Zamora
University of Chicago
Chicago, Illinois, USA
yzamora@uchicago.edu

Logan Ward
Argonne National Laboratory
Lemont, Illinois, USA
lward@anl.gov

Ganesh Sivaraman
Argonne National Laboratory
Lemont, Illinois, USA
gsivaraman@anl.gov

Ian Foster
University of Chicago & Argonne National Laboratory
Chicago, Illinois, USA
foster@uchicago.edu

Henry Hoffmann
University of Chicago
Chicago, Illinois, USA
hankhoffmann@cs.uchicago.edu

ABSTRACT

Atomistic-scale simulations are prominent scientific applications that require the repetitive execution of a computationally expensive routine to calculate a system's potential energy. Prior work shows that these expensive routines can be replaced with a machine-learned *surrogate* approximation to accelerate the simulation at the expense of the overall accuracy. The exact balance of speed and accuracy depends on the specific configuration of the surrogate-modeling workflow and the science itself, and prior work leaves it up to the scientist to find a configuration that delivers the required accuracy for their science problem. Unfortunately, due to the underlying system dynamics, it is rare that a single surrogate configuration presents an optimal accuracy/latency trade-off for the entire simulation. In practice, scientists must choose conservative configurations so that accuracy is always acceptable, forgoing possible acceleration. As an alternative, we propose Proxima, a systematic and automated method for dynamically tuning a surrogate-modeling configuration in response to real-time feedback from the ongoing simulation. Proxima estimates the uncertainty of applying a surrogate approximation in each step of an iterative simulation. Using this information, the specific surrogate configuration can be adjusted dynamically to ensure maximum speedup while sustaining a required accuracy metric. We evaluate Proxima using a Monte Carlo sampling application and find that Proxima respects a wide range of user-defined accuracy goals while achieving speedups of 1.02–5.5× relative to a standard implementation with no surrogate.

CCS CONCEPTS

• **Computing methodologies** → **Uncertainty quantification**; **Computational control theory**; *Online learning settings.*

KEYWORDS

Modeling and Simulation, Machine Learning, Control Theory

ICS '21, June 14–17, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8335-6/21/06.
https://doi.org/10.1145/3447818.3460370

ACM Reference Format:
Yuliana Zamora, Logan Ward, Ganesh Sivaraman, Ian Foster, and Henry Hoffmann. 2021. Proxima: Accelerating the Integration of Machine Learning in Atomistic Simulations. In *2021 International Conference on Supercomputing (ICS '21), June 14–17, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3447818.3460370

1 INTRODUCTION

Scientific computing applications, such as continuum fluid dynamics (CFD), lattice quantum chromodynamics (QCD), and atomistic simulations account for a large fraction of the supercomputing cycles used at national laboratories and other supercomputer facilities [1]. These applications are often dominated by the repetitive execution of a few high-cost functions [6, 22]. In the past, speedups in these domains have relied on advances in hardware architecture and numerical-algorithm development [18, 21, 35]. However, recent advances in machine learning have enabled another promising acceleration technique: replacing expensive functions with machine-learned *surrogates* [4, 15]. Because the learned surrogate is an approximation of the expensive *target function*, the use of learned surrogates introduces opportunities for trade-offs between computation latency and accuracy (or error).

Pioneering work on surrogate methods has proved the value of integrating machine learning into scientific applications [8, 9, 38, 50]. However, that work has focused primarily on the construction of the surrogate models themselves, rather than on how to integrate these approximations into larger simulations. In particular, it employed ad hoc, heuristic integration strategies, such as featurizing the expensive function and then using the surrogate only when a function call's features are in the range seen when the surrogate model was trained [7, 8]—an approach that demonstrates the potential value of surrogate usage, but has significant practical drawbacks.

The speed and accuracy of a simulation that uses a surrogate model depends on factors such as (1) the form of the surrogate model, (2) when to use the surrogate model, (3) how much training data to use to generate the surrogate model, and (4) how often to retrain said model while the simulation runs. Prior methodologies for using surrogate models have typically fixed these parameters or left the choice of values to the user—approaches that prevent optimal choices or impose significant burdens on users in terms of

profiling to find acceptable surrogate configurations. Furthermore, prior methodologies use the same configuration values during an entire run (and from run to run), leading to suboptimal outcomes when the simulation's properties change as it evolves.

We describe in this paper a new approach to this problem that both simplifies and optimizes the use of machine learning in scientific simulation by creating a dynamic surrogate configuration engine. In so doing, we remove the need for a previously trained model, precomputed training set, or user-specified retraining schedule—and permit surrogate-based simulations to adapt dynamically to changing simulation behaviors. To do so, we introduce Proxima, an application-agnostic Python library developed to incorporate surrogate models within a scientific simulation, allowing the user to specify a desired maximum mean absolute error to be met. Proxima achieves this acceptable requested error by continually monitoring the simulation execution, dynamically adapting the surrogate configuration parameters and determining when to retrain the surrogate model at run-time.

Specifically, we focus on the decision engine which sets the criteria for when to use the surrogate model or revert to the original, high-cost, high-accuracy target function. We implement a domain agnostic decision engine based on control theory. Our control-theoretic decision engine ties in how often the model is retrained while determining the training set size and thus establishing a relationship between performance and accuracy. Unlike prior work that sets key parameters on surrogate usage before the simulation is run, when using Proxima, these values are determined automatically by Proxima's run-time system. Thus, instead of requiring extensive testing and specific parameters for each application run, Proxima dynamically tunes the parameters for each test case based on run-time feedback without prior training. Additionally, unlike prior approaches, Proxima no longer requires the model to be retrained at every step, significantly reducing overall run-time.

We showcase Proxima by applying it to an example application involving Monte Carlo (MC) simulation of a methane molecule (CH4) across a wide range of temperatures. Methane is the smallest member of the hydrocarbon family. Interactions in methane are dominated by many-body weak dispersion interactions, for which surrogate models must provide first-principles accuracy [47]. Finding training datasets and configuration parameters that can deliver this accuracy with good performance is a tedious process. We compare Proxima to a system with no surrogate and to a surrogate-based approach that uses prior methodologies based on profiling to find the best fixed surrogate configuration parameters for the entire simulation. We demonstrate that Proxima, when run for temperatures in the range {100, 200, ... 1000}K, obtains speedups from 1.02× to 5.52× over the non-surrogate version, with a harmonic mean speedup of 1.64×, while respecting the error bounds in all cases. Testing with error bounds that stress Proxima verifies that Proxima works across a wide range of these user-defined error bounds. Finally, we double-check the accuracy results by comparing approaches along a secondary physical property, the radius-of-gyration (ROG), that relies on not just average accuracy, but the accuracy of each individual simulation step. We find that Proxima achieves a mean absolute error of less than 0.00126Å. In contrast, the fixed parameter approach of prior work yields results beyond the acceptable error bounds across most of the temperatures tested.

In summary, our contributions are:

- Proposing dynamic tuning of machine-learned surrogate usage in scientific computing.
- Designing the algorithms that can perform this tuning while respecting error bounds.
- Developing a library to make surrogate integration a lightweight addition to existing simulation software.
- Demonstrating the value of tuning surrogate usage parameters to optimize performance with accuracy guarantees.
- Open source release of the code.[1]

2 BACKGROUND

In this work, we use atomistic simulations as an example scientific application domain with a history of combining physics and machine learned surrogates. Here, we explain the methods behind the physics and machine learning components and how they are combined in prior work. We also provide a brief background on control theory, as it forms the basis of our proposed method for tuning surrogate configurations in running science simulations.

2.1 Atomistic Modeling

Atomistic modeling computes interactions between atoms to capture the complex behavior of materials and molecules [10]. With atomistic modeling, scientists can study material properties and defects difficult to observe experimentally due to both spatial and temporal constraints. To do so, the simulation evaluates the energy of a system and the forces acting on each atom in many different configurations. Dominant atomistic modeling methodologies, like density functional theory (DFT) and Hartree-Fock (HF), that rely on computing interactions from first principles (i.e., quantum mechanics), can take many minutes, hours, or days to complete such energy computations [5, 42]. They also quickly become limited in terms of the size of the system that can be simulated, as computational requirements scale with the number of electrons cubed or worse [13]. Much time and resources are invested in these calculations with the goal of understanding the dynamic behavior of metals, semiconductors, thin films, ceramics, and biological materials [10] and significant new applications are possible if the length and timescale of these models can be expanded.

2.2 Atomistic Machine Learning

Machine learning based atomistic simulations gives access to times and length scales not accessible to first-principle simulation, while maintaining first-principle accuracy. In particular, supervised learning techniques can be built to compute the potential energy of a system in milliseconds (10^5 times speedup over some quantum mechanical methods) and with computational costs that scale linearly with problem size. The key innovation which has enabled the use of machine learning is how to represent the structure of an atomic system in a form amenable to machine learning [3, 23]. For our work, we rely on the large body of prior work on representations and how to use them in conjunction with modern machine learning approaches [11, 20, 25, 29, 41, 48, 49].

[1] https://github.com/globus-labs/proxima/tree/proxima_control

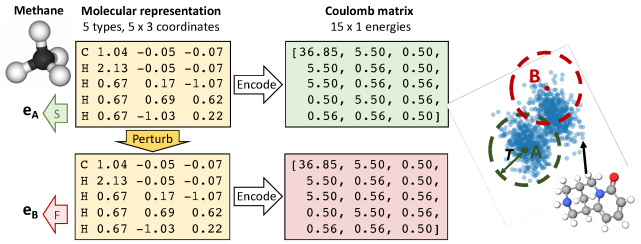


Figure 1: Illustrates the fundamental ideas that this work leverages in atomistic modeling. A molecule state is represented by coordinates, say A. These can be perturbed to yield a different state, B. A state can be encoded in a Coulumb matrix, which when treated as a point in a multi-dimensional space allows for distance computations. A is within a distance T of a previously evaluated state, B is not.

In this work, we use two common methods for atom representation: Coulomb Matrix and Smooth Overlap of Atomic Positions (SOAP). Both representations are designed specifically for modeling atomic systems. For example, they are invariant to translating and rotating the coordinate system and permuting the order in which atoms are number. These invariances, in conjunction with being designed to capture that atomic interactions are dominated by local, many-body interactions, make the Coulomb Matrix and SOAP suitable for building surrogates. We chose the two methods to give a tradeoff between speed and accuracy for the machine learning methods themselves.

SOAP [3] is a common atom representation used with training models for the energy and forces acting on atoms that uses a similarity measure between atomic neighbor environments. We use the SOAP implementation in the Dscribe library [25] as the featurization and training data for our model. Then, in a similar manner to the non-linear kernel ridge regression (KRR) method used by Botu et al. [7], we train a simple Bayesian ridge regression model to predict the potential energy.

In addition, we also use the Coulomb Matrix as a quicker-to-compute alternative to SOAP. The Coulomb Matrix representation of the atom, proposed by Rupp et al. [38], describes an atom based on the atomic numbers and pairwise distances between atoms. As illustrated in Figure 1, we use the Coulomb matrix as a similarity metric between different molecular geometries when quantifying how similar a new geometry is from those used to train our model. Because the similarity check step of Proxima occurs at every time step, regardless of whether or not we need to invoke the SOAP-based surrogate model, the small computational cost of the Coulumb matrix is beneficial.

2.3 Configuring Surrogate Usage

Previous work has established there are benefits in switching between using surrogate models and the high-cost, target function during a simulation, which presents an obvious tradeoff between accuracy and speed [8, 30, 32]. Many implementations of these "hybrid" physics + machine learning applications require a decision about whether a new set of function inputs can be effectively treated with the surrogate model rather than the target function. The metrics used to inform these "domain of applicability" judgements are numerous and each require setting a *threshold* value based on empirical evidence (i.e., profiling the simulation with a surrogate across a range of thresholds) [36, 39]. As we demonstrate, setting an applicability threshold is complicated by the ideal threshold being dependent on the boundary conditions for a simulation and, potentially, even the current state of the simulation.

In the work reported here, we use the distance of a set of function inputs from all entries used to train a model as our domain of applicability metric. The target function is then used for a function call whenever the distance from that call's inputs to all training data exceeds a specified threshold. Our use of a threshold metric is based on work from Botu et al. [7], who observed that the error of a surrogate model for DFT calculations scales quadratically with distance from the training set. We use this result as a justification for setting a single threshold to identify which predictions should be feasible. We elected for this method over alternative approaches, such as measuring the variance of an ensemble of models [36], due to the low cost of computing nearest neighbors.

As hybrid physics+ML applications evolve, surrogate configuration may even go beyond deciding when a surrogate should be used. For example, the amount of computational resources devoted to the machine learning and physics components of the multi-scale partitioning strategy of Caccin et al. [12] would be an example of a parameter that strongly controls application performance. The high computational cost of retraining machine learning models also introduces opportunities for accelerating applications by deferring training until sufficient data are collected—introducing more parameters to be tuned. Further, models such as sparse KRR [43] provide easy tradeoffs between model accuracy and inference speed. The additional opportunities for performance optimization further motivate the need to automatically tune performance parameters.

2.4 Control Theory

Prior work statically configured surrogate usage: scientists determined a single threshold for acceptable surrogate use and then used that threshold for the life of the program. Our proposal is that dynamically tuning the threshold results in better outcomes. Key to our approach is using a control theoretic design to dynamically tune surrogate usage.

Control theory is a discipline for managing dynamic systems [19]. At a high level, a controller is given a target metric and then measures dynamic feedback from the system. The feedback is used (in combination with a model of the system to control) to determine how to adjust parameters such that the desired behavior is achieved. As computers are dynamic systems, several researchers have proposed methods for building controllers that manage computer systems [14, 16, 24, 33, 44], with a particular emphasis on controlling accuracy and performance tradeoffs [26–28]. One major challenge of applying control theory to computer systems is that control theory was developed for continuous linear systems, and computers are discrete, non-linear systems.

Control systems thus appear to be a natural match for our problem. We want to adjust surrogate usage such that a user-defined error bound is met. To apply this technique, we need to do three

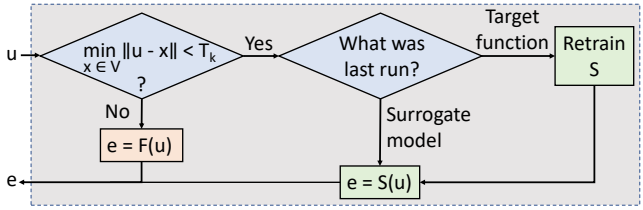


Figure 2: Logical flow of the Proxima surrogate modeling process. At upper left, an input value u is received and checked relative to a distance threshold from recently evaluated values. Ultimately either the target function F or the surrogate model are used to compute the return value e. A key difference between Proxima and prior work is that the threshold used to determine whether the surrogate should be used (T_k in the upper left) is determined dynamically; i.e., this threshold changes with time *k*.

things: (1) find an appropriate feedback metric that can be measured at runtime and relates to a scientifically meaningful error metric, (2) find appropriate configuration parameters that can be dynamically tuned to change error and latency tradeoffs, and (3) account for the non-linearities in the relationship between surrogate usage and error. We explain in detail how we address these three issues in Section 3.

3 PROXIMA

The basic idea of surrogate modeling is to replace an expensive target function with a faster machine-learned *surrogate model*. The strategy is to speed up the overall simulation by sacrificing accuracy in a systematic manner. Thus the surrogate must provide an acceptable level of accuracy, but take less time to train and execute than the target function. The crux of the problem is knowing when to use the surrogate model, when to add data to the training set, and how often to retrain said model. In prior work, these decisions are made explicitly by the scientist—who is responsible for setting appropriate surrogate configuration parameters—and require laborious profiling to find an acceptable accuracy/performance tradeoff. In contrast, Proxima automatically and dynamically configures these values to meet accuracy constraints with good performance, eliminating the scientists' burden of manually tuning these parameters.

In this section, we describe Proxima, independently of any specific scientific application. In the subsequent section, we describe how it is applied to atomistic Monte Carlo, replacing a Hartree–Fock-energy prediction target function with a Proxima-managed surrogate.

To begin, in every application, Proxima has a target function **F**. We need to process a series of requests for the value of that function for different arguments, each of which we can be evaluated either by running **F** on the supplied argument, or alternatively by running a surrogate model **S**. The surrogate model is dynamically trained using results from previous evaluations of **F**.

There are thus two key decision points in the system: 1) For each call to the target function, whether to use **F** or **S** to evaluate it; and 2) for each new evaluation of **F**, whether or not to retrain **S**, and which past results to use for that retraining. We organize our solutions to these problems as an **Executor**, which decides whether to execute **F** or **S** based on the distance logic shown in Figure 2, and a **Controller**, which updates the configurable parameters of the **Executor** during execution. One parameter could be a distance threshold, T_k, that controls how similar inputs must be to the training set of **S** before the Executor chooses to run **S**. Prior work (see Section 2.3) uses a static distance threshold to determine when to use the surrogate. However, we observe that the relationship between surrogate accuracy and distance changes as the simulation executes.

3.1 Executor: Surrogate Selection Logic

We now explain the logic behind the **Executor**. We use the following notation. Let $\tilde{\mu}$ be a user-supplied target error expressed as mean absolute error (MAE) on a specific value in the scientific simulation; and V be a vector of results collected so far, ordered by time of the corresponding request, and each of the form (x, y, y', d), where x is a valid argument to **F**, $y = \mathbf{F}(x)$, $y' = \mathbf{S}(x)$, and d is a distance, computed as described below. Also, let N be the number of recent observations in V used for computing MAEs.

Let V_k be V after k observations have been made and T_k be the current distance threshold. Now consider a new request for a function evaluation on a value u. We compute the distance from u to the nearest observation in the N most recent observations in V, denoted as $V_k[\mathrm{N}]$:

$$d = \min_{x \in V_k[\mathrm{N}]} \|x - u\| .$$

If $d < T_k$, then we retrain the surrogate model **S** if the target function was run for the preceding request, and return the value $\mathbf{S}(u)$ and continue to the next request. If $d \geq T_k$, then we run both the target function and the surrogate model, and add $(u,\ \mathbf{F}(u),\ \mathbf{S}(u),\ d)$ to V_k, producing V_{k+1}. Here we assume that the surrogate model is significantly cheaper than the target function so running both presents very little overhead.

3.2 Controller: Setting Distance Threshold

We now discuss how we use the **Controller** to perform dynamic online adjustments of the Executor. Here we are computing a threshold T_k for the current time k.

We formulate this task as a control problem. Specifically, we want to control the simulation error to meet the user-specified error bound $\tilde{\mu}$. We want to use the surrogate as much as possible (to maximize speedup) while maintaining an error at or below the bound. At any time k, we can compute the achieved error as μ_{k+1}, the MAE of $V_{k+1}[\mathrm{N}]$; i.e., the average of the absolute differences between the y and y' values in the N most recent elements of V_{k+1}. In this case, *controlling* the error means that we want $\mu_k - \tilde{\mu} \leq 0$. We can control the error by setting the threshold T_k. Intuitively, if we set the threshold extremely high, the surrogate is always used, while if we set it extremely low, the target function is always used. Our goal is to formulate a controller that dynamically sets the threshold to bring the error to the user-specified bound.

To formulate the controller, we need to know the relationship between error and threshold. A simple linear model characterizes this relationship as:

$$\mu = \alpha \cdot T, \tag{1}$$

where α is simply a coefficient that represents how much a change in threshold affects the change in error. With this model we can formulate a basic control system that manages the error by dynamically tuning the threshold:

$$T_{k+1} = T_k - \frac{1}{\alpha} \cdot (\mu_k - \tilde{\mu}) \tag{2}$$

This controller is simple and low-overhead, requiring just a handful of floating point computations to compute a new threshold. The drawback is that the linear relationship, α from Equation 1, rarely holds in practice because the relationship between the error and the threshold changes as the simulation evolves. For example, in our case study later in this paper we find that at higher temperatures, the same threshold will produce a higher error than at lower temperatures.

One approach to address this issue would be to build a non-linear model for α. In some sense, however, this approach would simply replace the laborious profiling prior work requires to set the threshold with a different laborious profiling task to build an appropriate non-linear model that adapts α over time. Therefore, we take a different approach and approximate the true version of α by continually estimating it with a time varying linear model. Specifically, we compute α_{k+1} via regression analysis of the formula $d = \alpha_{k+1}|y - y'| + \beta$, for $(x_i, y_i, y'_i, d_i) \in V_{k+1}$[N]. We then compute this dynamic version of α_{k+1} and use it in Equation 2:

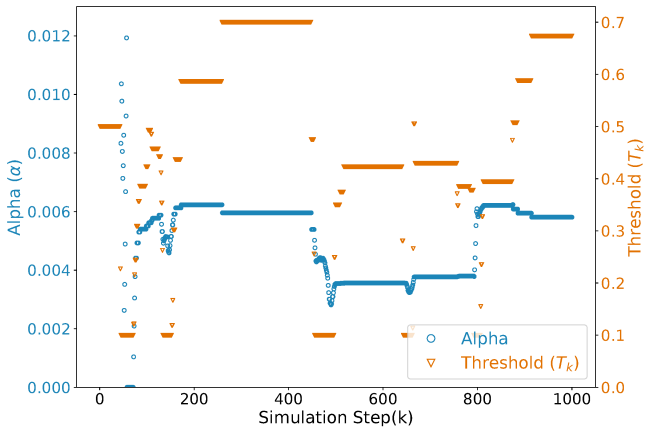
$$T_{k+1} = T_k - \frac{1}{\alpha_{k+1}} \cdot (\mu_k - \tilde{\mu}). \tag{3}$$

Intuitively, the threshold for surrogate usage in the next time step $(k + 1)$ is a function of the previous threshold, the estimated relationship between threshold and error at the current time, and the difference between the measured and desired error.

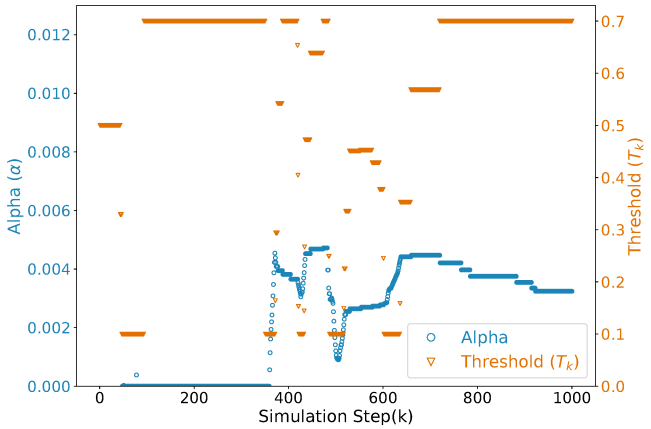
The above approximation of α works well in practice, however to insure stability, it is necessary to bound the change in threshold, t_k by a maximum change of ±0.1. In the latter case (i.e., if V_{k+1} is produced), we also compute a new distance threshold, T_{k+1}, bound by reasonable operating minimum and maximum thresholds. We also compute μ_{k+1}, the MAE of V_{k+1}[N] (i.e., the average of the absolute differences between the y and y' values in the N most recent elements of V_{k+1}). A potential problem could arise if Equation 3 oscillates, producing large swings in threshold from one update to another. This could occur if the approximation of α is consistently off by more than a factor of 2 [16]. However, the bounding of the threshold described above prevents extreme oscillation in practice. Furthermore, Proxima can detect the attempted oscillation and report it to the scientist for further examination.

To provide intuition as to how this update rule works, consider three cases. 1) If $\mu_{k+1} = \tilde{\mu}$, i.e., if the MAE of V_{k+1}[N] is equal to the user's desired maximum MAE, T is left unchanged. 2) If $\mu_{k+1} > \tilde{\mu}$, i.e., if the MAE of V_{k+1}[N] is greater than the user's desired maximum MAE, T is increased, by an amount that is larger if α is smaller. 3) If $\mu_{k+1} < \tilde{\mu}$, i.e., if the MAE of V_{k+1}[N] is less than the user's desired maximum MAE, T is decreased, by an amount that is smaller if α is larger. Figure 3 shows the operation of the update rule in practice; the α-threshold relationship is clearly visible.

The above starts with a basic control formulation (Equation 2), which is provably convergent to the goal using basic control analysis [16]. The convergence proof relies on the rate of change in the threshold. Our modifications to the basic control formulation (in Equation 3 and the preceding paragraph) can only reduce the change in threshold, never increase it. Therefore, we expect Proxima's control formulation to converge under any circumstances where the basic approach will converge. Proxima, may converge more slowly, however, because it may choose to reduce the change in threshold.



(a) 500 K



(b) 800 K

Figure 3: Proxima examples of the relationship between α and threshold T. In these two simulation runs, at 500 K and 800 K, the threshold is directly effected and changed by α.

3.3 Configuration

To apply Proxima to a specific problem, we must establish a target function and a machine learning model. For the machine learning model we also need a *distance metric* for the features and an *accuracy metric* for the prediction.

The only required user parameter is the target error which should be more intuitive to estimate than a distance threshold; we assume scientists know an acceptable error and that can be determined without profiling. In contrast, prior work required scientists to determine a threshold that may not have an intuitive mapping to error. If desired, the user may specify the number of training-set

```
# Xo: Initial state of molecule
# T:  Temperature
# N:  Number of steps to simulate
def simple_monte_carlo(Xo, T, N):
    X = Xo
    for n in range(N):
        X_next = perturb(X, T)
        E = energy_function(X) # Target function
        R = random()
        if accepted(E, R, T):
            X = X_next
```

Listing 1: Pseudo-code for the Monte Carlo sampling application used in experiments.

initialization steps, window comparison size, and initial distance threshold. However, the results were not particularly sensitive to the variations in the default settings.

4 EXPERIMENTAL SETUP

We next describe the example application that we use both to illustrate the use of Proxima and to evaluate various aspects of its performance. This application is run on an Intel Core i7-8700 CPU with 16GB of memory. The Monte Carlo sampling application (MCSA), for which pseudo-code is provided in Listing 1, computes the average property of an atomic system, using the Psi4 simulation code [45] as the underlying energy calculator.

The Monte Carlo algorithm makes small perturbations to the system, choosing whether to accept the perturbation as a new starting point based on a probability related to the energy change, and then repeating for many iterations. The average of the value of a property over all iterations is the expected value at the set temperature (T) if the acceptance probability is $P(\Delta E) = \max\{\exp(\frac{-\Delta E}{kT}), 1\}$, where ΔE is the energy change [34]. An example of a physical property that can be computed in this way is the average radius of gyration of a molecule, which is expected to increase with temperature.

In order to converge on a realistic structure, we need an accurate energy calculation at every Monte Carlo step. Therefore, Proxima's job is to speed up the simulation while capturing the energy as accurately as possible.

To instantiate Proxima, a Python wrapper, also called Proxima, is used as an interface to the application. The arguments to Proxima are the target function, the machine learning model, and the MAE bound. For this example, Bayesian ridge is the machine learning model used, the data in the training set are represented using SOAP, and the decision engine uses a Coulomb Matrix representation to quickly calculate whether or not to use the surrogate model. We report results in the following with a MAE bound of 0.002, unless otherwise stated.

We use the energy calculated by Psi4 as ground truth. We measure error as the MAE of all steps, with the baseline for each step being the Psi4 prediction. For steps where the surrogate energy is used, there will be some error. For steps where the surrogate energy is not used, there will be no error. The MAE includes both of these cases, unless otherwise stated.

In the MCSA example considered here, the target function takes a molecule as its argument, and molecules are represented as a

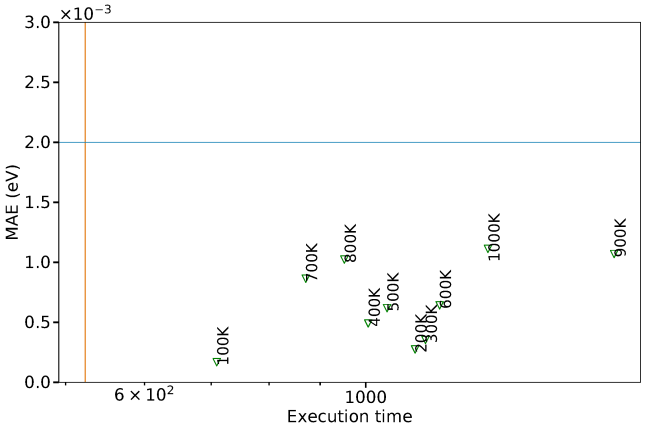


Figure 4: Results of running `Fixed` with T = 0.3 and no retrain interval, for temperatures 100–1000 K in increments of 100 K. Results show slow downs of up to 5× when compared to a no surrogate application.

multi-dimensional Coulomb matrix [38] used to calculate distances and a SOAP representation as featurization for the training data.

MCSA parameters are the target molecule (e.g., methane), the temperature at which the molecule is to be simulated, the perturbation size, the number of steps to be run, and a random seed.

4.1 Baseline Workflow

This section presents methods used to evaluate Proxima. We compare Proxima to a non-surrogate system, which we call `Baseline`, and to a fixed parameter surrogate system, which we call `Fixed`. `Fixed` uses prior work where the scientist is responsible for configuring the surrogate usage by setting an appropriate threshold for surrogate usage. In the following sections, strategies for `Baseline`, `Fixed`, and the methodology for acquiring the best fixed parameters are discussed.

4.1.1 Baseline and Fixed Surrogate Strategies. We define two strategies against which we compare Proxima proper in later sections.

The `Baseline` strategy runs the target function in response to every request. We determine the accuracy of other methods by comparing the result obtained at each step against that achieved by the target function; thus, `Baseline` has, by definition, the highest accuracy, as it uses Psi4 to calculate all energies. However, as it uses the target function at every step, its computational cost is high.

The `Fixed` strategy runs with a fixed value for the threshold T and retrains the surrogate model after a specific number of new data points, the retrain interval (RI), have been added to the dataset. As we explain below, we performed parameter sweeps in which `Fixed` was run with a variety of (T, RI) combinations in order to study sensitivity to those two parameters. These optimal parameters were used to run `Fixed` for 10 temperatures between 100 and 1000 K.

Finally, we performed runs with a lazy training method used in Proxima, in which retraining is performed only if the last step used a target function. This is the case when the model is only retrained if the input data are calculated to be within the specified T. Figure 4, shows that no runs, even with a conservative threshold of 0.3, met

both the error and time bounds. Therefore, lazy training was not used for Fixed.

4.2 Establishing Best Fixed Parameters

We next discuss how we find the optimal parameters for Fixed. These parameters achieve the best speedup while staying below a given mean absolute error bound at 500 K and 1000 K.

Due to the significant stochastic variation of the application, we must use *reference data* to compare the performance and accuracy of different methods. Reference data are the saved atom coordinates and energies obtained from a simulation that uses the target function only. Using reference data allows for an equivalent comparison between parameter choices. They are needed because the atomistic simulations that we consider here proceed by starting with a molecule's atoms in a particular state, and then repeatedly using the target function to compute potential energies on those atoms and then using the computed potential energies to update the positions of the atoms. As a result, the molecule's atoms trace out a trajectory in space: a trajectory that is highly sensitive to minor perturbations, so that a small change in potential energies (as might result from the use of a surrogate rather than the target function to compute potential energies) can result in the simulation following an entirely different trajectory.

Such differences between trajectories are not a problem scientifically, because atomistic modeling is concerned not with individual trajectories but with the statistics of many trajectories. However, they make comparisons of different methods on the basis of individual runs challenging, because different trajectories might involve different numbers of surrogate function evaluations as the molecule visits different parts of molecular space. To overcome this problem, we use what we call reference data. First, we perform a simulation using only the target function, saving all atomic coordinates and energies. Then, when running other methods that we want to compare with that first simulation, we make the molecule follow exactly the same trajectory.

In the end, we still need Proxima performance data without using any reference data, which is shown below, to compare full runs.

Using reference data, we ran a parameter sweep on Fixed for $7 \times 17 = 119$ parameter combinations (T, RI) in ($T \in \{0.1, 0.2, ..., 0.7\}$) $\times$ ($RI \in \{1, 2, 5, 10, ...50, 100, 200, ..., 500\}$), while keeping fixed the number of steps (1000), for temperatures at both at 500 K and 1000 K, the molecule (methane), random seed (1), and perturbation (0.003). For the 1000 K, the T of 0.1 is not taken into account as runs would take more than 24 hours to be finished, and would not be a parameter that could be used in the end. Therefore, we consider a total of 221 combinations: 119 at 500 K and 102 at 1000 K.

From this parameter sweep, we see in Figures 5 and 6, nine (RI, T) combinations for 500 K and 14 combinations for 1000 K that meet both the error and time bounds. Of these, four combinations meet the bounds for both temperatures. From those four, ($T = 0.3$, $RI = 50$) achieve the best speedup (2.31× for 500 K and 1.99× for 1000 K), while staying within the MAE bound for both 500 K and 1000 K. We establish that these are the best fixed parameters for Fixed.

5 RESULTS

We present the results of MCSA for three different methods: (1) no surrogate (Baseline), (2) surrogate with fixed parameters (Fixed), and (3) Proxima. We discuss the practical details of surrogate modeling with Proxima, compare it to other approaches, and discuss its scientific significance.

5.1 Accuracy and Speedup Results

We first run MCSA with the Baseline and Fixed strategies to obtain data for later comparisons with Proxima. In these runs, we keep fixed the number of steps (1000), molecule (methane), random seed (1), and perturbation (0.003).

Running MCSA first with the Baseline strategy (i.e., always using the target function), we observe that target-function execution takes a cumulative time of 523 seconds: an average of of 0.523 seconds per evaluation.

Next, we obtain results for Fixed. As the combination $T = 0.3$, $RI = 50$ gave the best speed and accuracy results for both 500 K and 1000 K, we ran Fixed with these parameter values. This combination was able to achieve a low MAE of 0.00149, with a 2.81× maximum speedup. Though Fixed can achieve high speedups at higher temperatures compared to Proxima, as illustrated in Figure 7, Fixed exceeds the error bound, especially at higher temperatures. For example, at 1000 K, Fixed's achieved error is over 50% greater than the bound. In fact, Fixed exceeds the error bound for all temperatures above 600 K. So while it can provide great speedups, those results are meaningless as the scientific simulation would have to be rerun to produce meaningful results. On the other hand, as illustrated in Figure 8, Proxima consistently stays within the given error bound across all temperatures. This is important, because exceeding the scientist-supplied bound by only a small amount can throw the scientific validity of a result into question, as we will explore in the next section. Finally, it is worth noting that the relatively low cost target function used in this work suggests that these speedups are conservative.

5.2 Scientific Significance of Surrogate Error

The results of the atomistic simulations considered in this paper can be used to derive a resulting secondary physical property, the radius of gyration (ROG). This quantity is highly sensitive to the accuracy of the entire Monte-Carlo trajectory and is not explicitly considered by Proxima during a run, but rather is determined only at the end of a simulation. Thus it provides a useful validation of the Proxima approach.

We shown in Figure 9, the ROG values obtained for 10 runs of each of Baseline, Fixed, and Proxima for temperatures from 100 to 1000 K. The multiple runs (with different initial random seeds) capture the variations that result from the randomness of the Monte-Carlo simulation.

We see in Figure 9 that Proxima outperforms Fixed in predicting an accurate ROG, achieving results that are closer to those of Baseline (indeed, coinciding with Baseline's error bars) and with less variation, as captured by the error bars, and without the increased variation in error with temperature that is seen for Fixed. This is further demonstrated in Figure 10, where the accuracy of Proxima is much more stable than that of Fixed. In other words,

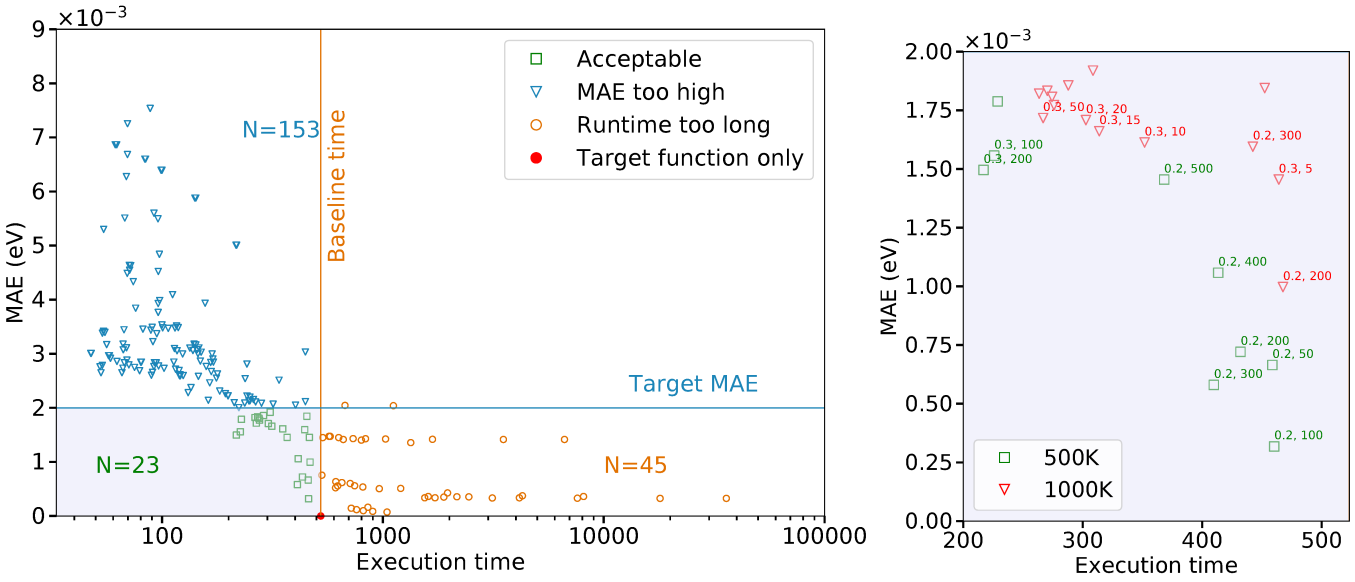


Figure 5: Left: MAEs and execution times achieved by `Fixed` with MCSA for methane at 500 K and 1000 K, for 221 different (*RI*, *T*) parameter combinations. The vertical line is the time taken by `Baseline` and the horizontal line is a target MAE. The `N=` numbers are of parameter combinations in different regions. Note that only the 23 parameter combinations in the lower left meet both error and time bounds. Also shown as a red circle at (0, 523) is the performance achieved by `Baseline`. Right: Highlighting the 500 K and 1000 K combinations that lie within the error and time bounds, with *RI*, *T* values shown for those with MAE less than 1.75. The best results are obtained with relatively low retrain intervals and distance thresholds.

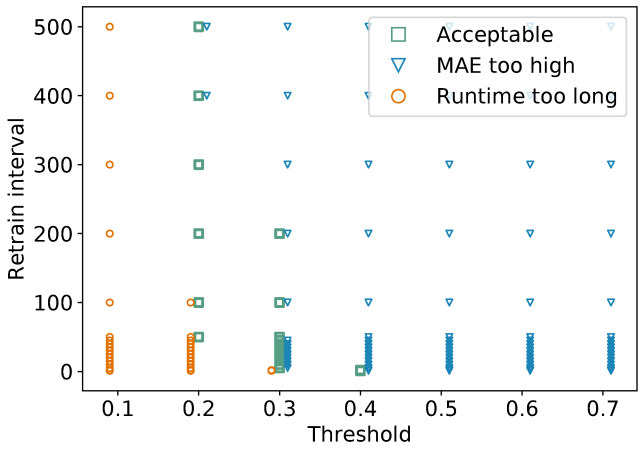


Figure 6: A scatter plot of the (*RI*, *T*) points in Figure 5, with markers classifying each point. The points with acceptable MAE and execution time (the green squares) fall in a relatively narrow range.

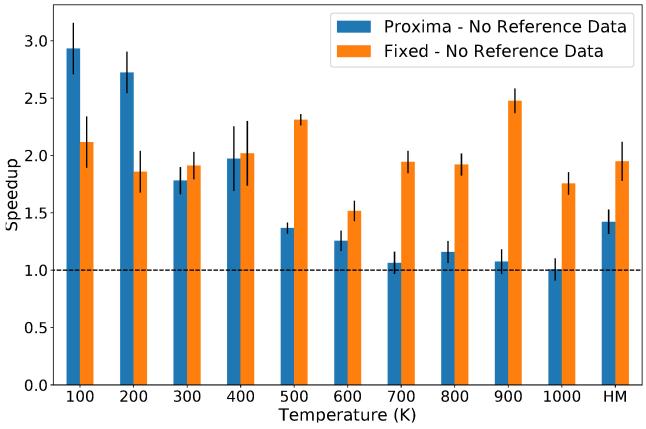


Figure 7: Speed-up results for `Fixed` (with parameters $T = 0.3$, $RI = 50$) and for Proxima without the use of reference data. The harmonic mean is labeled as HM.

Proxima achieves scientifically meaningful results where `Fixed` fails to do so.

5.3 Results for Different Error Bounds

Many existing frameworks for surrogate use consider only inference time and model error when selecting 'optimal' parameters, without detailed results of model training and decision-engine execution time, nor any focus on managing error [7, 8, 17]. The work presented here, in contrast, focuses on control mechanisms that can meet a user-defined error bound while optimizing end-to-end execution of an application across a range of temperatures. As we demonstrated above, the use of control mechanisms is important because even with extensive profiling, it is difficult to find parameters that satisfy an error bound across a range of temperatures.

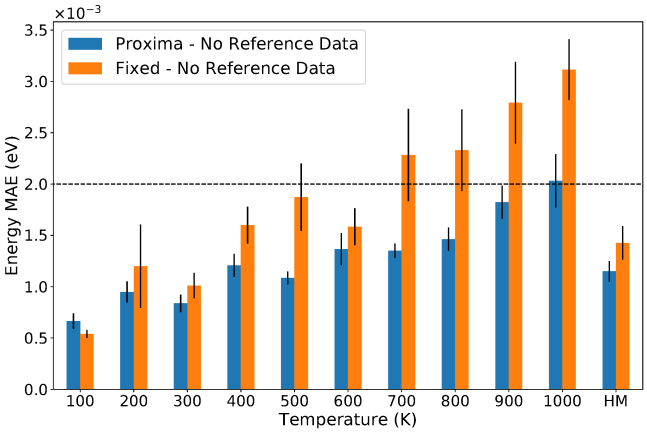


Figure 8: MAE of the energies predicted by surrogates, not including target function calls, across 10 temperatures. The harmonic mean is labeled as HM.

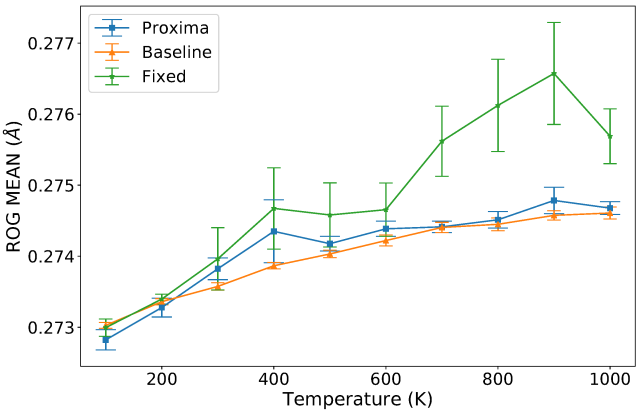


Figure 9: Mean of ROG comparison between `Baseline`, `Fixed`, and Proxima without the use of reference data. `Fixed` is with parameter values $T = 0.3$, $RI = 50$.

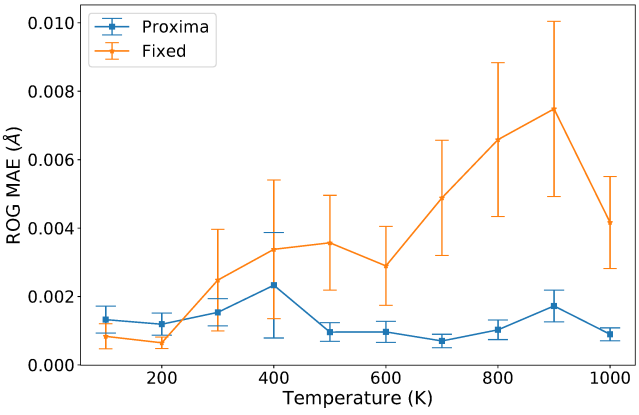


Figure 10: MAE of ROG comparison between `Fixed` and Proxima without the use of reference data.

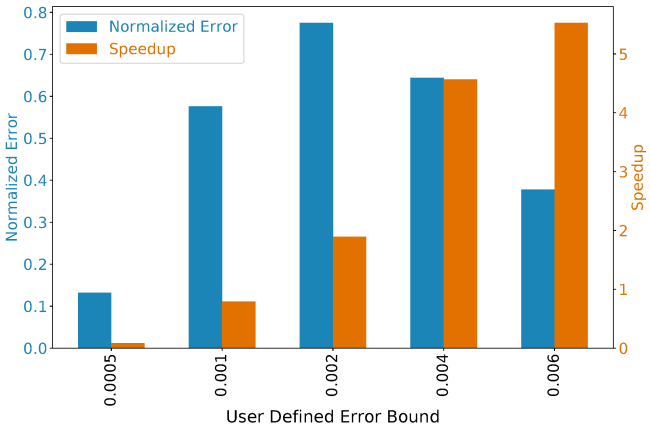


Figure 11: Proxima accuracy and speed vs. user-defined error bound. As the error bound increases from 0.0005 to 0.006, normalized error remains less than 1, indicating that Proxima always stays within the user defined error, while speedup increase to a maximum of 5.52.

We report here on experiments that evaluate Proxima's ability to meet a wide range of user-specified error bounds. Specifically, we run Proxima for error bounds in the range 0.0005 – 0.006 and measure the achieved error and speedup in each case. The results, displayed in Figure 11 presents the the error bound on the x-axis and the normalized error (where a value of 1 indicates the error bound, and values <1 indicate staying below the target error) on the y-axis. As expected, Proxima abides by the given error bounds, while achieving up to 5.52× speedup for the highest error bound. These results emphasize the point that surrogates are most useful when there is some room for error. The results with low error bounds are important as they show that Proxima performs acceptably even in those stressful situations; however, we should not be concerned that performance is poor in those cases, because they are not the cases that we are targeting.

5.4 Error Sensitivity and Reduction

Since both the energy landscape and surrogate model are nonlinear, a conservative distance threshold, T, for one configuration may result in significant error for another configuration. Therefore it is impossible to choose a static distance threshold that can satisfy a specific user-defined error bound. Proxima solves this problem by dynamically changing the threshold, illustrated in Figure 3, based on simulation error feedback and a user-defined error bound.

5.5 Proxima Overhead

The largest source of overhead in Proxima is the time needed to train the underlying surrogate model. While model inference is typically orders of magnitude faster than the target function, training time grows with training set size, and can reach 60% of total runtime in worse-case scenarios. Proxima's decision engine can also be a source of meaningful overhead. For MCSA, the decision engine requires a Euclidean-distance calculation based on a Coulomb-Matrix representation of the atomistic geometry. Calculating this distance

```
# Xo: Initial state of molecule
# T: Temperature
# N: number of steps to simulate
import Proxima
def simple_monte_carlo(Xo, T, N}:
   X = Xo
   # Make the Proxima wrapper
   prox_func = Proxima(calc.energy_function,
                       ml_model, mae_bound)
   calc.energy_function = prox_func
   for n in range(N):
      X_next = perturb(X, T)
      E = calc.energy_function(X) # Target function
      R = random()
      if accepted(E, R, T):
         X = X_next
```

Listing 2: Applying Proxima to MCSA.

can take as much as 9% of total runtime. The controller logic requires much less overhead, taking ~10μs per step. The reported Proxima speedups consider all costs, including Proxima logic,model (re)training, surrogate usage, and inference.

5.6 Ease of Use

Identifying the best parameters for `Fixed` required running 221 simulations. Running them all is expensive, because while some simulations run in five minutes, others take days. The results must then compared based on speedup and error obtained. Even removing the need for the retrain interval, and using the retraining technique applied in Proxima, results in only four configurations stay within both error and latency bounds, as shown in Figure 4. Additionally, Figure 8 shows that parameter values that work well for one temperature are not necessarily effective at other temperatures, where they result in errors above a given bound. Proxima removes these steps while staying below the user-defined error bound and achieving speedup: see Figure 7.

The user no longer needs to run the many simulations to find the best parameters and can import Proxima as a simple Python library, as shown in Listing 2.

6 RELATED WORK

The potential for performance gains via the integration of machine learning into atomistic simulations has spurred much research in this area [7, 8, 31, 37, 46].

Botu and Ramprasad developed a numerical fingerprint to represent an atom configurations and proposed an algorithm for surrogate decision usage [7, 8]. They use this numerical fingerprint as a feature vector to represent the atom coordinates in a way that can then be mapped to molecular properties, such as energy and force. Similarly to Proxima, they add training data each time that the target function is invoked and choose the surrogate when the input is within a certain distance threshold of the training data; in contrast to Proxima, they use a static threshold. Proxima's dynamically changing threshold allows it to invoke the surrogate model more often while meeting a required error bound, and thus to achieve higher performance.

Vandermause et al. use the internal uncertainty of a Gaussian process regression model to decide whether to accept a model prediction or to use the target function [46]. They applied their on-the-fly learning methodology to a range of single and multi-element systems. Though they attempt to keep the amount of training data low, their methodology retrains the model after every data addition and they do not discuss speedups.

Rupp et al. describe a surrogate implementation, presenting results on accuracy and discusses using hyperparameters. They used a specified training set and found that machine learning can predict potential energy with high accuracy [37].

In summary, although prior work has yielded many advances in the integration of surrogate models into atomistic simulations, none are able to guarantee a user-specified level of accuracy, as Proxima has shown to be able to do.

Other related work has investigated methods for accelerating surrogate model creation via automated model selection and learning algorithms that mimic the underlying structure of algorithms [2, 40]. That work requires the full training data set prior to to prediction, but could potentially be adapted to improve the models used within Proxima.

7 CONCLUSION

We have presented Proxima, a novel method for simplifying the incorporation of machine-learning-based surrogate models into science applications. A surrogate model is effective when it is sufficiently accurate for scientific goals and the cost of its (re)training is less than that saved by its use. We used the example of atomistic simulations to illustrate the challenges inherent in the resulting speed-accuracy tradeoffs, which are typically too complex for users to navigate without extensive and expensive experimentation. We showed how simple approaches to the integration of surrogate models, in which default values are used for various surrogate model configuration parameters, can easily result in inaccurate results and/or extreme slowdowns. We also showed how the complexity of such simulations means that users cannot readily identify good values for parameters without performing extensive experimentation. We then showed how with Proxima, a user does not need to perform extensive testing, curate a training set, or pre-train a machine learning model. Instead, Proxima used control theory to determine values for configuration parameters automatically, in ways that satisfy error bounds while also delivering substantial speedups: up to 5.52× in the case studied here.

We have focused in this paper on a simple atomistic modeling problem, namely computing the energy of methane, the simplest hydrocarbon. In future work, we will apply the method to larger atomistic modeling problems, where we expect Proxima to provide yet greater benefits. The relationship between threshold and error is not expected to change with increased problem size, so we expect Proxima to continue to meet the error bounds. And because Proxima replaces a target function that scales as $O(n^3)$ with a linear surrogate, the speedups could be even larger for larger problems. By delivering large speedups with only minor modifications to the science application, Proxima thus further opens the capabilities of using machine learning in these science applications.

8 ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their comments, which we have used to improve the paper. Funding support for this work comes from NSF (grants CCF-2028427, CNS-1956180, CCF-1837120, CNS-1764039), ARO (grant W911NF1920321), and a DOE Early Career Award (grant DESC0014195 0003). LW and GS were supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

REFERENCES

[1] Katie Antypas, BA Austin, TL Butler, RA Gerber, Cary Whitney, Nick Wright, Woo-Sun Yang, and Zhengji Zhao. NERSC workload analysis on Hopper. *Lawrence Berkeley National Laboratory Technical Report*, 6804:15, 2013.

[2] Prasanna Balaprakash, Ananta Tiwari, Stefan M Wild, Laura Carrington, and Paul D Hovland. AutoMOMML: Automatic multi-objective modeling with machine learning. In *International Conference on High Performance Computing*, pages 219–239. Springer, 2016.

[3] Albert P Bartók, Risi Kondor, and Gábor Csányi. On representing chemical environments. *Physical Review B*, 87(18):184115, 2013.

[4] Jörg Behler. Perspective: Machine learning potentials for atomistic simulations. *The Journal of Chemical Physics*, 145(17):170901, 2016.

[5] F Matthias Bickelhaupt and Evert Jan Baerends. Kohn-Sham density functional theory: Predicting and understanding chemistry. *Reviews in computational chemistry*, 15:1–86, 2000.

[6] Kurt Binder, Jürgen Horbach, Walter Kob, Wolfgang Paul, and Fathollah Varnik. Molecular dynamics simulations. *Journal of Physics: Condensed Matter*, 16(5):S429, 2004.

[7] Venkatesh Botu, Rohit Batra, James Chapman, and Rampi Ramprasad. Machine learning force fields: Construction, validation, and outlook. *The Journal of Physical Chemistry C*, 121(1):511–522, 2017.

[8] Venkatesh Botu and Rampi Ramprasad. Adaptive machine learning framework to accelerate ab initio molecular dynamics. *International Journal of Quantum Chemistry*, 115(16):1074–1083, 2015.

[9] Venkatesh Botu and Rampi Ramprasad. Learning scheme to predict atomic forces and accelerate materials simulations. *Physical Review B*, 92(9):094306, 2015.

[10] Markus J Buehler. *Atomistic modeling of materials failure*. Springer Science & Business Media, 2008.

[11] Keith T Butler, Daniel W Davies, Hugh Cartwright, Olexandr Isayev, and Aron Walsh. Machine learning for molecular and materials science. *Nature*, 559(7715):547–555, 2018.

[12] Marco Caccin, Zhenwei Li, James R. Kermode, and Alessandro De Vita. A framework for machine-learning-augmented multiscale atomistic simulations on parallel supercomputers. *International Journal of Quantum Chemistry*, 115(16):1129–1139, June 2015.

[13] Christopher J Cramer and FM Bickelhaupt. Essentials of computational chemistry. *Angewandte Chemie*, 42(4):381–381, 2003.

[14] Yixin Diao, Joseph L Hellerstein, Sujay Parekh, Rean Griffith, Gail Kaiser, and Dan Phung. Self-managing systems: A control theory foundation. In *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 441–448. IEEE, 2005.

[15] Scott E Field, Chad R Galley, Jan S Hesthaven, Jason Kaye, and Manuel Tiglio. Fast prediction and evaluation of gravitational waveforms using surrogate models. *Physical Review X*, 4(3):031006, 2014.

[16] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated design of self-adaptive software with control-theoretical formal guarantees. In *36th International Conference on Software Engineering*, pages 299–310, 2014.

[17] Nathan A Garland, Romit Maulik, Qi Tang, Xian-Zhu Tang, and Prasanna Balaprakash. Progress towards high fidelity collisional-radiative model surrogates for rapid in-situ evaluation. In *3rd Workshop on Machine Learning and the Physical Sciences*. PMLR, 2020.

[18] Luigi Genovese, Matthieu Ospici, Thierry Deutsch, Jean-François Méhaut, Alexey Neelov, and Stefan Goedecker. Density functional theory calculation on many-cores hybrid central processing unit-graphic processing unit architectures. *The Journal of chemical physics*, 131(3):034103, 2009.

[19] Torkel Glad and Lennart Ljung. *Control theory*. CRC press, 2018.

[20] Andrea Grisafi, David M Wilkins, Gábor Csányi, and Michele Ceriotti. Symmetry-adapted machine learning for tensorial properties of atomistic systems. *Physical Review Letters*, 120(3):036002, 2018.

[21] Mohamed Hacene, Ani Anciaux-Sedrakian, Xavier Rozanska, Diego Klahr, Thomas Guignon, and Paul Fleurat-Lessard. Accelerating VASP electronic structure calculations using graphic processing units. *Journal of computational chemistry*, 33(32):2581–2589, 2012.

[22] J Hafner. Atomic-scale computational materials science. *Acta Materialia*, 48(1):71–92, 2000.

[23] Christopher Michael Handley and Jörg Behler. Next generation interatomic potentials for condensed systems. *The European Physical Journal B*, 87(7), July 2014.

[24] Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. PID controllers. In *Feedback Control of Computing Systems*, chapter 9, pages 293–335. John Wiley & Sons, Ltd, 2004.

[25] Lauri Himanen, Marc OJ Jäger, Eiaki V Morooka, Filippo Federici Canova, Yashasvi S Ranawat, David Z Gao, Patrick Rinke, and Adam S Foster. DScribe: Library of descriptors for machine learning in materials science. *Computer Physics Communications*, 247:106949, 2020.

[26] Henry Hoffmann. CoAdapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *26th Euromicro Conference on Real-Time Systems*, pages 223–232. IEEE Computer Society, 2014.

[27] Henry Hoffmann. JouleGuard: Energy guarantees for approximate applications. In Ethan L. Miller and Steven Hand, editors, *25th Symposium on Operating Systems Principles*, pages 198–214. ACM, 2015.

[28] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin C. Rinard. Dynamic knobs for responsive power-aware computing. In Rajiv Gupta and Todd C. Mowry, editors, *16th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 199–212. ACM, 2011.

[29] Giulio Imbalzano, Andrea Anelli, Daniele Giofré, Sinja Klees, Jörg Behler, and Michele Ceriotti. Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials. *The Journal of Chemical Physics*, 148(24):241730, 2018.

[30] T. L. Jacobsen, M. S. Jørgensen, and B. Hammer. On-the-fly machine learning of atomic potential in density functional theory structure optimization. *Physical Review Letters*, 120(2), January 2018.

[31] Alireza Khorshidi and Andrew A Peterson. Amp: A modular approach to machine learning in atomistic simulations. *Computer Physics Communications*, 207:310–324, 2016.

[32] Zhenwei Li, James R. Kermode, and Alessandro De Vita. Molecular dynamics with on-the-fly machine learning of quantum-mechanical forces. *Physical Review Letters*, 114(9), March 2015.

[33] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Automated control of multiple software goals using multiple actuators. In Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman, editors, *11th Joint Meeting on Foundations of Software Engineering*, pages 373–384. ACM, 2017.

[34] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[35] Phani Motamarri, Sambit Das, Shiva Rudraraju, Krishnendu Ghosh, Denis Davydov, and Vikram Gavini. DFT-FE–A massively parallel adaptive finite-element code for large-scale density functional theory calculations. *Computer Physics Communications*, 246:106853, 2020.

[36] Andrew A Peterson, Rune Christensen, and Alireza Khorshidi. Addressing uncertainty in atomistic machine learning. *Physical Chemistry Chemical Physics*, 19(18):10978–10985, 2017.

[37] Matthias Rupp. Machine learning for quantum mechanics in a nutshell. *International Journal of Quantum Chemistry*, 115(16):1058–1073, 2015.

[38] Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole Von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Retters*, 108(5):058301, 2012.

[39] Faizan Sahigara, Kamel Mansouri, Davide Ballabio, Andrea Mauri, Viviana Consonni, and Roberto Todeschini. Comparison of different approaches to define the applicability domain of QSAR models. *Molecules*, 17(5):4791–4810, April 2012.

[40] Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. *arXiv preprint arXiv:1902.09701*, 2019.

[41] Jonathan Schmidt, Mário RG Marques, Silvana Botti, and Miguel AL Marques. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Materials*, 5(1):1–36, 2019.

[42] John C Slater. A simplification of the Hartree-Fock method. *Physical review*, 81(3):385, 1951.

[43] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18, pages 1257–1264. MIT Press, 2006.

[44] Martin Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-time systems*, 14(3):219–250, 1998.

[45] Justin M Turney, Andrew C Simmonett, Robert M Parrish, Edward G Hohenstein, Francesco A Evangelista, Justin T Fermann, Benjamin J Mintz, Lori A Burns, Jeremiah J Wilke, Micah L Abrams, et al. Psi4: An open-source ab initio electronic

structure program. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 2(4):556–565, 2012.

[46] Jonathan Vandermause, Steven B Torrisi, Simon Batzner, Yu Xie, Lixin Sun, Alexie M Kolpak, and Boris Kozinsky. On-the-fly active learning of interpretable Bayesian force fields for atomistic rare events. *npj Computational Materials*, 6(1):1–11, 2020.

[47] Max Veit, Sandeep Kumar Jain, Satyanarayana Bonakala, Indranil Rudra, Detlef Hohl, and Gábor Csányi. Equation of state of fluid methane from first principles with machine learning potentials. *Journal of Chemical Theory and Computation*, 15(4):2574–2586, 2019.

[48] Nicholas Wagner and James M Rondinelli. Theory-guided machine learning in materials science. *Frontiers in Materials*, 3:28, 2016.

[49] Logan Ward, Ruoqian Liu, Amar Krishna, Vinay I Hegde, Ankit Agrawal, Alok Choudhary, and Chris Wolverton. Including crystal structure attributes in machine learning models of formation energies via Voronoi tessellations. *Physical Review B*, 96(2):024104, 2017.

[50] Mitchell A Wood, Mary A Cusentino, Brian D Wirth, and Aidan P Thompson. Data-driven material models for atomistic simulation. *Physical Review B*, 99(18):184305, 2019.