THE UNIVERSITY OF CHICAGO

MACHINE LEARNING FOR PERFORMANCE ACCELERATION AND PREDICTION IN SCIENTIFIC COMPUTING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
YULIANA ZAMORA

CHICAGO, ILLINOIS
JUNE 2022

Dedication Text

Epigraph Text

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

ABSTRACT

Scientific applications often require massive amounts of compute time and power. With the constantly expanding architecture landscape and growing complexity of application runs, understanding how to improve performance is vital. In this thesis, we will use machine learning in two distinct ways to improve science applications.

First, we use a data-driven approach and leverage machine learning to understand and improve performance in high-performance computing applications. The goal of this work is to create a streamlined workflow of integrating machine learning surrogates into such applications. Using control theory to automatically and dynamically configure parameters, we can meet accuracy constraints while maximizing performance. This workflow, which we examine in the context of molecular dynamics simulations, will allow for faster and larger simulations by improving overall performance. By replacing typically high-cost functions, such as density functional theory calls or Hartree-Fock calls, with a low-cost machine learning inference call, our proposed workflow can reduce run-time while producing scientifically usable results. We create a decision engine that will automate finding the accuracy and performance trade-off relationship between using a high-fidelity, high-cost function call, and a lower fidelity machine learning inference.

Second, in an effort to understand future performance, we focus on predicting performance across multiple architectures. Cross-architecture metric mapping is heavily studied with hopes of understanding application performance on future or untested machine architectures. Often, there are months or even years spent on porting applications to new architectures, that may or may not provide an increase in performance. Here, we will use a data-driven approach to predict total throughput across different GPU architectures in order to understand future success of these applications. Our goal is to create a general framework that can predict application performance on a target hardware, given performance metrics from a different hardware architecture, without expert input. In this thesis, we propose such

a framework and use it to predict total throughput and IPC between two GPU architectures.

CHAPTER 1

ACCELERATING SCIENTIFIC COMPUTING

1.1 Proxima: Accelerating the Integration of Machine Learning in Atomistic Simulations

1.1.1 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 [5]. These applications are often dominated by the repetitive execution of a few high-cost functions [48, 18]. In the past, speedups in these domains have relied on advances in hardware architecture and numerical-algorithm development [47, 43, 89]. However, recent advances in machine learning have enabled another promising acceleration technique: replacing expensive functions with machine-learned *surrogates* [15, 40]. 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 [97, 21, 120, 20]. 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 [20, 19]—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 work 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 [114]. 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]

1.1.2 Related Work

The potential for performance gains via the integration of machine learning into atomistic simulations has spurred much research in this area [20, 19, 113, 96, 67].

Botu and Ramprasad developed a numerical fingerprint to represent an atom configurations and proposed an algorithm for surrogate decision usage [20, 19]. 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 [113]. 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

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

learning can predict potential energy with high accuracy [96].

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 [12, 100]. That work requires the full training data set prior to to prediction, but could potentially be adapted to improve the models used within Proxima.

1.1.3 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.

1.1.3.1 Atomistic Modeling

Atomistic modeling computes interactions between atoms to capture the complex behavior of materials and molecules [23]. 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 [17, 104]. They also quickly become limited in terms of the size of the system that can be simulated, as computational requirements scale

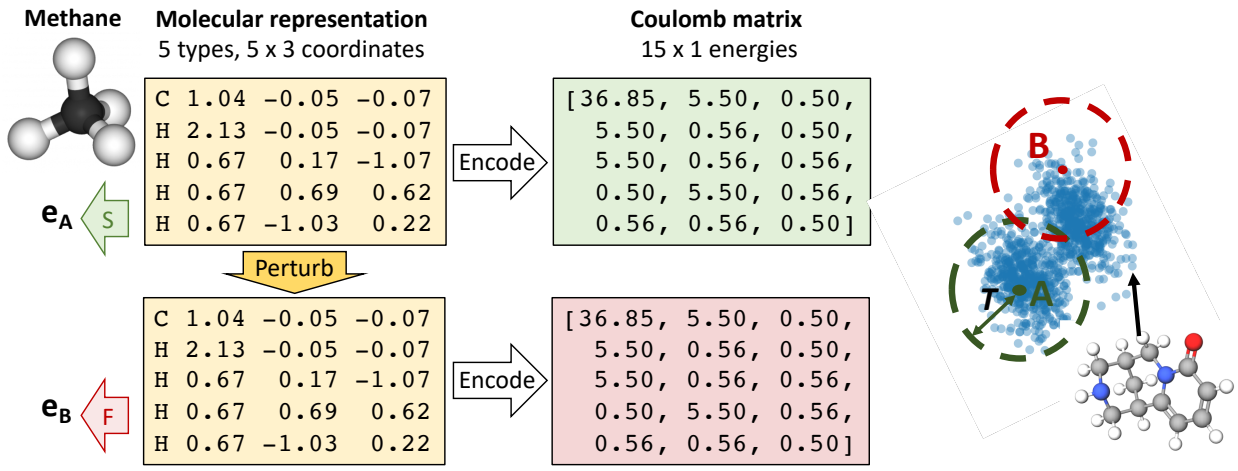


Figure 1.1: Illustrates the fundamental ideas that this work leverages in atomistic modeling. A molecule state is represented by coordinates, say A. These can be perturbed to yield a different state, B. A state can be encoded in a Coulumb matrix, which when treated as a point in a multi-dimensional space allows for distance computations. A is within a distance T of a previously evaluated state, B is not.

with the number of electrons cubed or worse [32]. 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 [23] and significant new applications are possible if the length and timescale of these models can be expanded.

1.1.3.2 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 [50, 14]. 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 [54, 46, 61, 116, 24, 101, 119].

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 [14] 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 [54] 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. [19], 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. [97], describes an atom based on the atomic numbers and pairwise distances between atoms. As illustrated in Figure 1.1, 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.

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 [76, 20, 64]. 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) [93, 98]. 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. [19], 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 [93], 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. [25] 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 [107] provide easy tradeoffs between model accuracy and inference

speed. The additional opportunities for performance optimization further motivate the need to automatically tune performance parameters.

1.1.3.3 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 [45]. 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 [53, 111, 36, 41, 78], with a particular emphasis on controlling accuracy and performance tradeoffs [55, 56, 57]. 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 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 1.1.4.1.

1.1.4 Methodology

1.1.4.1 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

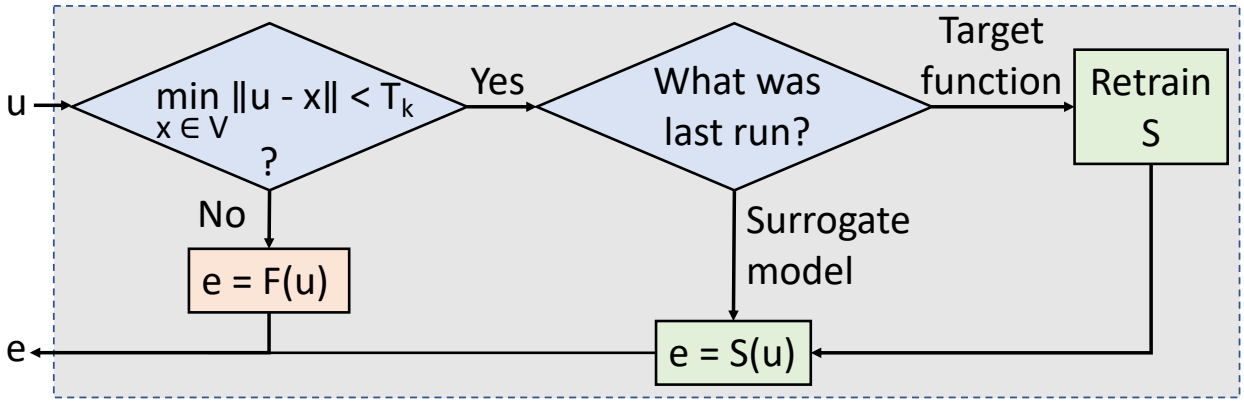


Figure 1.2: Logical flow of the Proxima surrogate modeling process. At upper left, an input value `u` is received and checked relative to a distance threshold from recently evaluated values. Ultimately either the target function **F** or the surrogate model are used to compute the return value `e`. A 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.

on the distance logic shown in Figure 1.2, 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 1.1.3.2) 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.

1.1.4.2 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[N]:

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

If $d < T_k$, then we retrain the surrogate model $\mathbf{S}$ 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.

1.1.4.3 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}[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:

$$\mu = \alpha \cdot T, \tag{1.1}$$

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:

$$T_{k+1} = T_k - \frac{1}{\alpha} \cdot (\mu_k - \tilde{\mu}) \tag{1.2}$$

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 1.1, 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 thesis 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 $(x_i, y_i, y'_i, d_i) \in V_{k+1}[\mathrm{N}]$. We then compute this dynamic version of α_{k+1} and use it in Equation 1.2:

$$T_{k+1} = T_k - \frac{1}{\alpha_{k+1}} \cdot (\mu_k - \tilde{\mu}). \tag{1.3}$$

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 1.3 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 [41]. 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 1.3 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 1.2), which is provably convergent to the goal using basic control analysis [41]. The convergence proof relies on the rate of change in the threshold. Our modifications to the basic control formulation (in Equation 1.3 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.

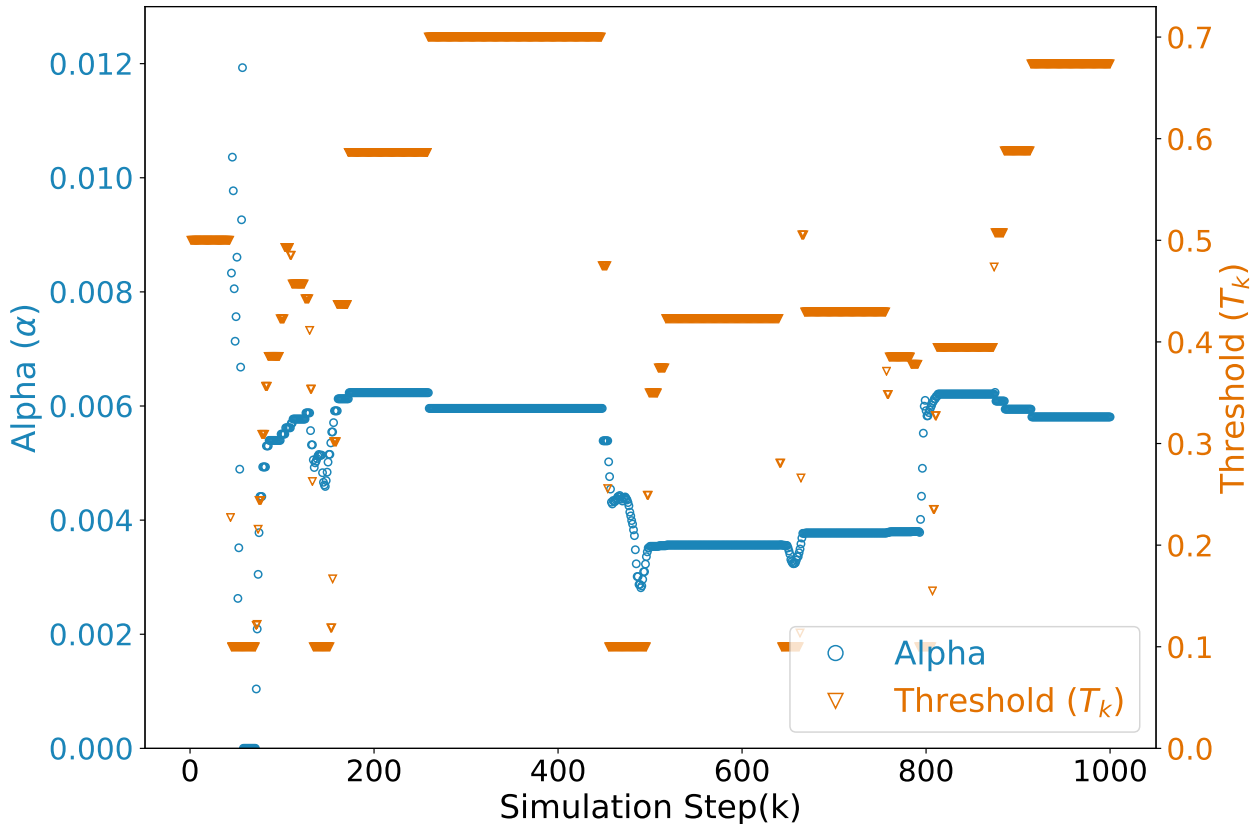
1.1.4.4 Configuration

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.

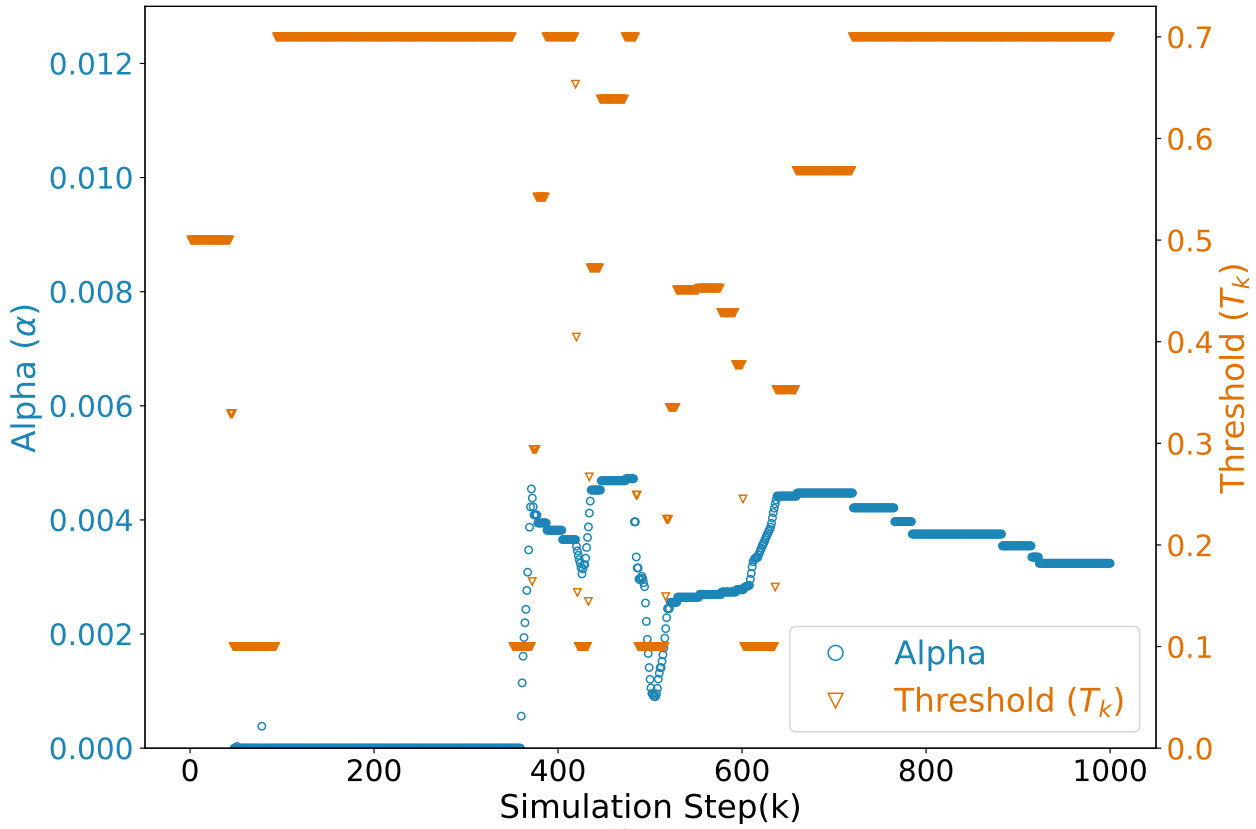
The only required user parameter is the target error which should be more intuitive to estimate than a distance threshold; we assume scientists know an acceptable error and that can be determined without profiling. In contrast, prior work required scientists to determine a threshold that may not have an intuitive mapping to error. If desired, the user may specify the number of training-set initialization steps, window comparison size, and initial distance threshold. However, the results were not particularly sensitive to the variations in the default settings.

1.1.4.5 Experimental Setup

```
# 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):
```



(a) 500 K



(b) 800 K

Figure 1.3: 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.

```
X = X_next
```

Listing 1.1: Pseudo-code for the Monte Carlo sampling application used in experiments.

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 1.1, computes the average property of an atomic system, using the Psi4 simulation code [112] 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 [84].

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 multi-dimensional Coulomb matrix [97] 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.

1.1.4.6 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.

1.1.4.7 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

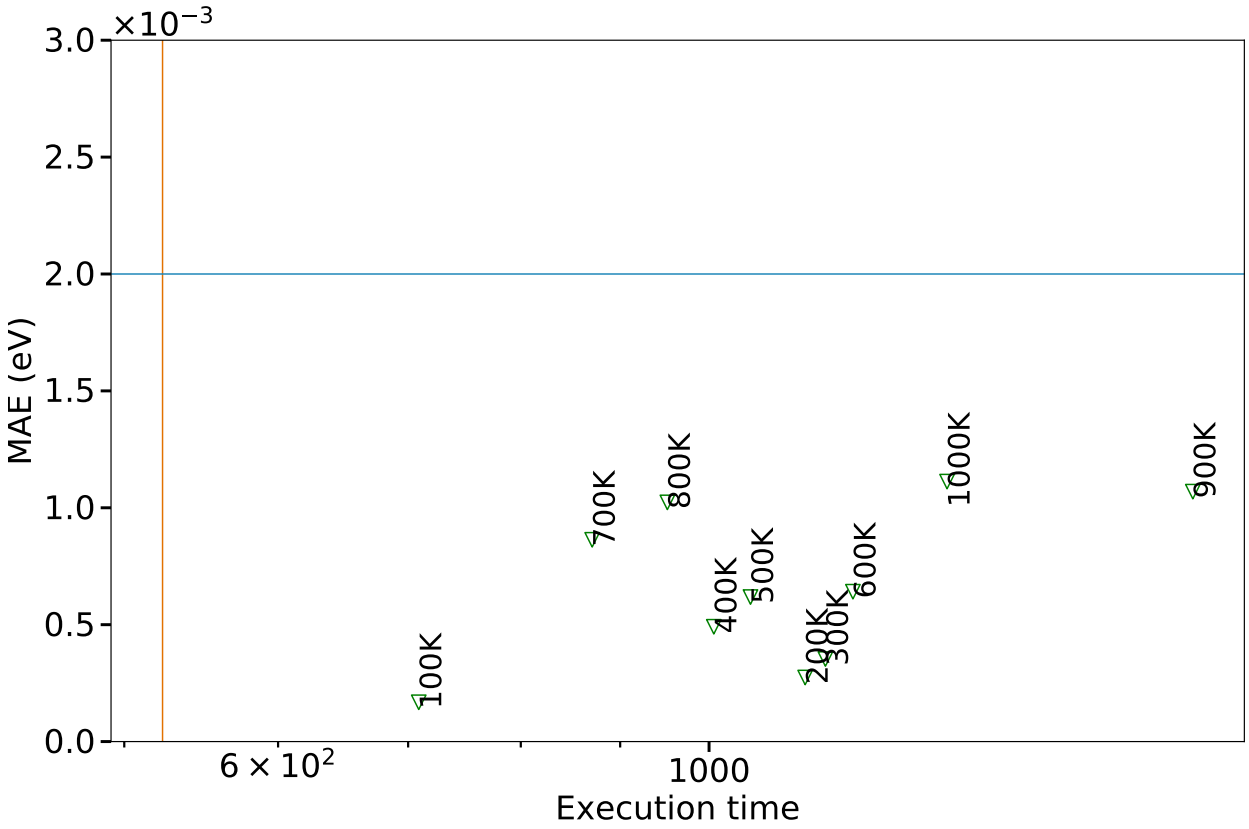


Figure 1.4: Results of running `Fixed` with $T = 0.3$ and no retrain interval, for temperatures 100–1000 K in increments of 100 K. Results show slow downs of up to 5× when compared to a no surrogate application.

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 1.4, 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`.

1.1.4.8 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

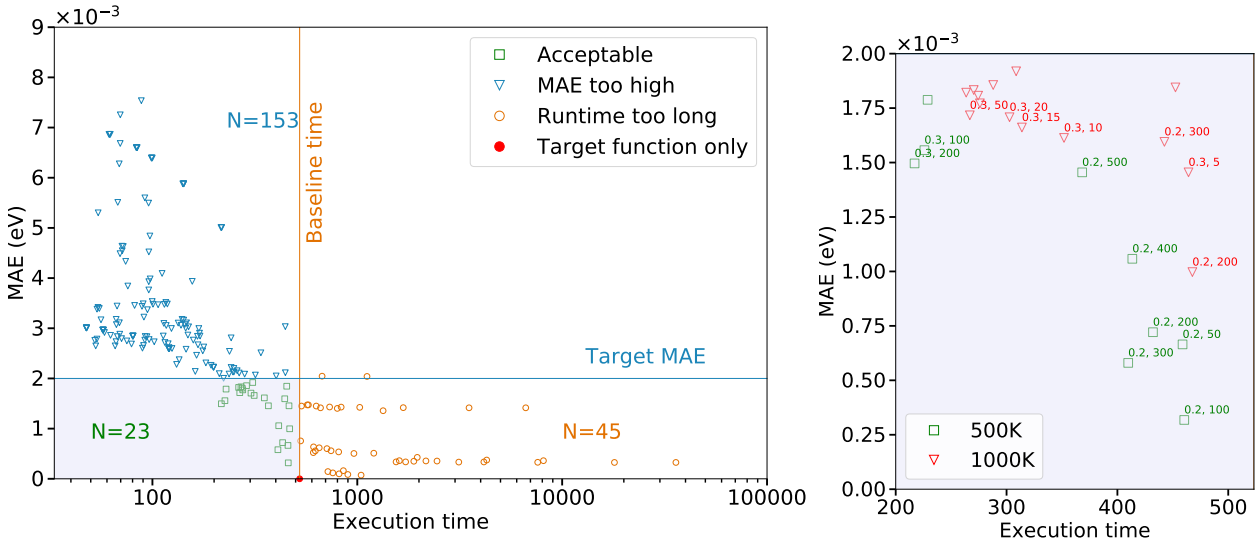


Figure 1.5: Left: MAEs and execution times achieved by `Fixed` with MCSA for methane at 500 K and 1000 K, for 221 different (RI, T) parameter combinations. The vertical line is the time taken by `Baseline` and the horizontal line is a target MAE. The `N=` numbers are of parameter combinations in different 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.

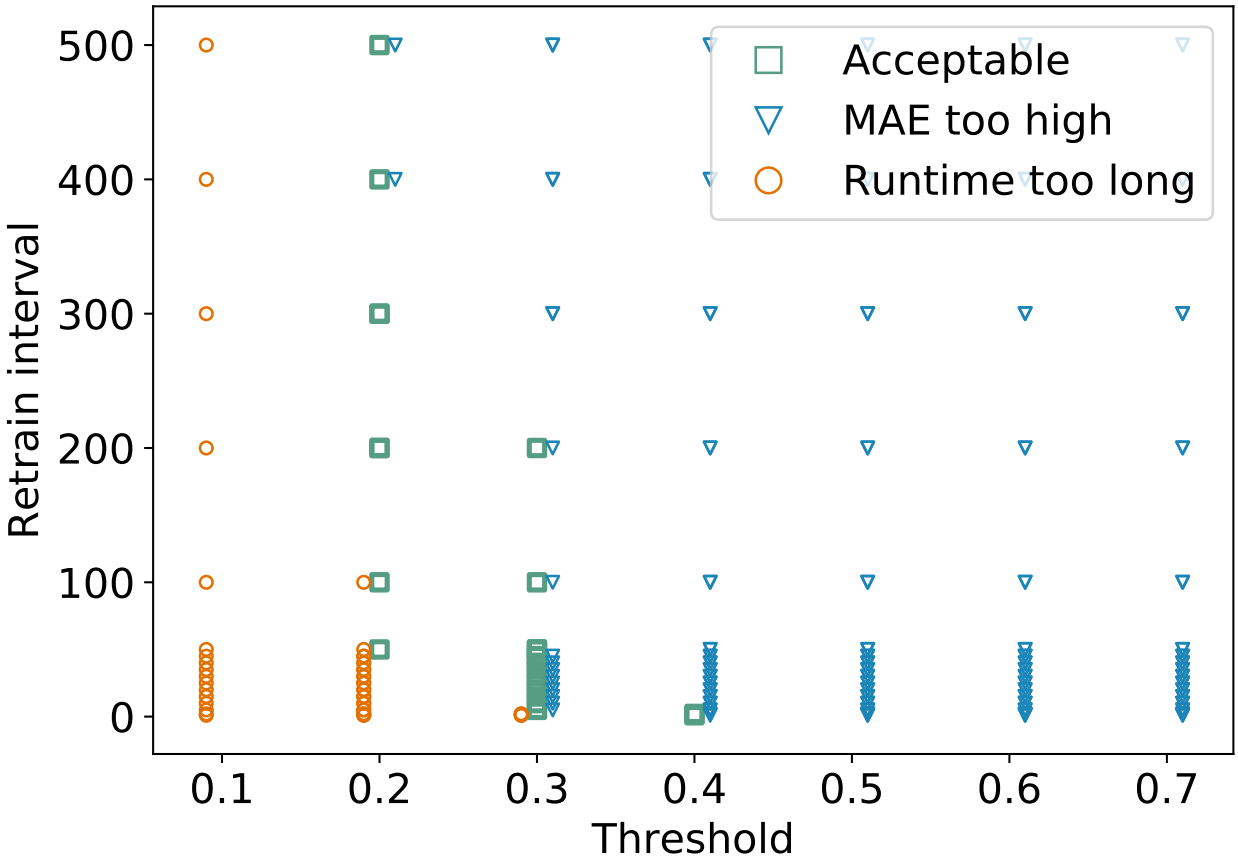


Figure 1.6: A scatter plot of the (RI, T) points in Figure 1.1.4.8, with markers classifying each point. The points with acceptable MAE and execution time (the green squares) fall in a relatively narrow range.

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 1.1.4.8 and 1.6, 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$\times$ for 500 K and 1.99$\times$ 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`.

1.1.5 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.

1.1.5.1 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 1.7, `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 1.8, 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

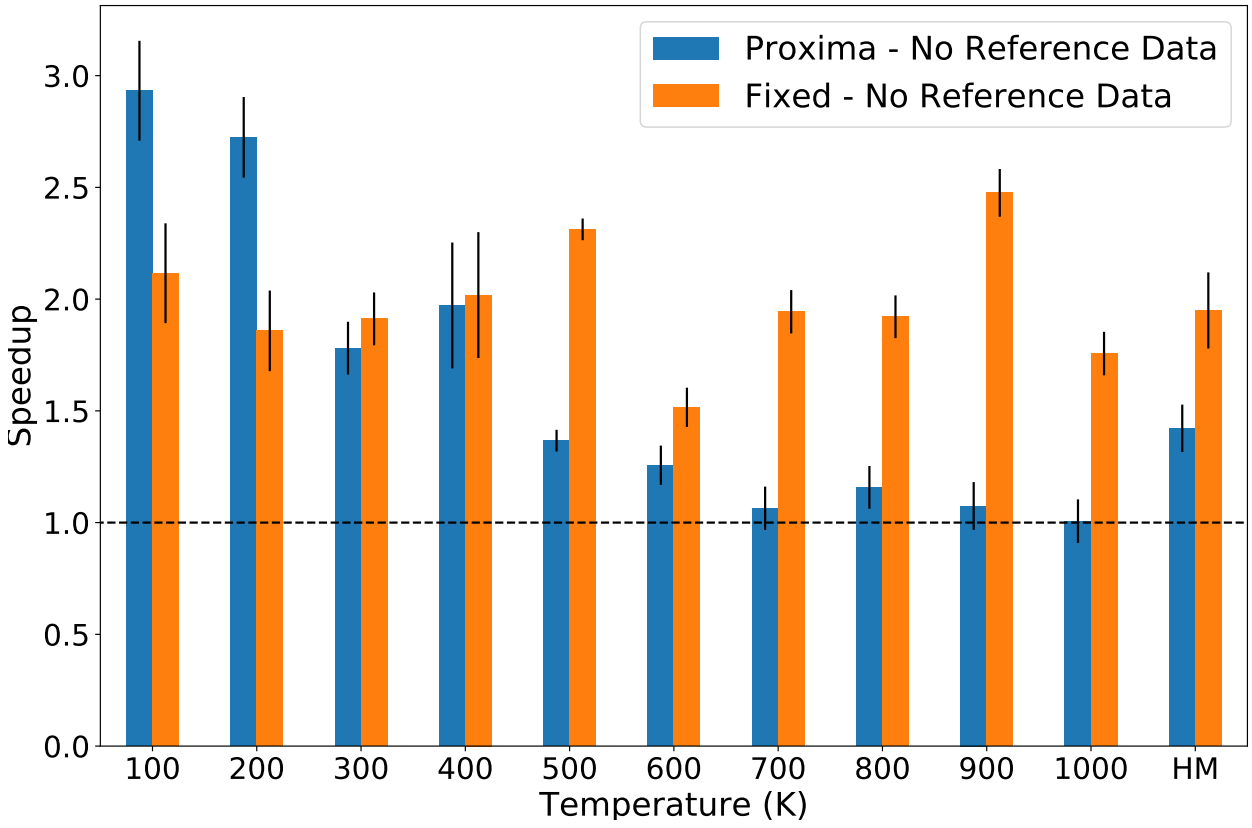


Figure 1.7: Speed-up results for `Fixed` (with parameters $T = 0.3$, $RI = 50$) and for Proxima without the use of reference data. The harmonic mean is labeled as HM.

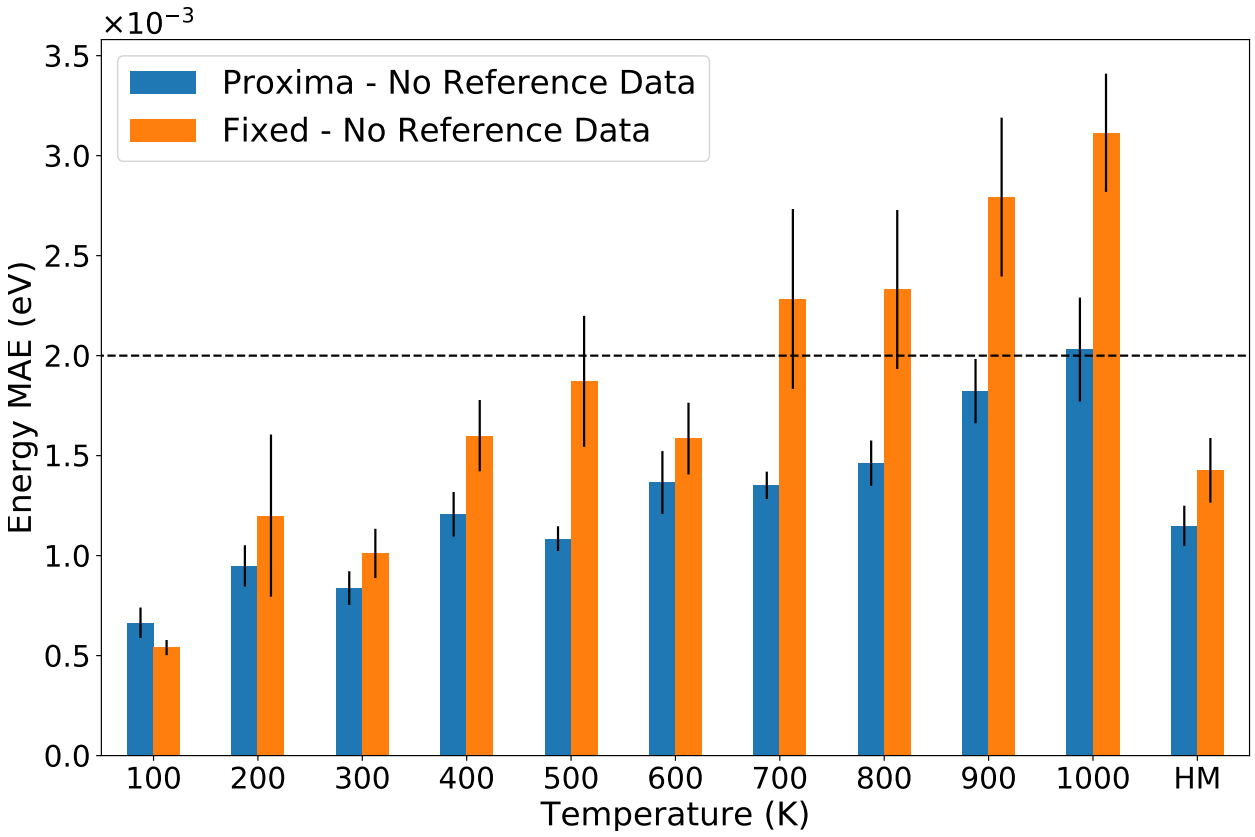


Figure 1.8: MAE of the energies predicted by surrogates, not including target function calls, across 10 temperatures. The harmonic mean is labeled as HM.

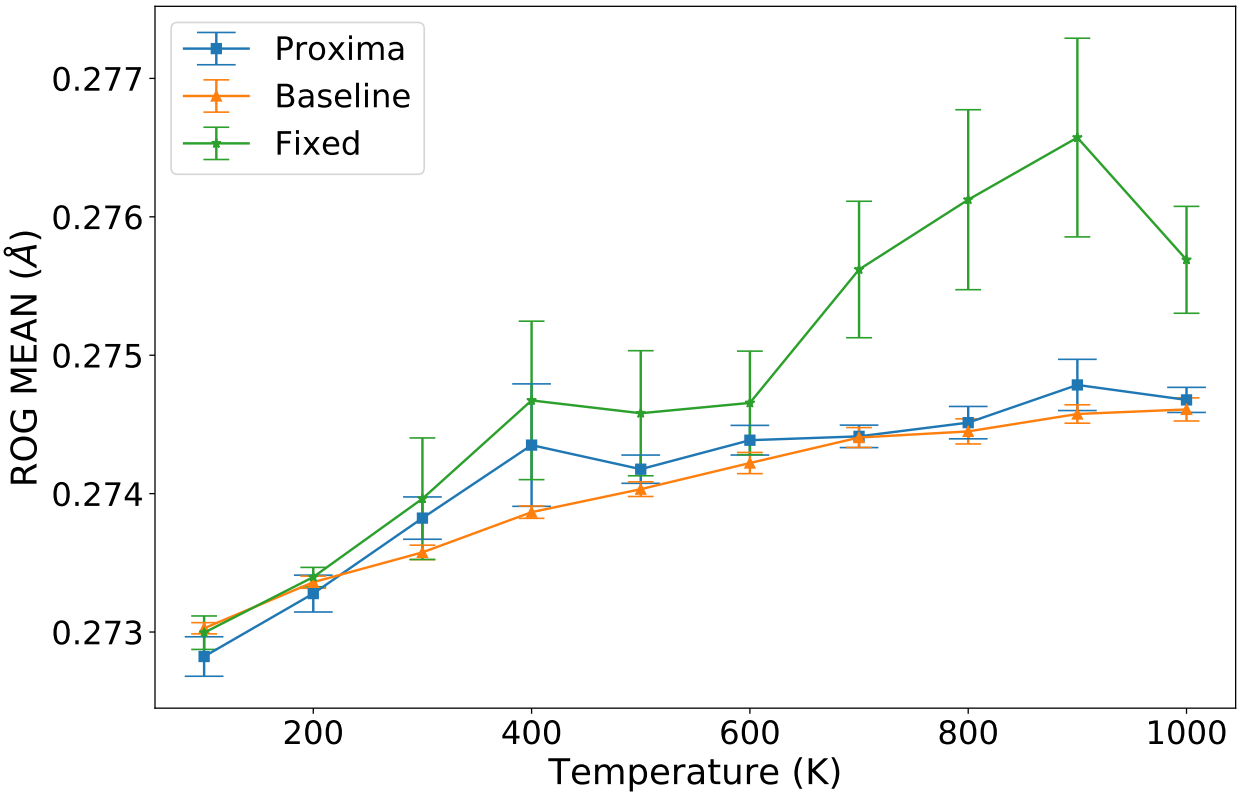


Figure 1.9: Mean of ROG comparison between `Baseline`, `Fixed`, and Proxima without the use of reference data. `Fixed` is with parameter values $T = 0.3$, $RI = 50$.

work suggests that these speedups are conservative.

1.1.5.2 Scientific Significance of Surrogate Error

The results of the atomistic simulations considered in this thesis 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 1.9, 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

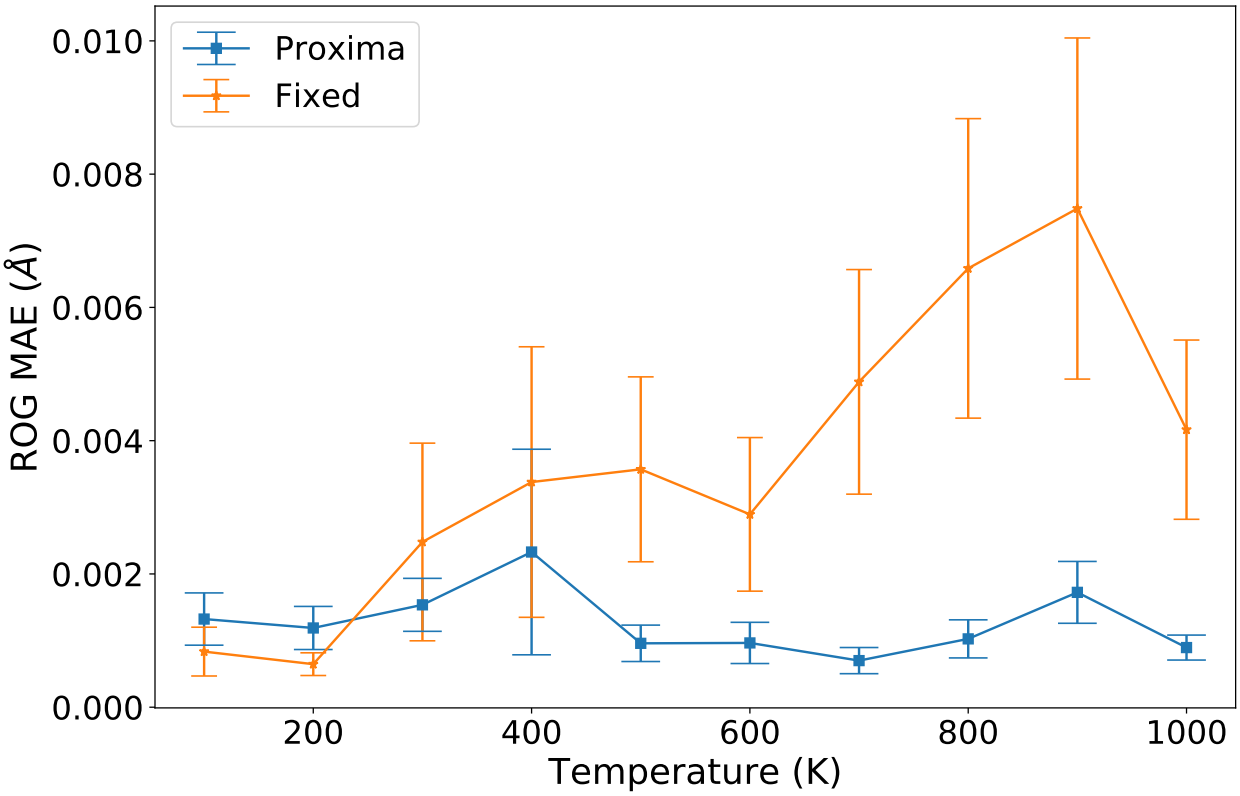


Figure 1.10: MAE of ROG comparison between `Fixed` and Proxima without the use of reference data.

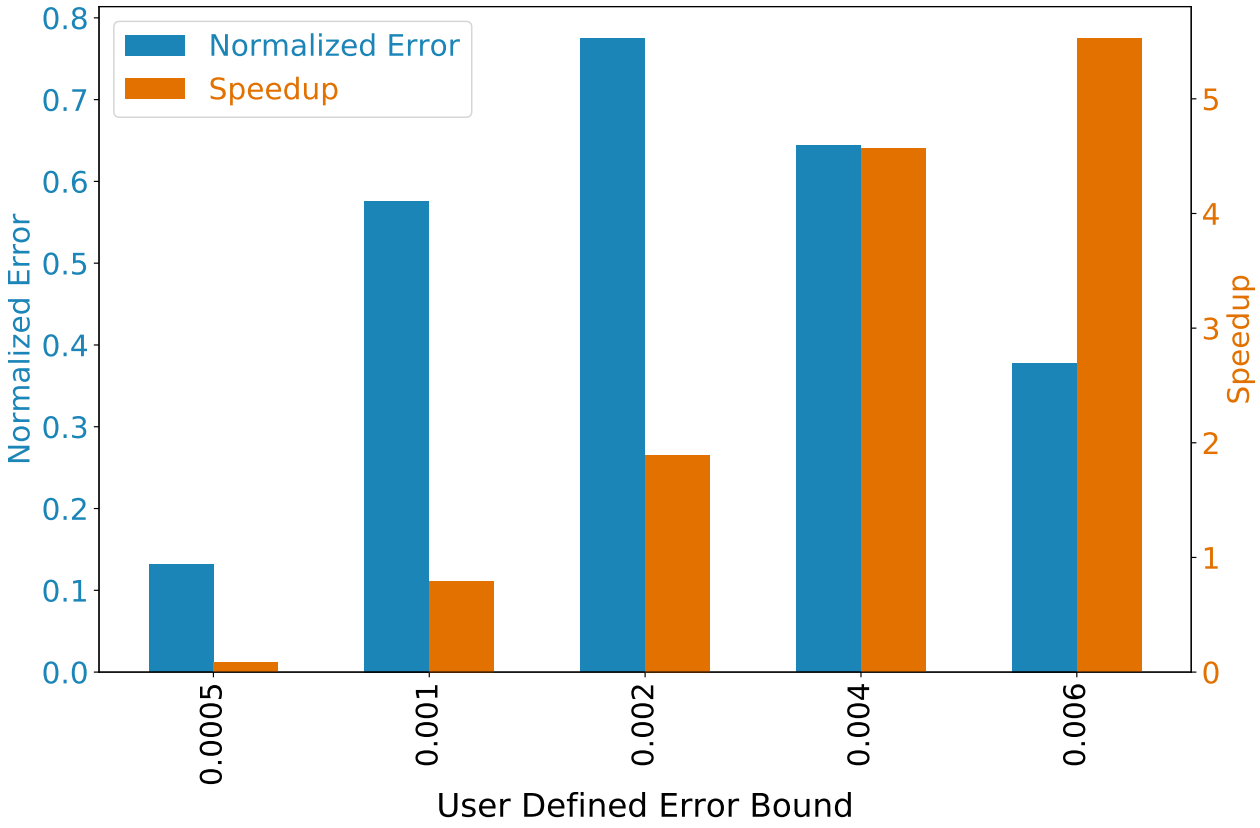


Figure 1.11: Proxima accuracy and speed vs. user-defined error bound. As the error bound increases from 0.0005 to 0.006, normalized error remains less than 1, indicating that Proxima always stays within the user defined error, while speedup increase to a maximum of 5.52.

simulation.

We see in Figure 1.9 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 1.10, where the accuracy of Proxima is much more stable than that of `Fixed`. In other words, Proxima achieves scientifically meaningful results where `Fixed` fails to do so.

1.1.5.3 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 [42, 19, 20]. 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.

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 1.11 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.

1.1.5.4 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 1.3, based on simulation error feedback and a user-defined error bound.

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

1.1.5.6 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 1.4. Additionally, Figure 1.8 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 1.7.

```
# 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 1.2: Applying Proxima to MCSA.

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 1.2.

1.1.6 Summary

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 work 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.

1.2 Dynamic On-The-Fly Integration of Surrogates in Molecular Dynamics Simulations

1.2.1 Introduction

As an extension of Proxima, we take our formulation one step further and approach a more complex problem in the world of molecular dynamics simulations. From drug to material discoveries, molecular dynamics (MD) simulations capture behaviour that would otherwise not be seen or require extensive expensive realistic simulation [51, 58]. MD can help the scientist understand an insight at the level of the atomic structure. Additionally, it is not only

a snapshot, but follows the movement of individual molecules and reproduces the behavior using model systems [94]. Hollingsworth et al. explain how molecular dynamics simulations can "predict how every atom in a protein or other molecular system will move over time based on a general model of the physics governing interatomic interactions." With the ability to both help interpret experimental results and guide the work, the popularity and accessibility of MD simulations is only growing. The trajectory produced in the Monte Carlo work from the previous chapter, requires one scalar value prediction for each step in the system. Here, we are looking at a molecular dynamics application, where each time-step requires a vector prediction for every atom in the system, in addition to changing the system's size and state (solid, liquid, solid and liquid).

The complexity with integrating machine learning into this MD problem comes from the need of the ML model to accurately predict states it has not necessarily trained on. To integrate machine learning into MD applications, prior methods require many iterations to find fixed usage parameters. As discussed in the previous chapter, it can take many iterations to set parameters for the Monte Carlo simulation, but setting the correct usage bounds is intensified in molecular dynamics because the ramifications of poor force predictions is likely to be more significant. In our work, we show an advanced version of Proxima to enable the dynamic usage of machine learning, without the need to run many iterations to find usage parameters or being tied to the specific MD algorithm.

The more atoms or complex the material is, the more complex the interactions will be to simulate. Running these simulations take a significant amount of compute time - taking up to several months to run large simulations, quickly becoming limited in size due to computational requirements (scaling with the number of electrons cubed) [33, 93]. There have been many attempts at accelerating molecular dynamics simulations - from high cost, gold-standard approximations, Density Functional Theorem (DFT) to lighter weight approximations, effective medium theory potential (EMT), to machine learning models. Often used

in high-cost design, scientists want to make sure approximations are as close to first-principle calculations with a shorter time-frame of execution. Current common methodologies rely on using one type of approximation for the entirety of the simulation, i.e. either using machine learning or more classical approximations. This is because atomistic modeling presents a distinct set of challenges for machine learning. Being that new states,structures, or regions are often explored where training examples are sparse or not yet covered, it is not often not sufficient to assess a model's fit with typical training and validation sets [93].

With this challenge, there has been a focus on understanding when to use machine learned models or surrogates during the simulation. Some work has focused on on-the-fly usage of machine learning in MD simulations. Botu et al. created a fingerprint technique where the atom configuration is characterized in a way that a distance metric is used as a deciding fixed factor or threshold on model usage [19]. For example, a new input data point is compared to the current training set based on a Euclidean distance. After many iterations, the scientists can find the best distance threshold to use for the studied model and simulation. The current state-of-the-art research in on-the-fly surrogate usage, Li et al. leveraged predictor-corrector molecular dynamics to embed Gaussian process model within first-principles molecular dynamics [75]. Similarly, after many iterations, they too would find the best fixed boundaries or parameters of surrogate usage during the simulation. The limitations of these past approaches is in the requirement for using fixed parameters that involve many iterations to find. These usage parameters are limited to specific simulations. Past work tested on simulations that were not changing in state. For example, going from solid to liquid, liquid to solid, or changing boundary conditions. Additionally, usage parameters were often tied into the time-stepping algorithm itself [75], limiting utilization across differing applications.

To tackle the current limitation of requiring and finding fixed threshold parameters for on-the-fly machine learning, we introduce an advanced version of Proxima. Within a complex molecular dynamics application, we establish the relationship between the standard devia-

tion of a bootstrap ensemble of neural network-based calculators and force error, proposed by Peterson et al [93], can be used to dynamically adjust the surrogate usage threshold. This is in addition to using the more accurate force prediction provided by the model ensemble. For this simulation, a traditional classical approximation (EMT) is replaced by a deep learning model based on whether the standard deviation of the model ensemble is less than a usage threshold. The controller feedback mechanism uses force error to then dynamically change this usage threshold throughout the simulation. In order to examine a more complex molecular dynamics applications, we use a deep learning model and methodology created by the DeePMD team [118].

The contributions of this work are:

- The ability to dynamically use surrogate models in molecular dynamic simulations in a way that is completely agnostic to the algorithms used by the application
- Confirm that the controller works for molecular-dynamics by reducing calls to the high-fidelity function. Show that the controller is a critical asset in surrogate integration as a fixed parameter cannot achieve both accurate science results and reduced high-fidelity function calls together.
- Establish that using ensemble models for uncertainty is a suitable method for on-the-fly surrogate integration in molecular dynamics.
- Given an uncertainty quantification metric (here we use an ensemble approach, previously we used distance), we can dynamically modify and adjust usage bounds - which have been previously fixed and required many iterations to find.

1.2.2 Related work

Csanyi et al. use both high-fidelity (DFT) and low-fidelity (classical force model optimization) at simulation time. Using a predictor-corrector approach (there auditing scheme), they

created a flow that after a n number of time-steps, runs quantum calculations then tunes the parameters of the classical potential to increase the force prediction accuracy [34]. Using this same methodology, Li et al. replaced the classical force model with a Gaussian Process model. In this case, after n time-steps, the quantum calculations will be run and the new data will be added to the QM database to improve model accuracy [75]. Both these methods use a specific time stepping algorithm to incorporate two different force approximations. Proxima is not connected to the dynamics the way it is in these past approaches, because it is not connected to the time-stepping algorithm, nor does Proxima rely on a fixed number of time-steps for surrogate usage. Proxima can be used without changing any part of the simulation that uses it.

1.1 has further details in work related to ML accelerated molecular dynamics simulation.

1.2.3 Background

The control theory and Proxima method is explained in the last chapter. The modifications and improvements made are on the error metric used, how it is calculated, and the model used. The methodology behind these changes are explained in the following section.

Molecular dynamics simulations are based on Newton's laws of motion. Given the atom positions in a system (in our case, aluminium atoms), the simulation calculates the force exerted on each atom by all the other atoms in the system using Newton's Law of motion to predict the spatial position of each atom as a function of time [58]. Simulation then steps through time, and captures snapshots of atom positions at each step, where it calculates forces on each atom and use those forces to update the positions and velocity of each atom. In the end, the simulation returns a specific atom-configurations at each time-step.

We are able to take advantage of the substantial advances in deep learning for molecular dynamics. Specifically, we use models developed by the DeePMD team using their DeePMD-toolkit [118]. The toolkit is interfaced with TensorFlow, allowing the training process to be

automatic and efficient. Though we do not look at on-the-fly re-training in this work, it is something that could be useful in future work. The models created by the DeePMD-kit allows for efficient molecular simulations. The methodology and example they used was based on learning the inter-atomic potential energy and forces of a water model using data obtained from density functional theory. The team was able to show their methodology reproduced accurate structural information, with the team winning the 2020 Gordon Bell Prize for "pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning." With confidence in thee DeePMD methodology, we use the same set-up to find the inter-atomic potential forces to simulate the solid-liquid phase boundary of aluminum to find the melting temperature.

Surrogate usage is typically seen in examples when temperature is held constant. In order to show the effectiveness of our methodology, we incorporate the method into an application where the temperature is changing throughout the simulation. Though there are many approaches for melting point calculations, we use the detailed guideline, SLUSCHI, provided by Hong et al. SLUSCHI or "solid and liquid in ultra small coexistence with hovering interfaces," is a guideline created for wide community use. The creators of SLUSCHI outlines and offers a detailed guideline to perform melting point calculations [59]. The set-up is as follows: SLUSCHI starts with a crystal or solid structure that the user specifies. SLUSCHI then builds a supercell—a large cell built from many unit cells. Then SLUSCHI prepares the solid-liquid coexistence and employs the small-cell coexistence method - the methodology that calculates the melting temperature.

Furthermore, because calculating the uncertainty quantification (UC) was a larger driver in the Monte Carlo simulation of the previous Proxima version, we incorporate a new uncertainty quantification methodology. Being that ensemble methods have been established in the machine learning and material science community, we found to also be a promising approach in our methodology [93, 3, 37]. In this work, we use an ensemble of models for

both the end prediction value and UC. The prediction value is the mean of the model results and standard deviation is used for UC.

1.2.4 Methodology

In this section, we explain the modifications and improvements made to Proxima and how deep learning model is dynamically incorporated into SLUSCHI. The uncertainty quantification method, model error calculation, and type of model input and output used are the items updated. As discussed in the previous chapter, the Proxima controller is able to dynamically change the surrogate usage parameters using an error feedback during the simulation. The standard workflow, edited from the original Proxima workflow, is shown in Figure 1.12. The simulation's high-cost function will be replaced with another estimator, based on whether the current error is below a given threshold (T_k). Simply, if the calculated error is below the threshold, the surrogate will be used, otherwise, the simulation uses the original function. Unlike the previous chapter, the models in this workflow will not be re-trained.

The uncertainty quantification method and prediction result uses the prediction of an ensemble of models. Three models were used in this case to provide sufficient samples for standard deviation as our uncertainty. A thorough study was not performed in choosing the optimal number of models needed to balance simulation accuracy with performance. We leave this optimization for future work. The standard deviation produced by the ensemble of models is used as the uncertainty metric in the controller. The mean of the three model predictions is used as the final value in the error calculation and prediction value.

There are two ways we evaluated at error. Both the surrogate model and EMT() method return 3N force matrix. N is the number of atoms in the systems and the three corresponds to the x,y,c coordinates of the atoms. For the error, we take the L2 force error for each atom. Then, we take the maximum value over all atoms. This final value is what is taken as the error for that current snapshot or window and can help with highlighting error localized to

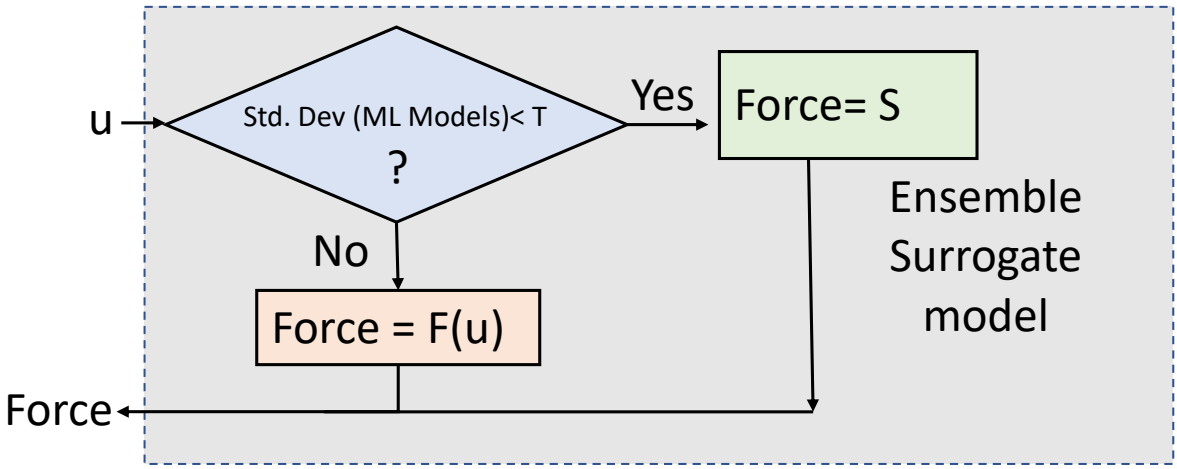


Figure 1.12: Logical flow of the Proxima-SLUSCHI surrogate modeling process. At upper left, an input value `u` is received as input to the ensemble of models where the standard deviation of the results is then calculated. Ultimately either the target function **F** or the surrogate model are used to compute the return value `Force`. 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 both time k and temperature T.

specific atoms in the simulation. This would a strict error as it would not let any atoms have an error about the given error threshold, regardless of the size of the system. For a less strict error calculation. We find the average of element wise difference of the force predictions.

The configurations of the SLUSCHI configurations are as follows. Being a commonly studied and used element, we also studied the melting temperature of aluminum. We start with a 4x4x4 super-cell of the face-centered cubic structure – a total of 256 aluminum atoms. After equilibrating this structure near the melting temperature, we duplicate the cell in the Z-direction and melt the new atoms at a temperature above the aluminum melting point while holding the original atoms fixed. The resultant half-solid, half-liquid cell has a total of 512 atoms. We equilibrate the system at a temperature of 900K (aluminium melting temperature is 933.5K).

We set an initial guess of melting temperature of 900K (aluminiums melting temperature

933.5K). There are 10,000 coexistence steps and 2,000 melting steps taken. For ease of iteration and development, we use the very fast EMT potential. EMT or effective medium theory is another common theoretical inter-atomic potential [29].

To use DeePMD, we first had to acquire training data, which is referred to as a list of systems where each system contains a number of frames. We ran the SLUSCHI application using different random seeds with the same configurations discussed above. These models are trained on NVIDIA A100 GPU's provided by Argonne's Computing Leadership Facility, taking around 3 hours to train. The models achieve a root mean squared error of 0.01 eV/A, where the loss across the training period can be seen in Figure 1.13. We trained and used three deep learning models created using the DeePMD software.

1.2.5 Results

We will show (1) the linear relationship between the UQ and error, allowing the controller to successfully adapt surrogate usage throughout the simulation, (2) target the given error, and (3) capture the necessary science. We will also discuss how using a constant threshold is not possible for this type of dynamic simulation.

First, to see the main contribution: dynamically changing the surrogate usage threshold based on error feedback, Figure 1.14, illustrates the changing thresholds throughout the simulation. As expected, this change in UQ, corresponds to the change in error during the simulation, as show in Figure 1.15. From Figure 1.16, the relationship between UQ and error can be seen, also illustrating how our intended maximum error is targeted. More specifically, this is the relationship that the controller is able to exploit in order to maximize surrogate usage.

Additionally, to explore how well the science was capture, we compare the melting results after 20 coexistence runs. That is, we calculate the percentage that the liquid-solid system turned all liquid or all solid. These results heavily depend on the user defined error threshold.

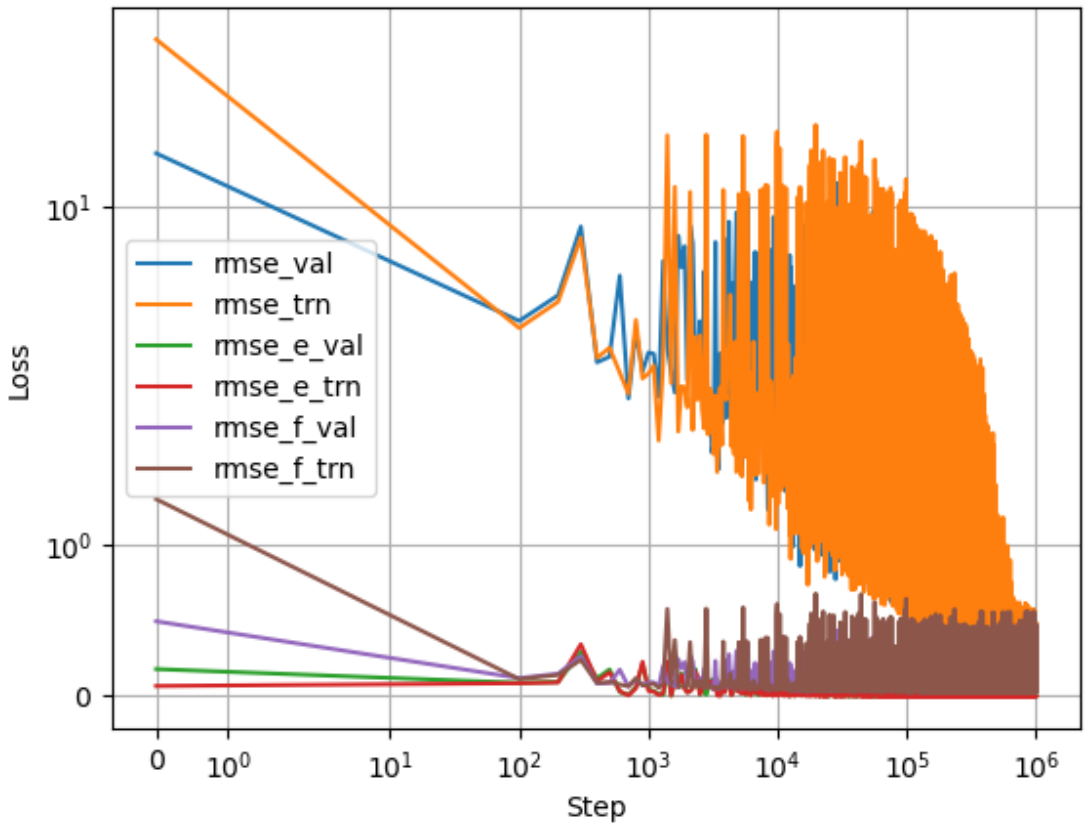


Figure 1.13: This is the learning curve of an aluminum system. The root mean square virial, energy, and force errors of the training and validation sets are presented against the training step.

Here, we used a high mean absolute error threshold of 0.03, to show heavy surrogate usage (65.35%). Three comparisons will be discussed here. Table 1.2.5, shows the results of SLUSCHI with EMT only, Proxima (EMT and ML), and a conservative fixed surrogate usage threshold (EMT and ML). This table clearly shows that even with a conservative fixed parameter, the approach does not work for this type of complex simulation, with changing states. Even with such a low usage parameter, the model was likely used in critical parts of the simulation, producing in incorrect final results.

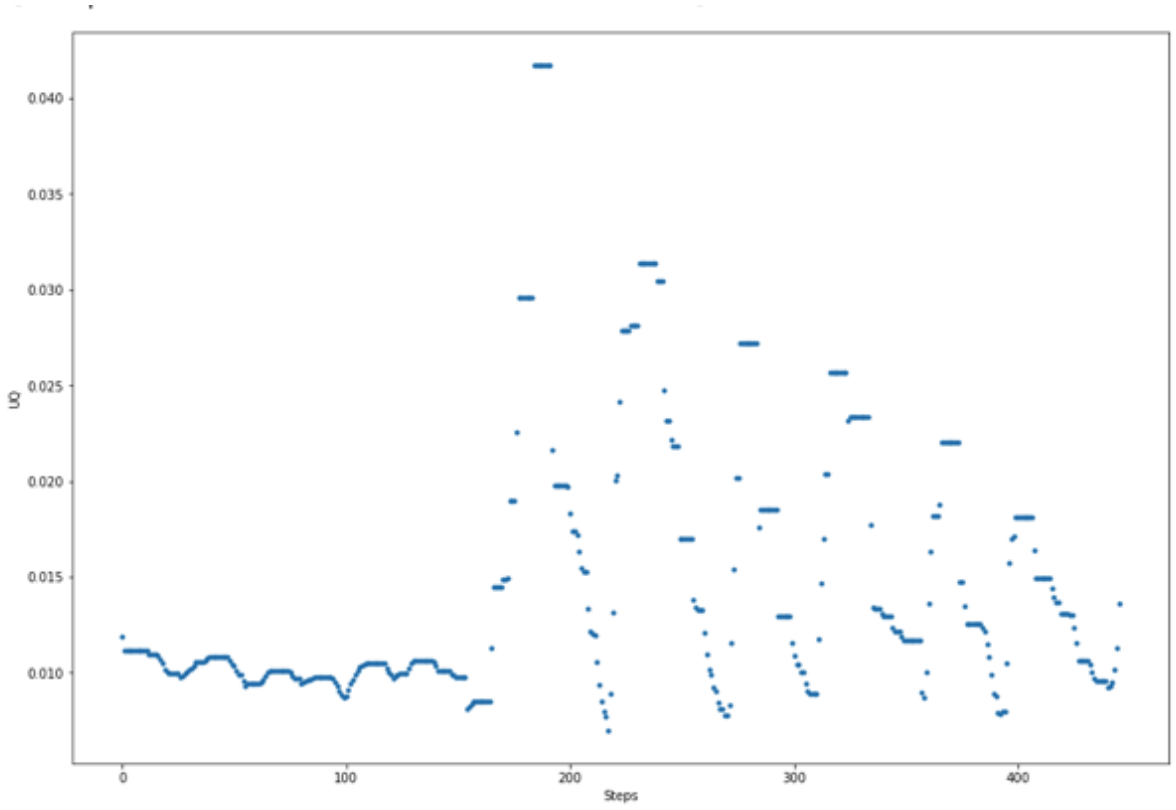


Figure 1.14: This graph presents the UQ threshold against the steps in the simulation.

Method	Liquid	Solid
EMT	87.5%	12.5%
Proxima	68.75	31.25%
Fixed	100%	0%

1.2.6 Summary

With the complex nature of molecular dynamics exploring new regions, identifying when to best use machine learning, is a complicated task. Prior work has often required many iterations to find the best machine learning usage parameters, often relying on fixed usage thresholds or number of time-steps. Combining advances in understanding MD ensemble uncertainty and MD deep learning, our improvements to Proxima have enabled the dynamic usage of deep learning models in molecular dynamics simulations. This advanced version of

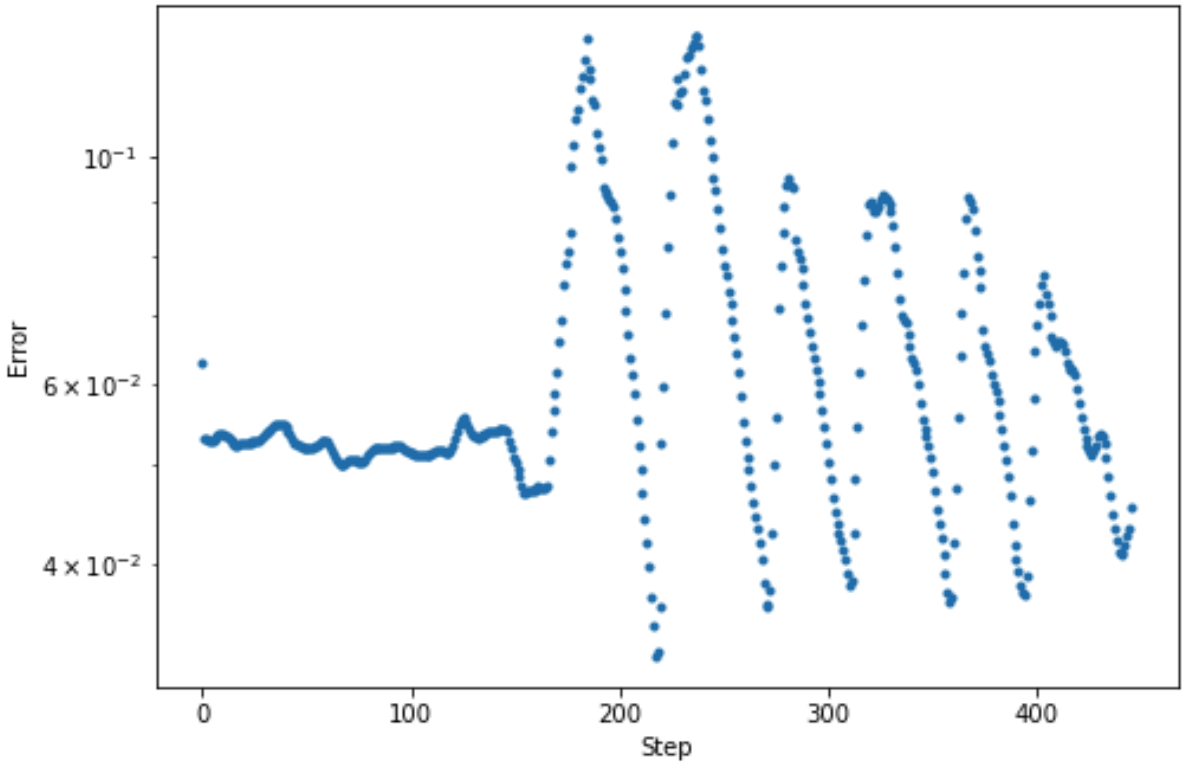


Figure 1.15: This graph presents the force error against the steps in the simulation.

Proxima, no longer requires multiple iterations of the application to find those best fixed usage boundaries. Furthermore, we demonstrated the ensemble method standard deviation relationship with force error as as a suitable method for on-the-fly surrogate integration. Proxima allows for a surrogate modeling framework that can be introduced into an application using a single function decorator and user error tolerance.

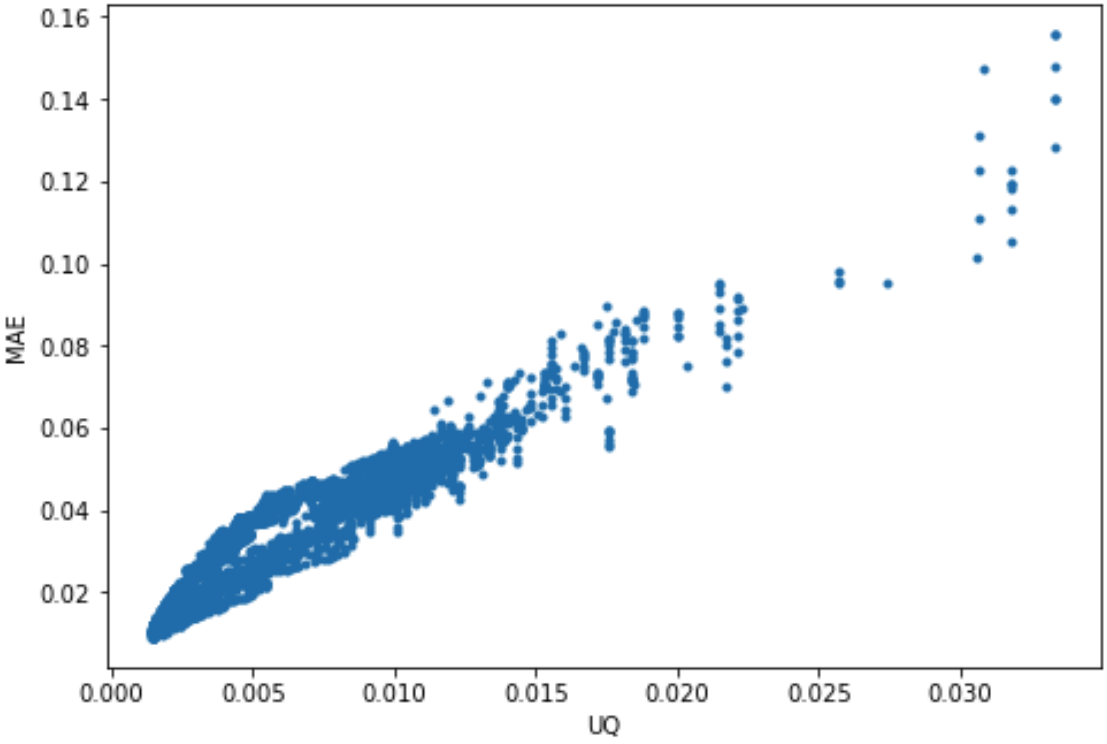


Figure 1.16: This graph presents relationship between UQ and Error.

CHAPTER 2

PERFORMANCE PREDICTION

2.0.1 Introduction

With computing architectures constantly evolving and diverging from the general-purpose CPU, the nature of understanding said architectures has only grown more complicated. These new architectures include, among many others, graphical processing units (GPUs), neuromorphic computing chips, tensor processing units (TPUs), and field-programmable arrays (FPGAs). The increase interest has been motivated in part by a growing popularity of deep learning methodologies within both industry and scientific research [16, 70, 31, 91, 117, 123, 125]. Understanding how to realize performance gains among these architectures can be a laborious process. The common direct simulations of computing architectures involve a clear trade-off between accuracy and performance, with cycle-accurate simulations often considered the gold standard in performance projection, but prohibitive for most real-world applications [90, 92, 13, 88]. Consequently, performance projection across architectures, has long been considered a difficult task.

Given this clear motivation to understand future performance, a large body of work has already explored application performance projection [6, 22, 121, 79, 83, 60, 63, 26, 72, 69, 9, 105, 106, 62]. Ipek et al. [63] and Carrington et al. [26] each looked at a single large applications, while Konstantinidis and Cotronis [69] and others studied simpler kernels such as DAXPY, DGEMM, FFT, and stencil kernels. There is also work on application projection within the same architecture [86, 63]. Prior work [26, 27, 9], also requires significant expert input, such as awareness and the ability to pinpoint and extract the most compute-intensive kernels in the application or reliance on a lab generated private profiling tool [83, 6, 10]. Additionally, application performance projection has been used for the benefit of improving efficiency in power consumption, grid job scheduling, and resource management [122, 71,

44, 62]. Prior work requires code changes and/or complex programming models, or time-consuming cycle accurate simulators to understand performance.

With the limited focus on data-driven methods, we believe there to be room for better projection accuracy. The limited access to these high-performance computing architectures along with the lack of publicly available performance data has constrained the data-driven approaches among GPU to GPU performance projections. With the increased adoption of GPU acceleration in the scientific and machine learning community, it is only becoming more vital to understand future performance among these type of architectures.

To the best of our knowledge, previous work, while valuable, has not targeted data-driven performance projection across modern architectures (e.g. GPU to GPU). Due to the scarcity of data-driven performance projection in the past, previous work has not contributed large usable datasets to the public. To address this, we make three novel contributions: a large dataset and two case studies of performance projections.

The dataset generated for this work was collected over a 6-month period on super computers at Argonne National Laboratory. This dataset will be made publicly available to help accelerate future research in data-driven performance projection. In this study, we use the Rodinia Benchmark in addition to the commonly used STREAM benchmark [28, 82]. With over 30,000 runs across two GPU architectures (NVIDIA P100 and V100), we used current state-of-the-art profiling tools (NVIDIA NVProf), to collect over 100+ low-level metrics. This produced a collection of over 3.6 million data points of performance metrics, that, to the best our knowledge, is the largest cross-architecture GPU metric dataset available.

The first case study on using this large dataset was focused on the projection of IPC performance. In this and the following case, we consider the P100 as the base architecture, and V100 as the target architecture. This decision was made because the V100 is the newer architecture in comparison to P100. Here, we were able to explore many points in design, such as any specific features that may matter in IPC projection, varying common machine learning

techniques, and using more advanced learning techniques in order to increase accuracy. With the large curated dataset, we were able to explore the use of the neural architecture search library (DeepHyper - used in cancer research), developed by a team at Argonne National Laboratory [8]. We tested over 1.4 million models, using over 1,200 core hours, which resulted in a mean absolute percentage error of around 12%, using 70% of the data to train the model.

For our second case study, we look at another popular performance metric: memory throughput. Being that a large amount of scientific applications are memory bound, understanding memory throughput on target architectures can give insights on potential optimizations of the application and limitations of the target architecture. More recently, memory throughput is being used as the main metric in understanding performance portability in terms of architecture efficiency [52]. Similar to case 1, we start by using standard machine learning techniques and progress in complexity to increase accuracy. We also identified nine important features that carry the most impact on accuracy. We achieved a mean absolute error of 15.82 Gb/s, where the memory throughput can range from 0.00067 Gb/s to 812.89 Gb/s. We also show how application developers can use the results of our analysis for better resource management. For example, using machine learning classification, we can identify when a workload changes from non-memory bound to memory bound as the application is ported from one architecture to another architecture.

Our contributions in this work are as follows:

- We describe how the community can leverage our datasets to perform various data-driven generic cross-architecture analysis.
- We give three more case studies on potential insights that others can do with our dataset.
- We empower data science in the field of computer architecture.

2.0.2 Related Work

With such strong motivation behind performance prediction, there is much prior work. From single intra-architecture performance prediction [85, 87, 7, 124, 38] and inter-architecture predictions [6, 121, 79, 26]. Ardalani et al. [6] want to understand the benefit of estimating GPU performance prior to writing a specific GPU implementation of their current CPU implementation. Here, the authors look at the potential benefits of a GPU implementation by looking at corresponding counterparts of the current CPU implementation. They note that since CPU programming is much easier than GPU programming, programmers can implement different algorithms for the CPU and use the CPU-based GPU performance prediction to get speed-up estimations for the different developed algorithms. This tool will then be able to help the developers into choosing and porting the correct algorithm. Specifically, they were able to use program properties, features inherent to the program or algorithm and independent of what hardware the program runs, to create a mapping between these features to GPU execution time. The tool built predicts GPU execution time for CPU code prior to developing the GPU code. Our work looks at a specific metric, IPC, and whether the application in question will become memory bound to obtain an understanding of whether the application is worth porting over to a new architecture. Their final dataset consists of 122 datapoints which was used to test and train 100 different ensemble models achieving an average relative error of 22.4%.

Similar to Ardalani et al., Boyer et al. [22] created a modeling framework, built on top of GROPHECY, that not only predicts kernel execution time, but data transfer time to represent the total execution time of a GPU application. They extended a GPU performance model to include a data usage analyzer for a sequence of GPU kernels, to determine the amount of data that needs to be transferred, and a performance model of the PCIe bus, to determine how long the data transfer will take.

Yang et al. [121] looks at relative performance between two platforms while only needing

to observe short partial executions of two ASCI Purple applications. Their method targets performance predictions in guidance for resource allocation. The partial executions require an API where the user must understand where the repetitive phases occur to understand execution behavior across the entire application without the need for full execution. The predictions and evaluations were done across CPUs only and partial execution on the target is required in order to extrapolate and predict whole application performance. Unlike this approach, our work does not require the user to understand the specific partial executions needed to run to implement the workflow.

Marin et al. [79] created a toolkit that semi-automatically measures and models static and dynamic characteristics of applications using the application binaries to predict the L1, L2, TLB cache miss counts, and execution time. They describe a methodology as a function of how the application exercises the memory subsystem, for constructing models of an application's characteristics parameterized by problem size or other input parameters. Though Marin et al. created a architecture-neutral models, our work doesn't require the developer to work on application binaries nor go through a complex workflow to create an initial characterization of the application.

Meng et al. [83] created a GPU performance projection framework, used by Boyer et al., that estimates the performance benefit of using a GPU without requiring GPU programming, but by providing pieces of CPU code that targets for GPU acceleration. The authors defined CPU code skeletons, automated a mechanism to restructure CPU code skeleton and mimic transformations need to tune GPU code, characterized the benefits and side effects of GPU code transformations, projected a CPU kernel's performance on GPUs without producing GPU code. The authors also allowed the ability to explore future GPU generations and evaluate their performance by varying GPU hardware specifications. The developed code skeletons are transformed to mimic tuned GPU codes where the cost and benefit of GPU development can then be estimated according to this transformed skeleton. Our workflow

would not require converting the application into skeleton code and instead would require the user to profile the application on the current architecture using NVIDIA's NVprof profiling tool to gain the applications characterization.

Hong et al. [60] created a power and performance prediction model (IPP) that predicts the optimal number of active processors for a given application. IPP, takes a GPU kernel as input and predicts both power consumption and performance together. Using these power consumption and performance outcomes, IPP predicts the optimal number of cores that result in the highest performance per watt. Their results show that by using fewer cores based on the IPP prediction, they would be able to save up to 22,09% of runtime energy consumption for the five memory bandwidth-limited benchmarks. In particular, this work characterizes performance prediction in terms of power modeling for the GPU.

Ipek et al. [26] created an easy to use model using one parallel application, SMG2000, to predict performance across two platforms. Similar to our work, they employed a multilayer neural network trained on input data from executions on the target platform capturing full system complexity achieving 5%-7% error across a large, multi dimensional parameter space. In our work, we expand the search beyond one application, to multiple applications creating a complex training set that is not application specific.

Carrington et al. [26] also presented a performance modeling framework, developed by the Performance Modeling and Characterization (PMaC) Lab that is faster than traditional cycle-accurate simulation and shown to be effective on the LINPACK benchmark. The framework is not design for a specific application or architecture. To do so, they developed a benchmark probe tool for collecting machine profiles (Memory Access Patter Signature) and a tool for gathering application signatures (MetaSim Tracer). They rely on a convolution method that maps this application signature onto a machine profile. Their MetaSim Tracer processes all of the memory addresses of an application on-the-fly. Similar to the Carrington et al. work, the returned NVProf performance metrics used in this work, characterizes the

application's memory and bandwidth utilization on the architecture and is what we use as the input features for our models. Carrington et al. is also predicting execution time and not IPC or whether the application becomes compute bound or not.

Carrington et al. [27] furthered development of their framework to include blind predictions for three systems as well as establishing sensitivity studies to advance understanding of observed and anticipated performance of bother architecture and application. Here, the defined that the Machine Profile is measurements of the rates at which a machine can perform basic operations, including message passing, memory loads and stores, and floating-point operations which is collected via low level benchmarks and probes. They specifically look at performance on two applications, Cobalt60 and HYCOM.

Lee et al. [72] looked deeply into parameter space characterization for highly parameterized parallel applications. They construct and compare two classes of effective predictive models: piecewise polynomial regression and artificial neural networks. They look at performance prediction of Semicoarsening Multigrid (SMG2000) and High-Performance Linpack(HPL). Here, a single neural network is developed and was tested using only 100 validation points. We tested over a million neural networks which were each tested with validation sets that had between 400 to thousands of validation points. As noted in the paper, they observed a non-monotonic trend when adding data to their training set illustrating the difficulty of identifying an optimal sample size, which is something we encountered as well.

Konstantinidis et al. proposed a performance prediction of GPU kernels on 4 different computation kernels: DAXPY, DGEMM, FFT and stencil kernels achieving an absolute error in predictions of 10.1% in the average case and 25.8% in the worse case [69]. To achieve realistic results the authors applied three adjustments to the theoretical peak device performance. The adjustments are on the compute and memory transfer peaks and the compute peak of a particular kernel.

Balaprakash et al. [9] present an iterative parallel algorithm that builds surrogate perfor-

mance models for scientific kernels and workloads on single-core, multi-core, and multi-node architectures. They developed ab-dynaTree, a dynamic tree model obtained through updates of the first in previous iterations which is then used to choose the next inputs for training. In [10], Balaprakash et al. use their previously developed active learning model, ab-dynaTree to obtain surrogate models for GPU kernels when concurrent evaluations are not possible due to the lack of availability of a GPU cluster. Here, they present an active learning approach for obtaining surrogate models for GPU kernels and an empirical study spanning a diverse set of GPU kernels and graphic cards. In line with our work, Balaprakash built these surrogate models to minimize execution times of benchmark problems.

From predicting execution time, specific metric predictions, and CPU to GPU mapping, prior work has made many advances in application performance prediction. That said, the work mentioned often requires expert input,lab specialized profiling tools, and/or pinpointing specific compute heavy kernels in the application. Additionally, prior results achieving good performance, error less than 5%, look at only 1-2 similar applications. Finding an automatically created generalized model without expert input can have significant benefits. A generalized model has an input of any application, from simple computations to machine learning models, and predict performance on a target hardware.

2.0.3 Background

In this section we provide background information on the following aspects of the work that we present in subsequent chapters: the two GPU architectures considered in this work-Nvidia P100 and V100, the main metric we are predicting - IPC, and modeling methods used - random forest, deep learning, neural architecture search, and active learning.

2.0.3.1 GPU Architectures: NVIDIA P100 and V100

The two architectures we use in this work are NVIDIA's P100 and V100 provided by Argonne's Leadership computing facility. In all cases presented, we use P100 as the architecture we are predicting from, and the V100 as the architecture we predicting to. The P100 metrics are used as features in the training set. The V100 IPC are used as the target value in the case of IPC prediction. For memory bound prediction, a labeling is created for each datapoint regarding whether they are considered memory bound on the corresponding architecture.

Though at face value, NVIDIA's P100 and V100 may not look that different, especially compared to older models (going from K40 to V100 has a 200% increase in computation capability whereas going from P100 to V100 only 16%), we can see substantial differences when comparing performance across these two architectures. Table 2.1 shows the architectural details of present and past NVIDIA GPUs. The last two columns, P100 and V100, are similar although showing an increase in the number of streaming multiprocessors (SMs) and memory cache for the V100. However, Figure 2.1 illustrates that there is no function that can map a P100 IPC value to a V100 IPC value, because one P100 IPC value can map to multiple V100 IPC values. Therefore, we are not looking a 1-1 mapping between one metric to another, but have an application characterization of multiple metrics. Though they are only one generation apart, the P100 and V100 have fundamental differences that can be seen when executing and profiling a variety of applications between them. The V100 was developed with a core purpose of creating better performance for AI applications, though it is not always the case, as shown here and by other benchmarks [80].

In this thesis, we use NVIDIA's NVprof to profile all applications, acquiring around 120 metrics, depending on the architecture, to give a detailed overview of an application's performance on the GPU architecture. In this work, we look at 116 shared metrics between the two GPU architectures, listed in the Appendix, as the P100 did not have the NVLink metrics the V100 has. Metrics such as *dram_read_throughput*, *dram_write_throughput*, *sin-*

gle_precision_fu_utilization, and *flop_count_dp*, along with 112 other metrics are used as features to predict IPC on the given architecture. The total metrics used are reduced to 116 as P100 does not have NVLINK performance metrics in the profiling result. Similar to how Carrington et al. created and collected machine profiles, these metrics give an overview of the applications performance. Unlike Carrington et al., these profiles are mapped to the specific architecture the application was run on.

Table 2.1: Key specifications of selected GPUs of different generations. Computation capability is considered numerical label for the hardware version.

	Kepler (K40)	**Maxwell (M60)**	**Pascal (P100)**	**Volta (V100)**
CPU	IBM Power8 @2.2GHz	Intel Xeon E5-2670 @2.60 GHZ	IBM Power8 @2.2GHz	Intel Xeon E5-2699 @2.20GHz
Computation capability	3.5	5.2	6.0	7.0
SMs	15	16	56	80
Cores/SM	192 SP cores/64 DP cores	128 cores	64 cores	64 SP cores/32 DP cores
Texture Units/SM	16	8	4	4
Register File Size/SM	256 KB	256 KB	256 KB	256 KB
L1 Cache/SM	Combined 64K L1+Shared	Combined 24KB	Combined 24 KB	128 KB Unified
Texture Cache	48KB			
Shared Memory/SM	Combined 64K L1+Shared	96 KB	64 KB	
L2 Cache	1536 KB	2048 KB	4096 KB	6144KB
Constant Memory	64 KB	64 KB	64 KB	64 KB
Global Memory	12 GB	8 GB	16 GB	16 GB

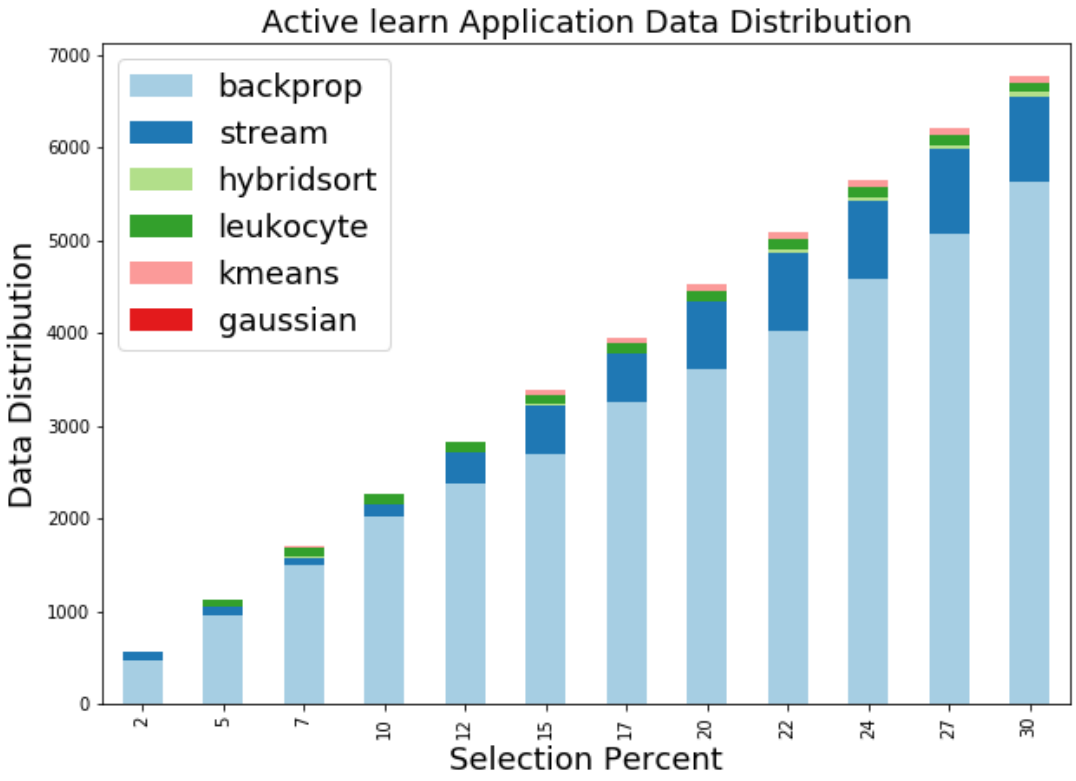


Figure 2.1: Comparison of P100 and V100 IPC. There does not exist a mapping from P100 IPC to V100 IPC.

2.0.3.2 Architecture Performance: IPC

Instruction per cycle (IPC) is often used as a simple metric of performance in the development of CPUs. IPC is a good indicator on whether an architecture is optimally performing. A high IPC is not always indicative of a more efficient architecture, but is a start at recognizing the potential of an architecture with a simple metric. Additionally, because of the complexities and differences between CPU and GPU's, IPC is an easily available metric that can be traced across these two differing chip architectures. In particular, NVIDIA defines the IPC value as the instruction throughput over a period of collection. As we hope this framework will eventually extend to cross-chip (GPU to CPU) predictions, we chose a metric can be carried from one architecture to another [2].

2.0.3.3 Active Learning

The key idea behind active learning is a machine learning algorithm that can perform better with fewer labeled training instances if it is allowed to choose the data from which it learns [103, 30, 73]. In our work, we employ the an active learning system due to the fact that we will have much more unlabeled data (P100 performance metrics), than labeled data (V100 performance metrics). Additionally, Settles notes that labeled instances are difficult, time-consuming, or expensive to obtain. In our case, there was only one V100 GPU with a long queue time, while collecting performance metrics of certain larger applications taking longer than the allotted time cap. Some of examples of the successful use of active learning are in speech recognition [49, 95], information extraction [110, 65], classification and filtering [126, 4]. All examples show how active learning systems overcame the labeling bottleneck querying in the form of unlabeled instances to be labeled by an *oracle*. With this querying scenario, active learning aims to achieve high accuracy using as few labeled instances as possible and as a result, minimizing the cost of obtaining labeled data.

Active learning has several scenarios in which is may pose queries. In this thesis, we will

focus on *pool-based* active learning cycle. Pool-based scenarios has been studied for many machine learning problems [73, 81, 108, 66]. In this active learning scenario, queries are selected from a large pool of unlabeled instances. An uncertainty sampling query strategy where the where the instances in the pool that the model is least certain how to label are chosen. The learner can begin with a small number of instances in the labeled training set and request labels for low confident instances learned from the querying results. These requested labels are added to the labeled set, and the learner proceeds from there in a standard supervised way.

There are different querying frameworks when using active learning: uncertainty sampling, query-by-committee, and expected model change to name a few. We use uncertainty sampling as our querying framework. Uncertainty sampling is a commonly used query framework where the active learner queries instances about which it is least certain how to label [74, 35]. The unlabeled instances will be ranked based on uncertainty and the learner queries the top most uncertain instance to have them labeled. Uncertainty sampling in regression, and used in this thesis, is the calculation of the sampling variance of the random forest. Wager, et al. developed this method of estimating the variance of bagged predictors and random forests. This variance estimation tells whether the random forest is more confident about certain predictions compared to others [115].

Though there are many potential benefits to employing an active learner, we should note that there are some caveats with active learning. First, the learner measures based on a single hypothesis. The training set returned is very small and as a result introduces potential sampling bias.

2.0.3.4 DeepHyper and Balsam for Neural Architecture Search

Neural architecture search (NAS) is the process of automating architecture design for a machine learning model [127]. We employ this powerful process to search across over a million

machine learning models predicting performance prediction. Zoph et al. present a Neural Architecture Search, that uses a recurrent network to generate the model descriptions of neural networks. They train the RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. Elsken et al. categorize methods for NAS according to three dimensions: search space, search strategy, and performance estimation strategy. Figure 2.2 shows an illustration of the Neural Architecture Search method often referred to and something employed by the version of NAS in DeepHyper. Specifically, the search space defines which architectures can be represented in principle. This stage can still introduce human bias which can in return prevent finding novel architectural building blocks. The search strategy details the exploration of the search space. The Performance Estimation Strategy refers to the process of estimating the predictive performance on resulting NAS architectures on unseen data [39].

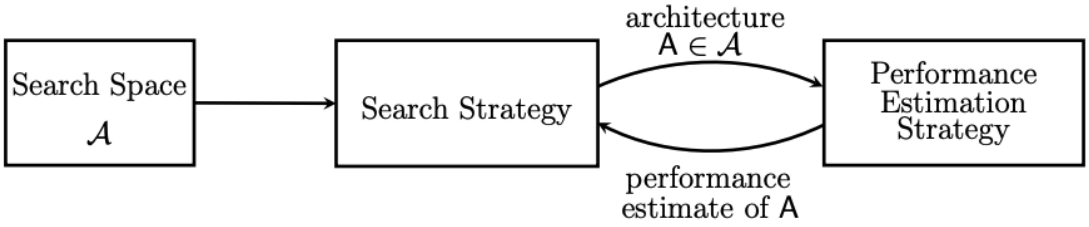


Figure 2.2: Illustration of abstraction presentation of Neural Architecture Search methods by Elsken et al. The search strategy, similar to the search strategy we use, selects an architecture $\mathbf{A}$ from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of $\mathbf{A}$ to the search strategy [39].

DeepHyper and Balsam frameworks developed at Argonne National Laboratory, are vital in conducting a scaled search over millions of deep learning models [11] [99]. DeepHyper is a scalable framework designed to search the hyper-parameter space of deep neural-network models. DeepHyper also includes an integrated neural-architecture search (NAS) mechanism, which enables the automated generation and testing of neural-network models. DeepHyper is tightly coupled with Balsam, a workflow manager that uses a persistent job/task database to

efficiently utilize leadership-scale distributed supercomputing resources. Balsam dynamically packages tasks into *ensemble jobs* and manages the end-to-end scheduling life cycle, ensuring fault tolerance along the way. Additionally, Balsam allows for a complex multi-workflow solution with little user configuration.

As validated using a class of representative cancer data [8], DeepHyper-NAS automates the generation of deep-learning models using a reinforcement-learning. To execute an NAS workflow, DeepHyper results are used to dispatch reward-estimation tasks, based on R^2, to Balsam. After the architecture search is completed, there are between 15,000 to 30,000 distinct models generated, trained and tested. From this large pool of models, the estimated reward values are used to select the top 50 DNN architectures. The top 50 DNN architectures are then submitted to a post-training sequence, during which each model is thoroughly trained on the full training data.

In this work, we utilize this Deephyper-based NAS workflow to predict performance metrics on NVIDIA GPU architectures. The use of this model-generation pipeline allowed us to test more than one million neural architecture models.

2.0.4 Methodology

The following sections will discuss both intra- and inter-architecture performance prediction, as well as the benchmark data used to train and validate the models. For the intra-architecture case, we consider the prediction of application performance (IPC) on a P100 GPU, given profiling metrics from a P100. Our results for intra-architecture prediction confirm that IPC can be accurately predicted, given either complete or partial profiling metrics. For the inter-architecture case, we consider the prediction of application performance (IPC) on a V100 GPU, given profiling metrics from a P100 GPU. We also explore the inter-architecture classification of specific application runs as either *memory bound* or *not memory bound*. The memory-bound classification task does not result in a numerical performance

prediction, but ideally captures a similar relationship between the alternative architectures. For inter-architecture IPC prediction, we compare various methodologies, including random forest, deep learning, and NAS.

2.0.4.1 Benchmark Data

Metrics are collected by Nvidia's NVProf profiling tool. NVProf instruments the CUDA kernels, and collects a variety of useful performance metrics. We acquire 116 NVProf performance metrics for each of the 46,039 application runs (see Appendix for metric list). The target applications include *backprop*, *hybridsort*, *kmeans*, *stream*, *gaussian*, and *leukocyte*; all except *stream* come from the Rodinia benchmark suite [28].

Backprop, containing two kernels, is a deep learning algorithm used in the training of feedforward neural networks for supervised learning. *Hybridsort*, containing seven kernels, is a parallel bucketsort that splits the list into enough sublists to be sorted in parallel using mergesort. *Kmeans*, containing two kernels, is a clustering algorithm that divides an initial cluster of data objects into K sub-clusters. *Kmeans* represents data by the mean values or centroids of their respective sub-clusters. Iterations of the algorithm compare the data object with its nearest center, based on a distance metric [28]. *Srad*, containing eight kernels, or Speckle Reducing anisotropic diffusion is a diffusion method for ultrasonic and radar imaging applications based on partial different equations [109]. *Leukocyte*, containing three kernels, is an application that detects and tacks rolling white cells [28]. *Stream*, containing five kernels, is a benchmark designed to measure sustainable memory bandwidth for contiguous, long vector memory accesses [82].

2.0.4.2 Intra-Architecture IPC Prediction

As seen in prior work, it is possible to predict performance metrics within the same architecture using a variety of techniques and tools. That being said, we acknowledged that there

is no function that can map P100 IPC directly to V100 IPC, as shown in Figure 2.1, and thus we use the fuller scope of P100 metrics to predict V100 IPC. Here we present a deep learning method that allows for IPC prediction of applications run on the P100 architecture. We consider this step as a proof of concept that a certain scope of metrics can predict IPC.

For our intra-architecture prediction, we use a feed-forward fully-connected deep-learning model that has weights initialized with a normal distribution. The model has 1 input layer and 2 hidden layers, with all layers (excluding the output layer) using ReLU activation functions. We use ReLU because it shows better convergence results compared to using other activation functions. The deep learning model uses the Adam optimizer, along with a mean squared error loss function. The Adam algorithm calculates an exponential moving average of the gradient and the squared gradient [68]. The model is trained for 100 epochs.

Furthermore, to stress the relationship between metrics and IPC, we reduced the input metrics from 112 to 5. The following definitions for the reduced metrics are provided by CUDAs Toolkit Documentation [1]. The reduced metrics include *shared_utilization, stall_other, single_precision_fu_utilization, dram_read_throughput,* and *dram_write_throughput. shared_utilization* is the shared memory relative to peak memory utilization. *stall_other* is the percentage of stalls occuring due to miscellaneous reasons. *single_precision_fu_utilization* is the utilization level of the multiprocessor function units that execute single-precision floating-point instructions on a scale of 0 to 10. *dram_read_throughput* is the device memory read throughput. *dram_write_throughput* is the device memory write throughput.

2.0.4.3 Inter-architecture Memory Bound Classification

We first look at cross-architecture memory-bound prediction. Similar to the results of comparing IPC between V100 and P100, Figure 2.3 shows that there is no linear function that can map between V100 dram write, read, and total utilization to P100 dram write, read, and total utilization. In Figure 2.5, you can also see the differences in dram utilization be-

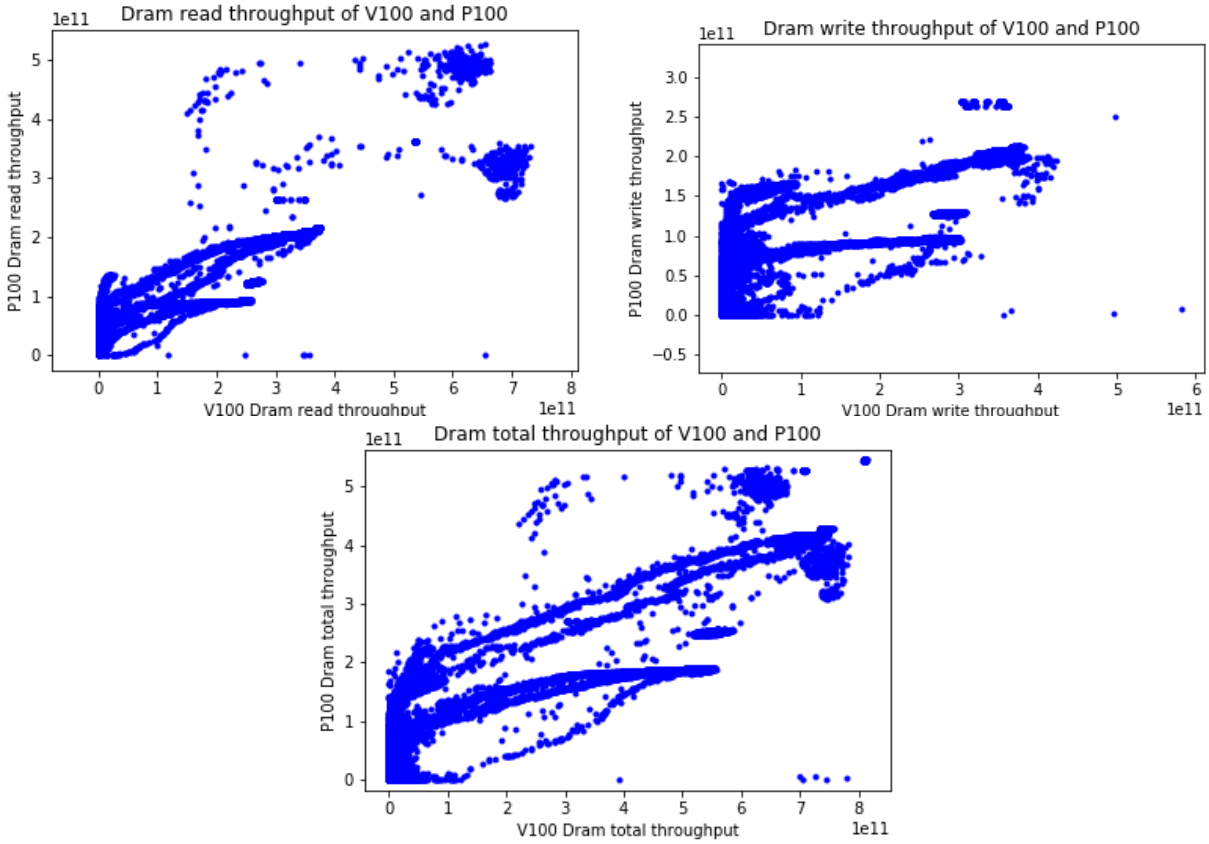


Figure 2.3: Illustration of DRAM read and write throughput and total dram throughput on the V100 and P100. We see that in all three cases, there is no function that can map DRAM read throughput, write throughput, or total utilization from one architecture to another.

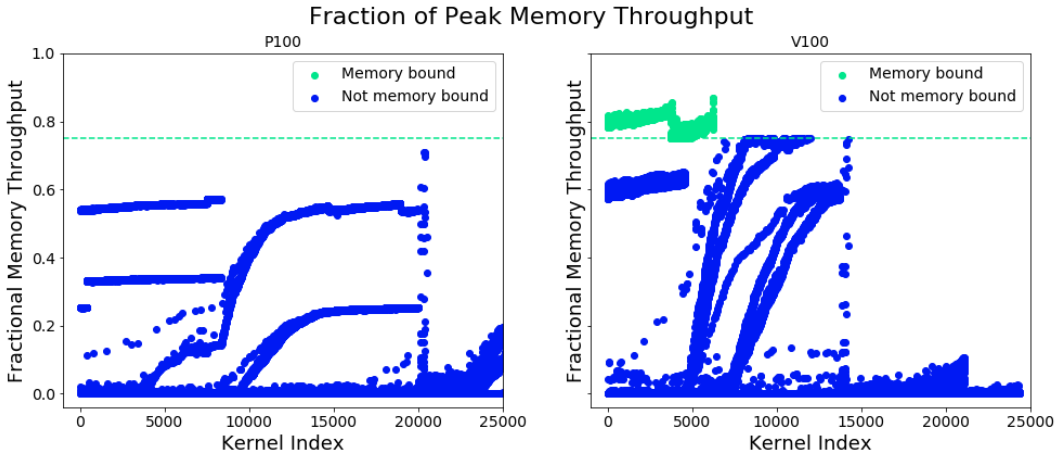


Figure 2.4: Illustration of memory throughput for Nvidia P100 and V100. Points above the green line are considered memory bound kernels, or having over 75% of memory bandwidth utiization. In comparison, the same applications run on the V100 show kernels becoming memory bound on the V100 that were not memory bound on the P100.

tween the two architectures in both scaled and un-scaled versions. There is no simple linear function that would map all P100 dram values to V100 dram values. In addition to the P100 data points collected for the intra-architecture prediction step, 32,291 V100 data points are collected using the NVProf profiling tool. Only P100 data points that had corresponding V100 data points were used. We explore whether an application was not memory bound on P100, but is on V100. Applications that require processing large amounts of data, such as multiplying large matrices, would likely be memory bound. In Figure 2.6, memory bound applications on both P100 and V100 NVIDIA architecture are plotted along their IPC value. Here, IPC results are more spread out with a high DRAM total throughput on the V100 compared to the P100.

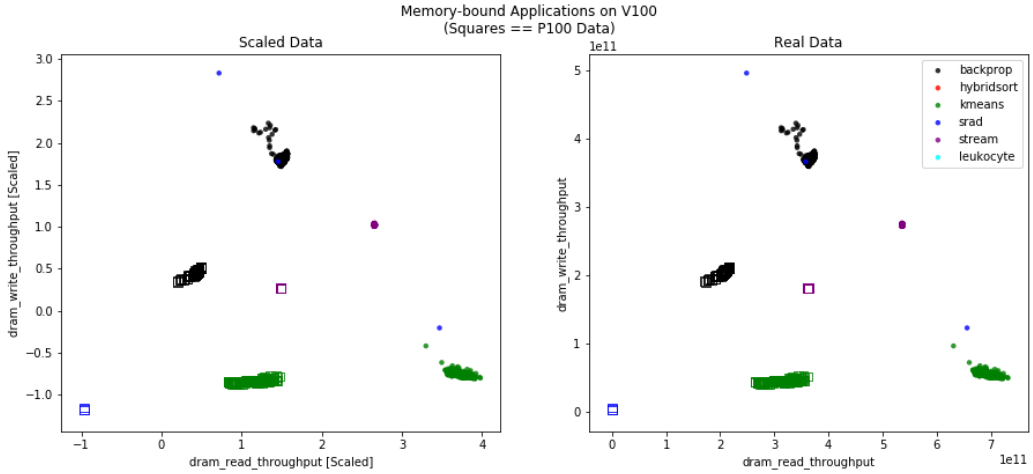


Figure 2.5: Dram read and write utilization on both P100 and V100 GPUs

By NVIDIA architecture standards, an application becomes memory bound on their architecture is an application having a dram utilization of over 75% on the architecture. As shown in Figure 2.4, there are kernels that become memory bound on the V100 that are not memory bound on the P100. In order to label our dataset, we calculate the ratio of the total DRAM throughput with that of the theoretical memory maximum. Total DRAM throughput is calculated by adding *dram_read_throughput* and *dram_write_throughput*. We consider all data points with ratios greater than 0.75 to be memory bound. Using a random

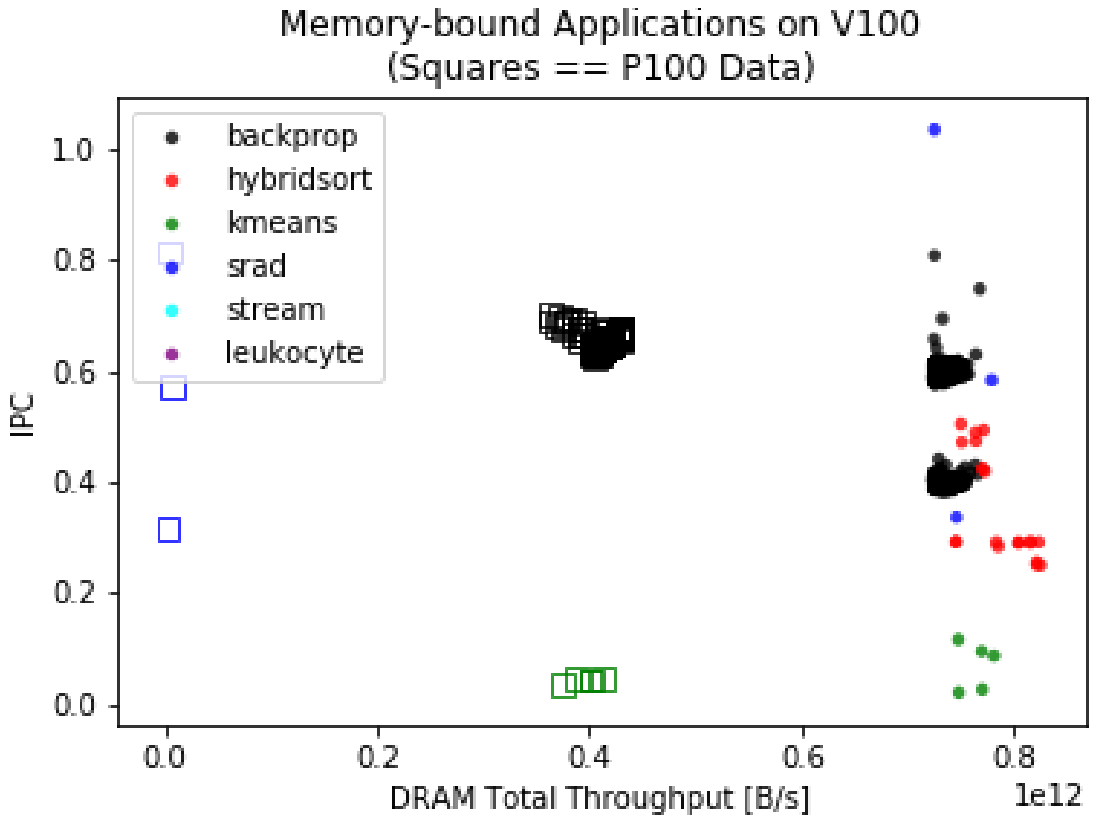


Figure 2.6: Memory bound applications vs. IPC of application on both P100 and V100 architectures

forest classifier, we were able to predict whether an application run on a P100 would become memory bound on a V100 (given P100 profiling data). We optimize the hyperparameters of the random-forest classifier using a grid search. The classifier is trained on a set of P100 metrics with the applications corresponding to V100 target values. It is then tested on the validation and test sets to confirm that the model is not over-fitted and performs well on untested data.

2.0.4.4 Inter-Architecture IPC Prediction Methodology

Here, we present the three modeling frameworks tested for IPC prediction. Due to the limited data size available of performance metrics, we employed the use of an active learner to identify if there were application points that best characterized applications performance on the architecture. Essentially, with the use of active learning, we want to avoid collecting more data than necessary. The total data set (32,291 points) is split into a training (22,603) and hold out set (9,688). We used Argonne's DeepHyper framework to test over a million deep learning models. We used a random forest model and conventionally developed deep learning model as a baseline comparison. All three modeling methodologies are tested with an active learner queried dataset and a training set created by random selection.

2.0.4.5 Curating Training Sets: Active Learning and Random Selection

In real world applications, gathering data can be quite time consuming. This, combined with the fact that new computing architectures are often scarce, means that labeled datasets are likely to be relatively small in practice. We will also not be training to the entire dataset and therefore, using active learning to create refined datasets. We do this for a range of different refined dataset sizes.

First, an initial base set is chosen. Though there are many forms of creating the base set, the original training set batch (250 points) are points randomly chosen from the data

set. We use a random forest as the supervised machine learning model in our active learning strategy. We use additions with batch size of 250 data points during each training cycle. 12 different training sets are created by taking sizes of 2.5 to 30 percent in increments of 2.5, from the training dataset. Therefore, the active learning training cycle is run 12 times, once for each size, creating 12 training sets with corresponding sizes.

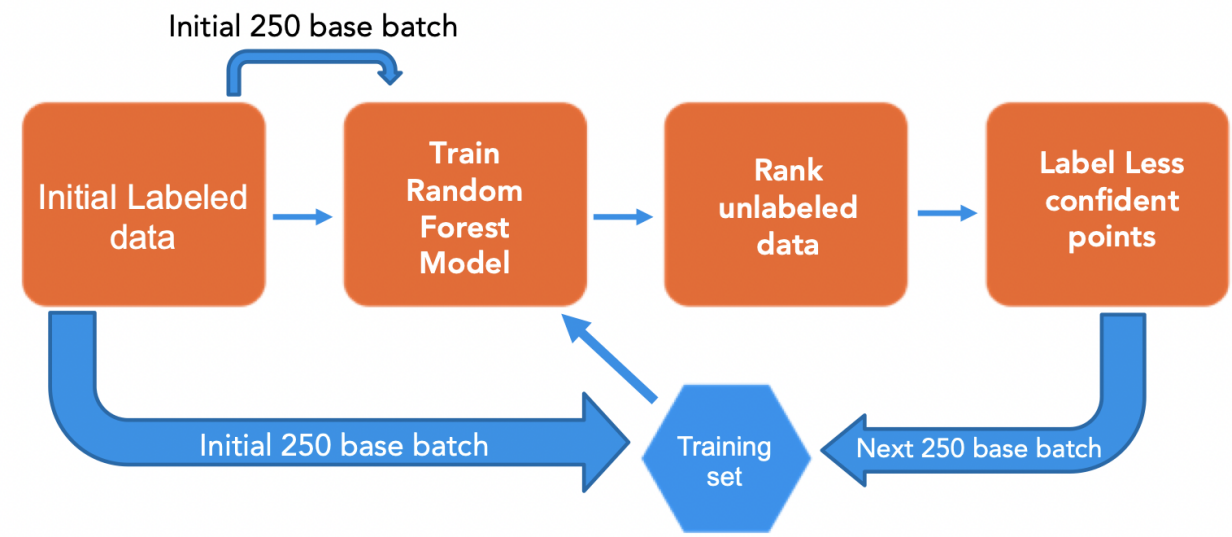


Figure 2.7: Active learning flow chart showing the first batch of 250 points being trained with random forest and the cycle of querying new data points and adding them to the training set. The cycle terminates once the target training set is reached.

We use a pool-based active learning strategy, where the active learner can query from a pool of data points. Figure 2.7 shows the active learning workflow of creating a queried training set. First, the supervised learning model, we use a random forest model, is initially trained with the base set and predicts on the unlabeled points. The random forest is trained on 250 datapoints with 116 P100 GPU architecture performance metrics as features, to predict IPC (target value) of the given application run on a V100 GPU architecture (inter-architecture IPC prediction). Next, the unlabeled points are then ordered from highest to lowest based on the model's confidence of the predicted unlabeled points. We use scikit-learn's implementation of Wager's uncertainty definition described here [115] to rank the

unlabeled data. The batch of points with the highest uncertainty are chosen. These points are considered valuable as the model has low confidence in its accurate prediction of them, and thus these points will be added to the training set. The model is retrained with the newly formed training set (base set and new points).

One full round of active learning can be seen in Figure 2.8, where the blue dots are the most uncertain points and the red points are the predicted points on the unlabeled data set. After the learning cycle is completed, the points are once again ranked dependent on their uncertainty score, and a new batch of points are added to the initial training set. After each addition to the training set, the model is retrained and the uncertainty is recalculated until the specified training size and learning cycles are completed. These specified 12 training sets are used in all models, as show in 2.11.

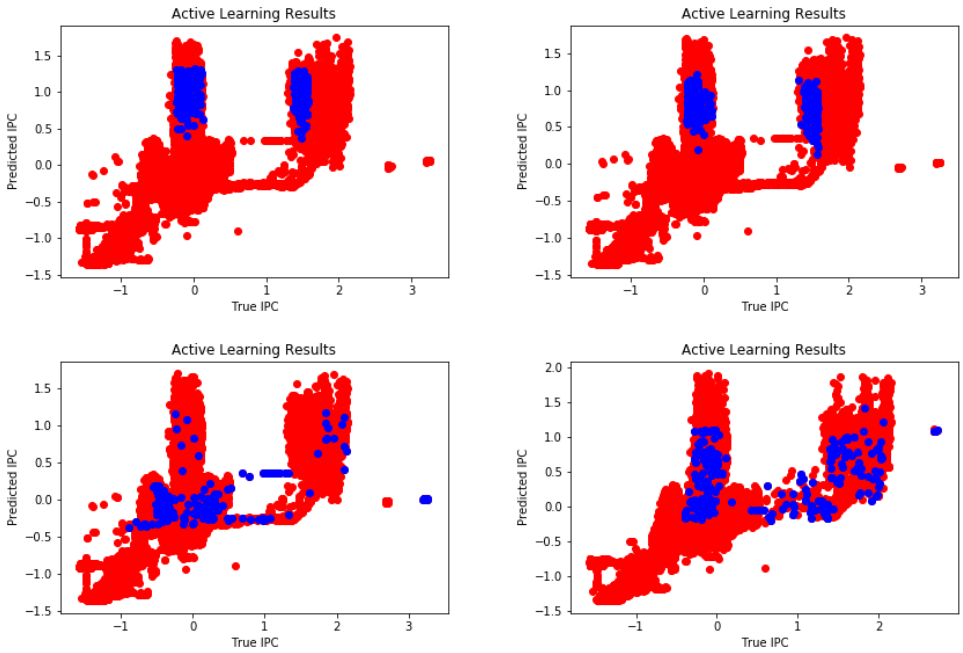


Figure 2.8: Results of one round of active learning. Blue points are the points chosen by the active learner and put into the training set. Red points are the predicted points on the unlabeled data set.

As a baseline for comparison, equivalent sized training sets are created with randomly chosen data points from the same training set the active learner has access to. Additionally,

the random forest model, conventional deep learning model, and DeepHyper are all trained on a the full training dataset, 70 percent of the full dataset (22,603 data points).

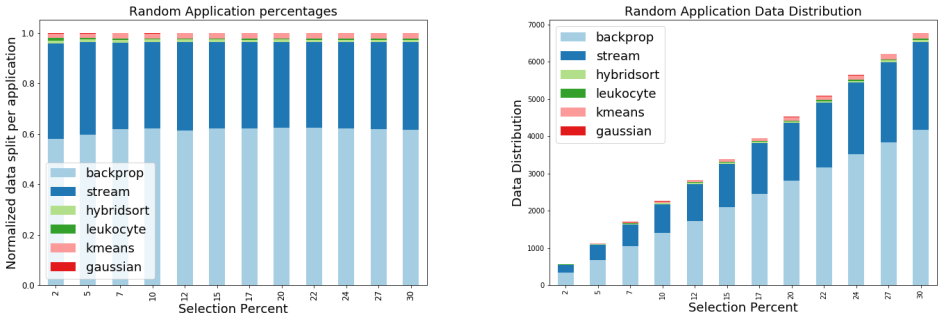


Figure 2.9: Normalized application percentage breakdown of data points that were chosen at random.

The data distribution and percentage breakdown of the application points selected by random and active learner selection can be seen in Figures 2.9 and 2.10, respectively. In particular, the random selection process gives a more stratified overlook of the total application data. The active learning selection process, slowly increases the amount of points from other applications, as the training set gets bigger. In particular, the active learning workflow shows a significant focus on the more difficult to predict backprop application. The continuous addition of the backprop data points can be understood by seeing that the backprop application obtains the lowest confidence when predicted. Additionally, the quantity of the backprop data dominates the data set sampling insuring a high amount of backprop application specific points will be ranked high.

2.0.4.6 Neural Architecture Search at Scale

Following preliminary success with DeepHyper-based NAS, we further scale the initial search procedure to generate specialized architectures for each of the fine-tuned active-learning datasets. Altogether, we consider 24 training sets, of which half are random and the rest are curated with active learning. The NAS framework utilizes Balsam to test and train a large

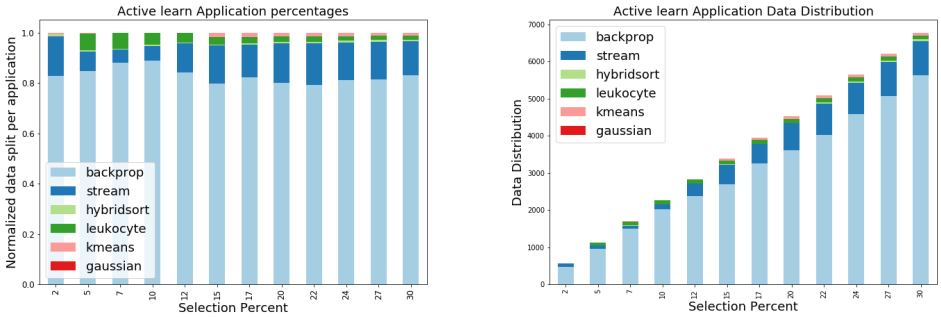


Figure 2.10: Normalized application percentage and distribution breakdown of data points created using active learning.

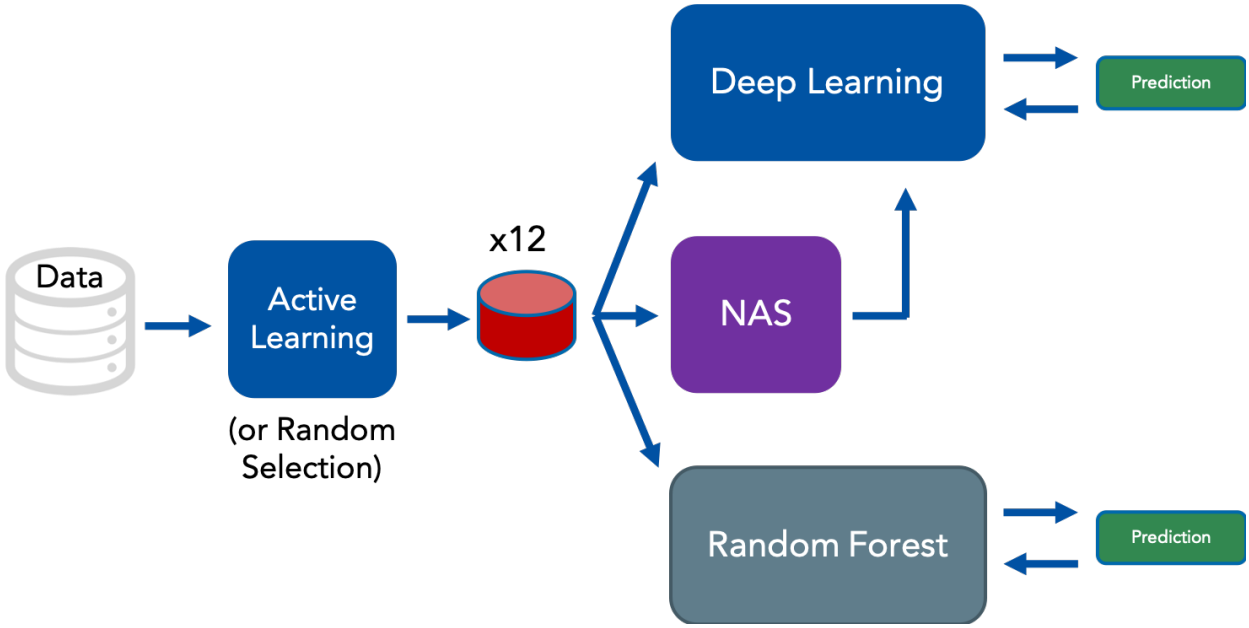


Figure 2.11: Illustration of modeling comparison workflow. We start we a pool of data points, create refined datasets with an active learner and have random selection as comparison. These datasets are then used in random forest, deep learning model, and neural architecture search.

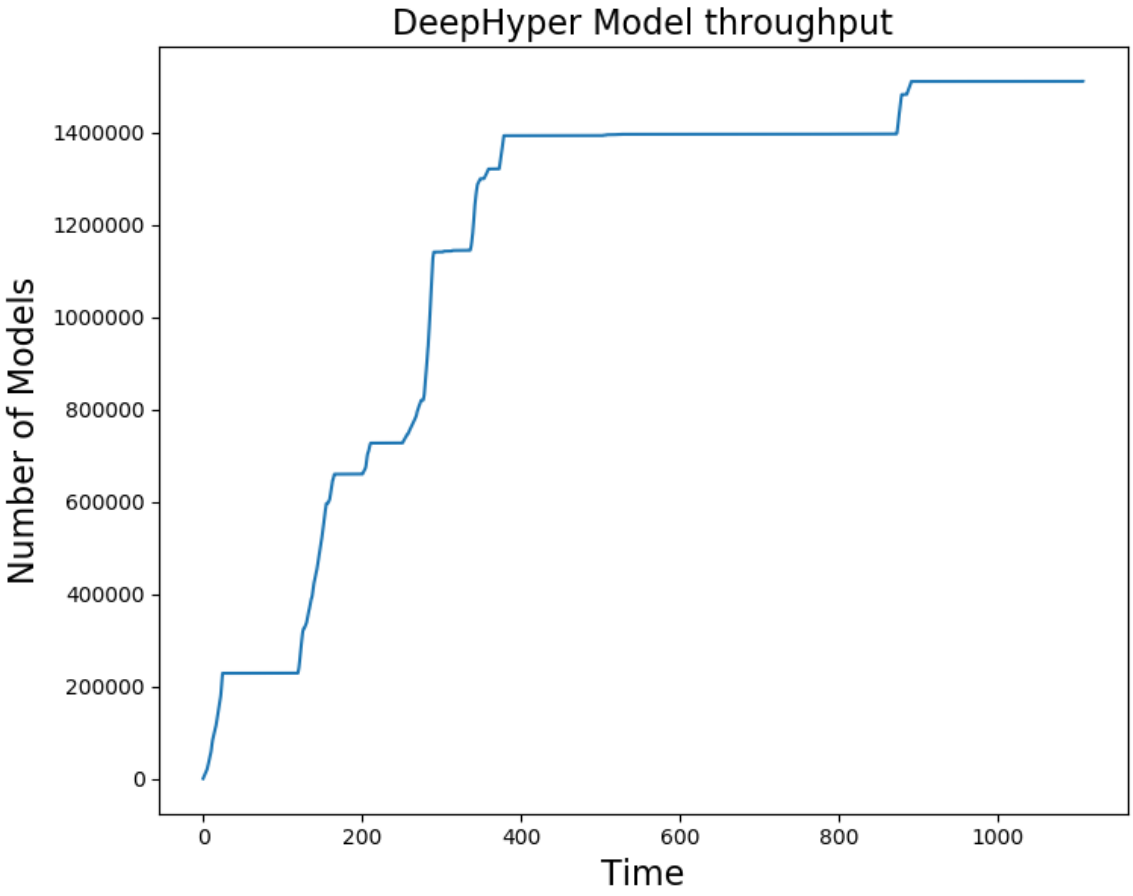


Figure 2.12: Model throughput results. The graph shows that over 1.4 million models were tested and created in about 1200 hours.

set of models needed for reinforcement learning based optimization.

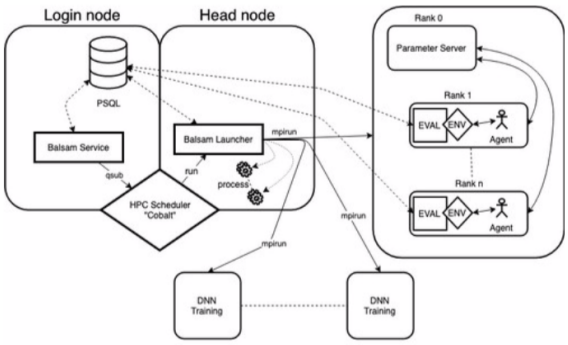


Figure 2.13: Illustration of distributed NAS architecture by Balaprakash et al. Balsam runs on a designated node where the launcher is a pilot job that runs on the allocated resources and launches tasks from the database. The multiagent search runs as a single MPI application, and each agent submits model evaluation tasks through the Balsam service API. The launcher continually executes evaluations as they are added by dispatch on idle worker nodes [8].

For each of the target datasets, we consider the following neural architecture search space. The search space comprises up to 10 architecture cells, with each cell containing a collection of architecture nodes. The nodes can manifest as various neural network features, depending on the actions of the reinforcement-learning agents. For example, the first node in each cell is the *Variable* node, which can be an identity operator (i.e. no layer added), or a dense layer with a range of sizes and activation-functions. In this case, the *Variable* nodes allowed for dense layers between the sizes of 16 and 96 (increment of 16), and activation functions in the set (None, relu, tanh, sigmoid). In order to enable skip connections, each cell also contained an optional connection to each of the previous three cells (including the input layer) and an addition operation to combine multiple input layers.

To use the defined search space to select an optimal neural architecture, NAS employs proximal policy optimization (PPO) search strategy. PPO is a policy gradient method for reinforcement learning which requires a reward estimation strategy for the agents. In this work we use R^2 for the reward estimation. The overall workflow for the DeepHyper-Balsam

NAS is illustrated in Figure 2.13 for the case of a single multi-agent search. The multi-agent search runs as a single MPI application with each of the agents submitting model evaluation tasks through Balsam. In this work, we leverage multiple multi-agent searches in parallel, allowing for the concurrent execution of multiple MPI applications.

The neural architecture search produces between 15,000 and 30,000 models for each training set. Since a thorough NAS requires large-scale computing resources for even a single target data set, the approach benefited greatly from access to Argonne's leadership computing facility. As shown in Figure 2.12, there were over 1.4 million models tested and created in about 1200 hours.

2.0.4.7 Deep learning and Random Forest

With the use of the same 24 training sets used for the NAS For comparison, we used a classical machine learning approach, random forest for performance prediction. A random forest is an essemble machine learning method that uses multiple decision trees. Each of the decisions trees are trained on different data sets where sampling is done with replacement [77, 102]. As with the previous models, we used P100 performance metrics as features and the V100 IPC as the target value. The particular random forest tested has 100 *n_estimators*. We use mean squared error as the optimization metric and used the full feature set.

Similar to the intra-node prediction model, we used a sequential fully-connected deep learning model. Deep learning creates models with multiple processing layers to learn representations of data with multiple levels of abstract [70]. Deep learning can discover complicated and intricate structures of large data sets. Our deep learning model uses an Adam optimizer and mean squared error for loss. In neural networks, the activation function transforms the summed weighted input from the node into the activation of the node or output for that input. All layers but the last layer use a $ReLU$ as an activation function as it showed better convergence compared to other activation functions. The results are trained

in batches of 250 points and for 1000 epochs. The model created can be seen in Figure 2.30. The same workflow is used when training with the full training set.

2.0.5 Results

Here we present the results of intra-architecture IPC prediction, inter-architecture memory bound classification, and inter-architecture IPC prediction. We also present the results of using two types of neural architectures.

2.0.5.1 Intra-node Architecture IPC Prediction

Although the ultimate goal is inter-architecture performance prediction, we use intra-architecture IPC prediction as a preliminary step. That is, we start by confirming that supervised machine-learning methods can be used to predict performance, given profiling data from the same GPU architecture. As shown in Figures 2.14 and 2.17, we observe mean absolute percentage errors of 4.11 and 2.96 when trained on 451 and 4,521 data points, respectively. Additionally, we explore the effects of using a reduced feature space (i.e. fewer profiling metrics) to train similar models. As illustrated in Figures 2.15 and 2.16, the use of a reduced feature space does not significantly degrade the accuracy of intra-architecture IPC prediction. This overall success implies that, even with a reduced characterization of the application, it is still possible to acquire accurate IPC predictions, thus motivating the more-challenging task of inter-architecture prediction.

2.0.5.2 Memory Bound Cross-Architecture Prediction

As shown in the confusion matrix in Table 2.2, both the false-positive and false-negative prediction rates for the inter-architecture memory-bound classifier are below 2%. The goal of the model is to predict if a particular application run will become memory bound on a V100 GPU, given profiling data from a P100 GPU. The task is complicated by the fact

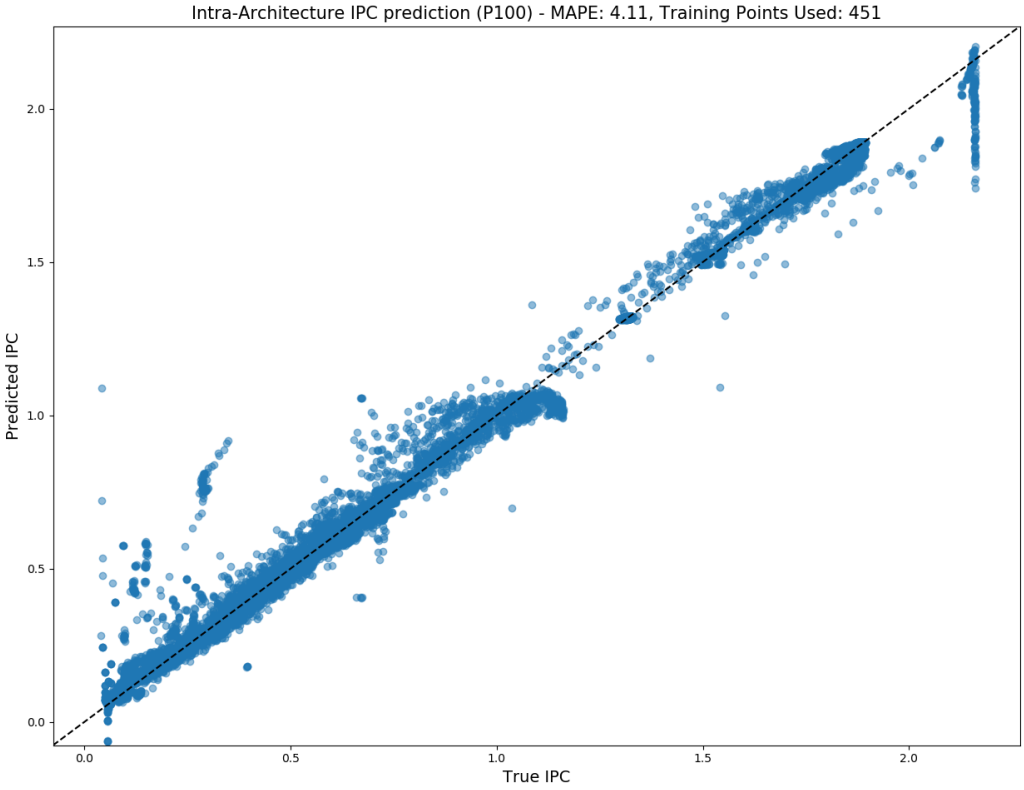


Figure 2.14: Results of P100 IPC prediction using 451 training data points.

Memory Bound Classifier Confusion Matrix		
	Predicted: No	Predicted: Yes
Actual: No	0.995	0.0048
Actual: Yes	0.0174	0.983

Table 2.2: Confusion matrix showing results of memory bound random forest classifier that predicts whether an application will be memory bound going from the P100 to V100 GPU architecture.

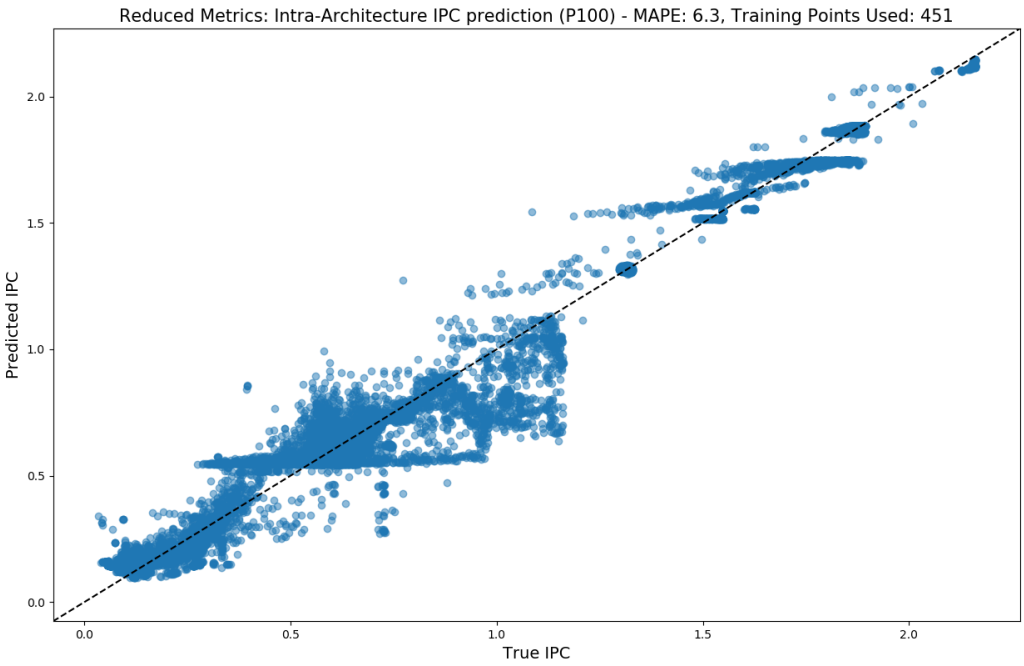


Figure 2.15: Results of P100 IPC prediction with reduced input metrics (5 total metrics) using 451 training data points.

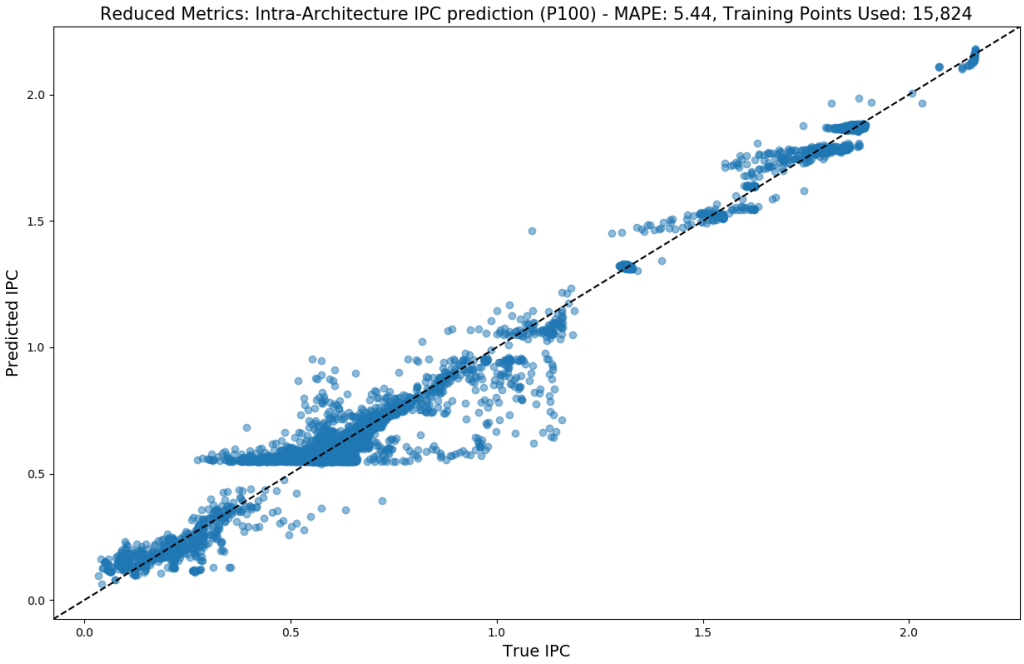


Figure 2.16: Results of P100 IPC prediction with reduced input metrics (5 total metrics) using 15,824 training data points.

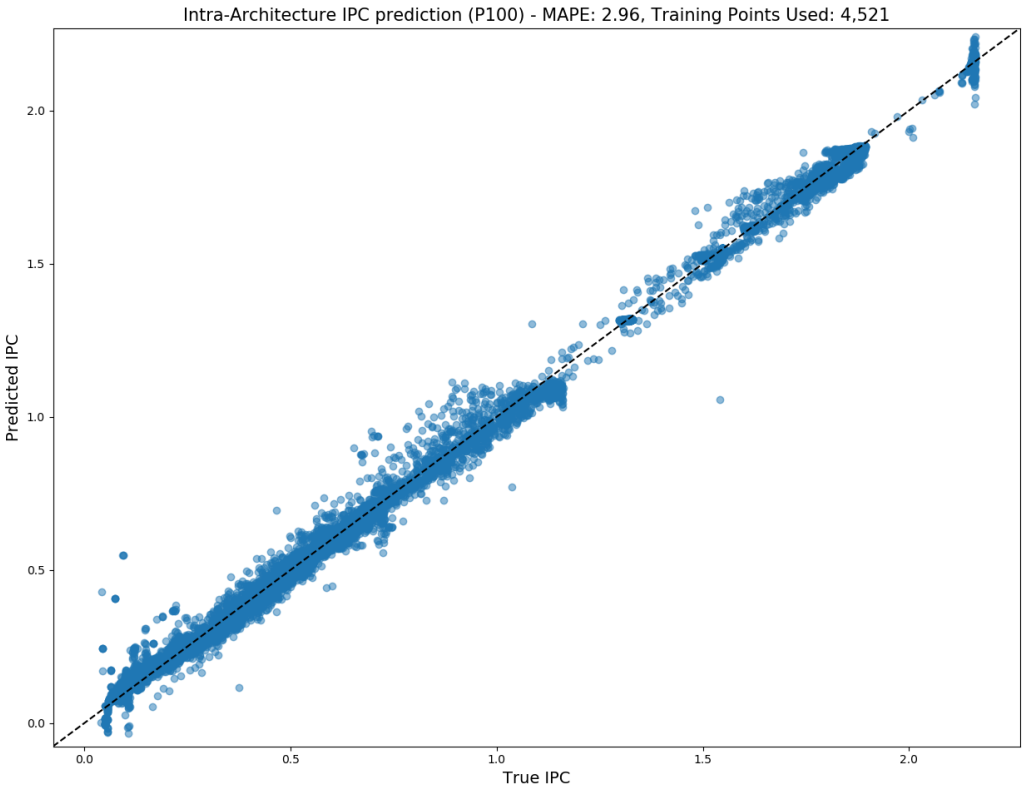


Figure 2.17: Results of P100 IPC prediction using 4,521 data points.

that none of the runs are memory bound on the P100 architecture, requiring the model to implicitly capture critical transitions in performance behavior. The results further suggest that there are some applications, such as backprop, that are likely to become memory bound when moved from the P100 to the V100.

2.0.5.3 Inter-Architecture IPC Prediction

First, we will look at overall IPC prediction across the three models tested. In Figures 2.18 and 2.19, the IPC prediction compared to true IPC across all applications is shown for random selection and active learning selection of 20% of the data, respectively, using DeepHyper. Across these graphs, it is clear that backprop dominates the data set and is a notoriously difficult application to predict compared to the other applications. Additionally, all models have difficulties in predicting leukocyte with good accuracy.

The majority of all training data corresponds to the backprop and stream applications. The results of backprop are further inspected in Figures 2.26 and 2.27, showing DeepHyper predictions using an active learning training set and a random selection training set. These results show that the generated models have trouble adjusting to scenarios when going from high-IPC values on P100 and to low-IPC values on V100, which is the case for a large portion of the backprop data points.

The stream prediction results are shown in Figures 2.28 and 2.29 for a DeepHyper model trained with active learning created and random selection training sets of size 20%. Although performance accuracy is relatively good for the reduced data set sizes, random forest surprisingly out performed a NAS optimized deep learning model with an actively learned training set.

These trends are further supported in the Figure 2.21, where the mean absolute percentage error (MAPE) is shown for each of the applications. Figure 2.23 shows the error bar for each application across all models along with a harmonic mean across the applications, as the

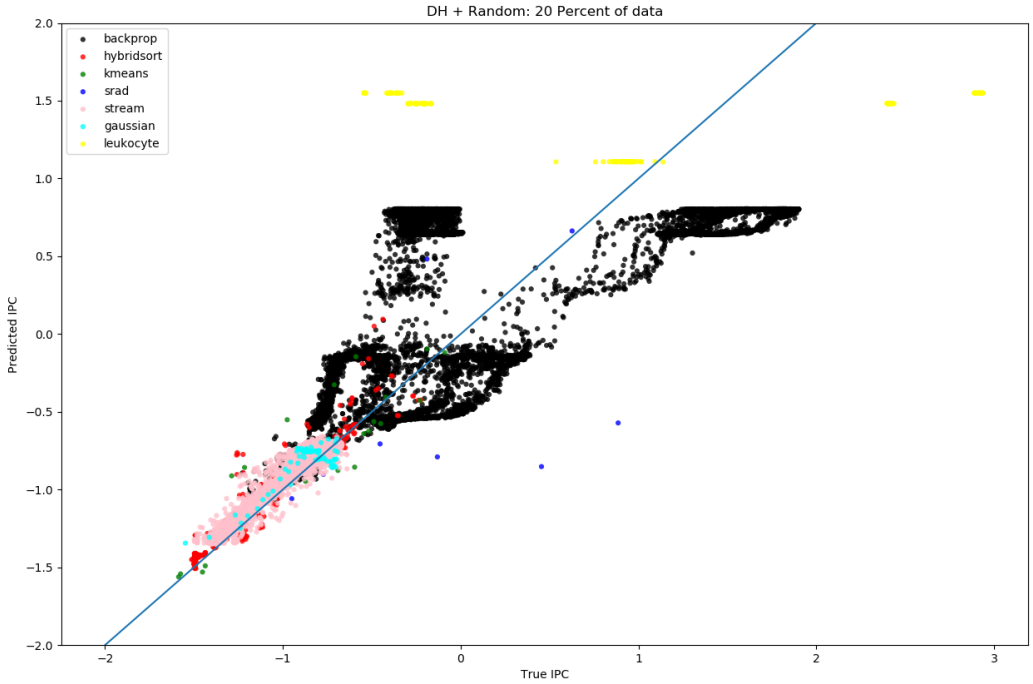


Figure 2.18: IPC prediction results of DeepHyper model using a training set with randomly chosen data points.

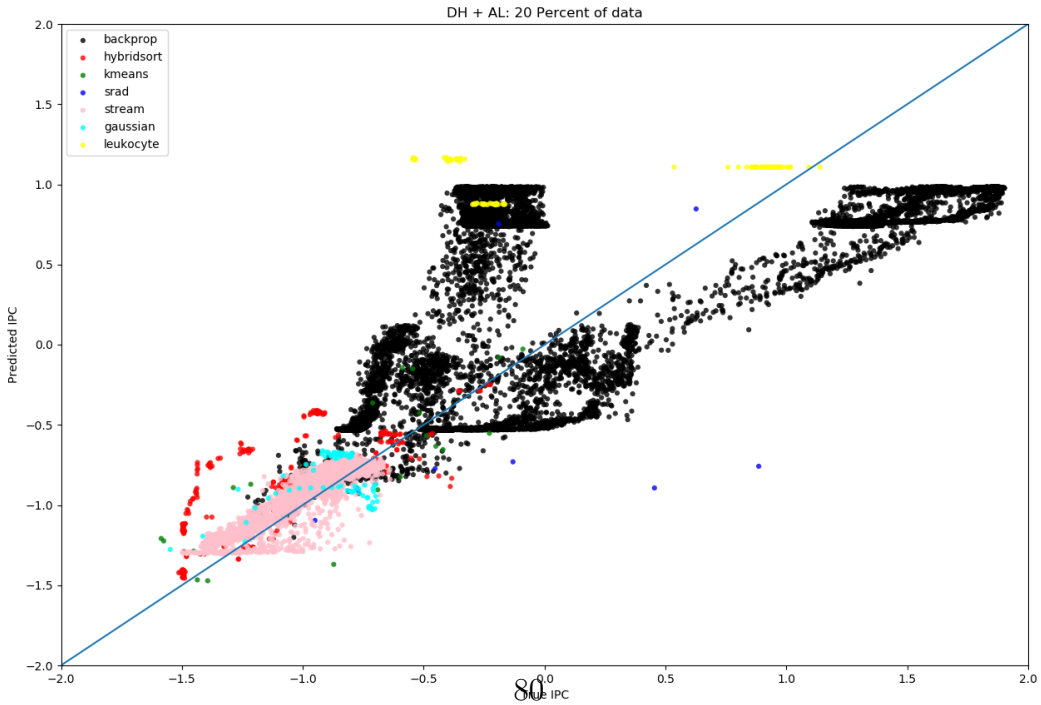


Figure 2.19: IPC prediction results of DeepHyper model using a active learning curated training set.

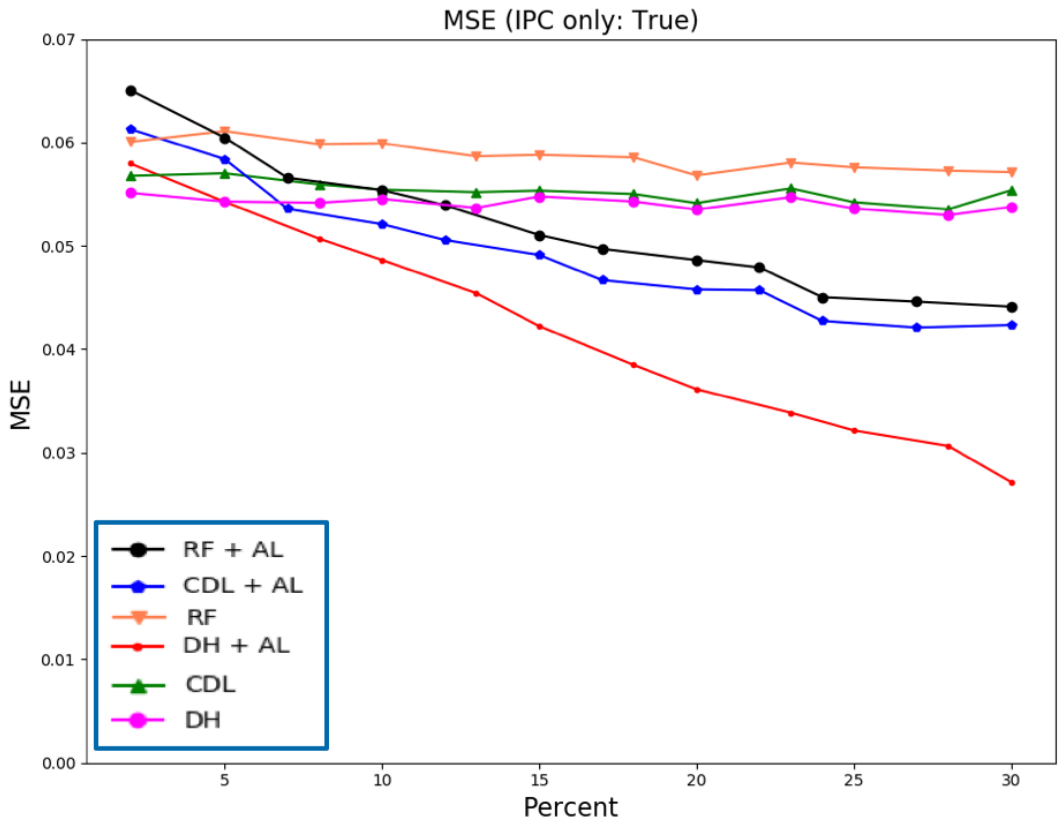


Figure 2.20: Mean square error, common loss metric used in machine learning models, shows a minor decrease for actively queried training sets. According to the MSE score, the DeepHyper returned model using an active learning queried training set shows a 36% decrease in error over a model using random selection.

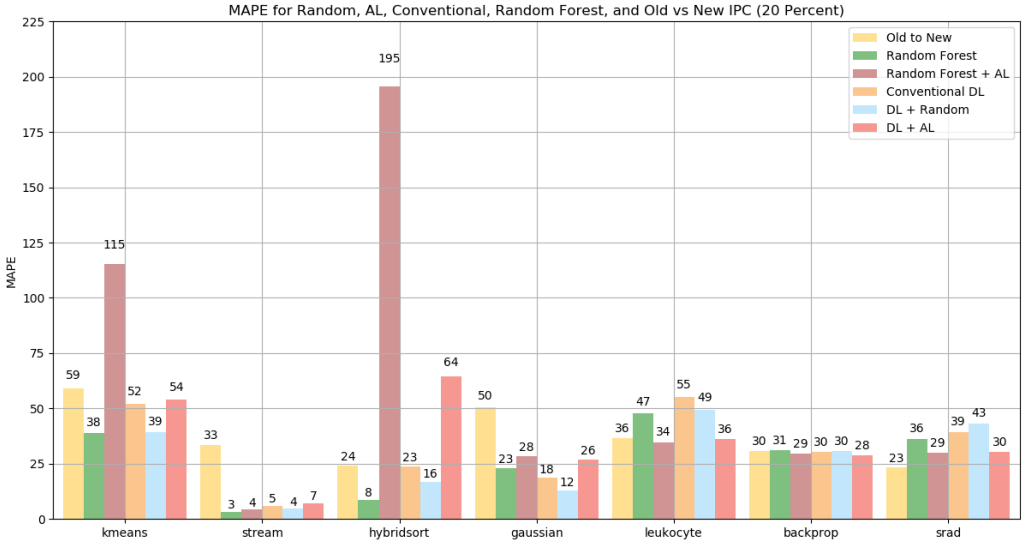


Figure 2.21: Mean absolute percentage error (MAPE) of each framework across applications tested with 20% of the training set used. Each number above the bar is the MAPE for each application and the corresponding model used

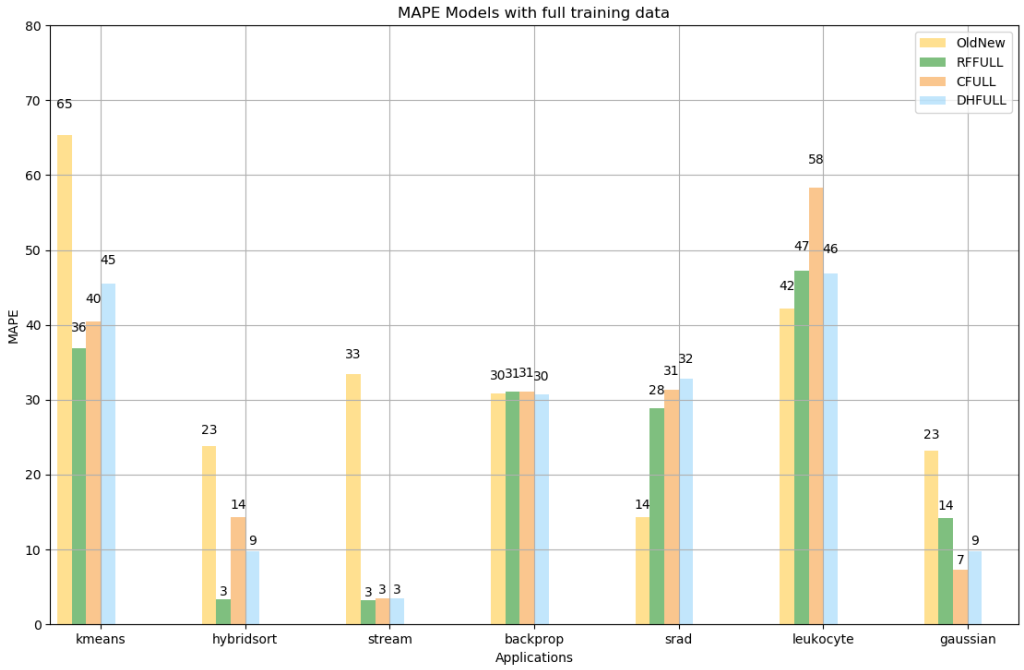


Figure 2.22: Mean absolute percentage error (MAPE) of models using full training set in comparison with Old and New IPC. Each number above the bar is the MAPE for each application and the corresponding model used

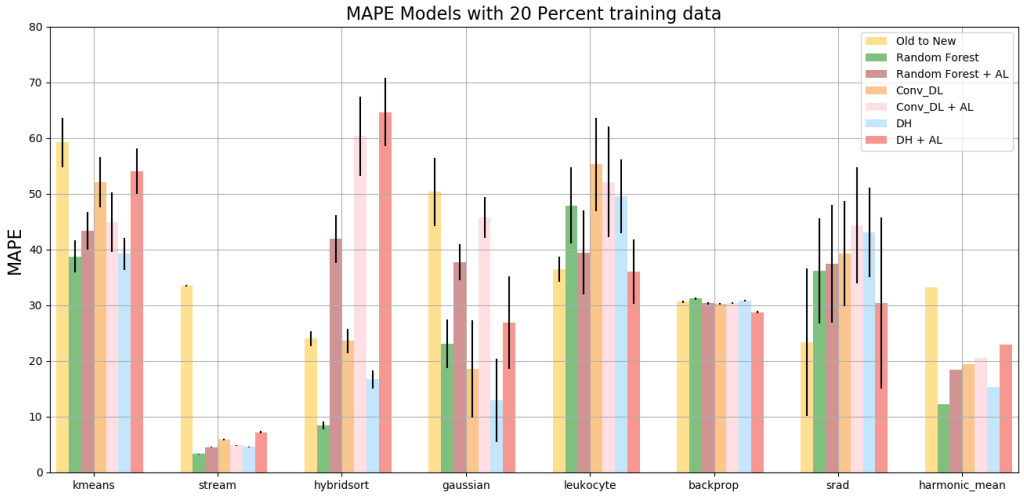


Figure 2.23: Mean absolute percentage error (MAPE) and error bar of each framework across applications tested with 20% of the training set used. The last set of the columns is the harmonic mean over the applications.

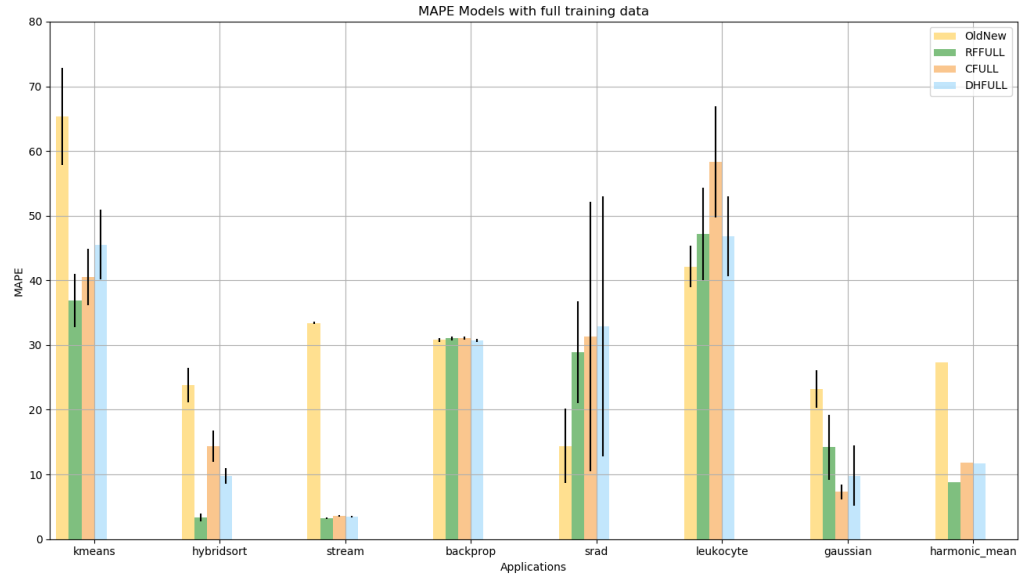


Figure 2.24: Mean absolute percentage error (MAPE) and error bar using full training set. The models shown here are the random forest (RFFULL), conventional deep learning model (CFULL), and DeepHyper NAS created model (DHFULL). Active learning results are not shown as the full training set is used. Harmonic mean across applications shows that random forest has better performance compare to both neural network models.

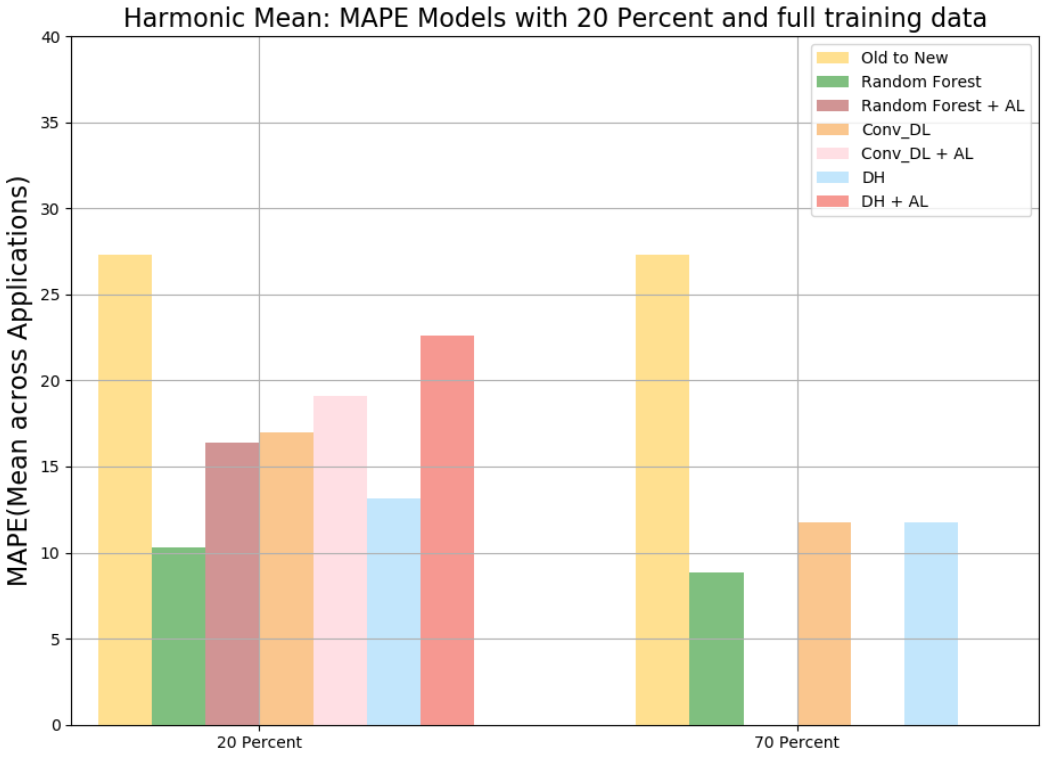


Figure 2.25: Graph of harmonic mean between application predictions of models using the full training dataset and 20% of the training dataset.

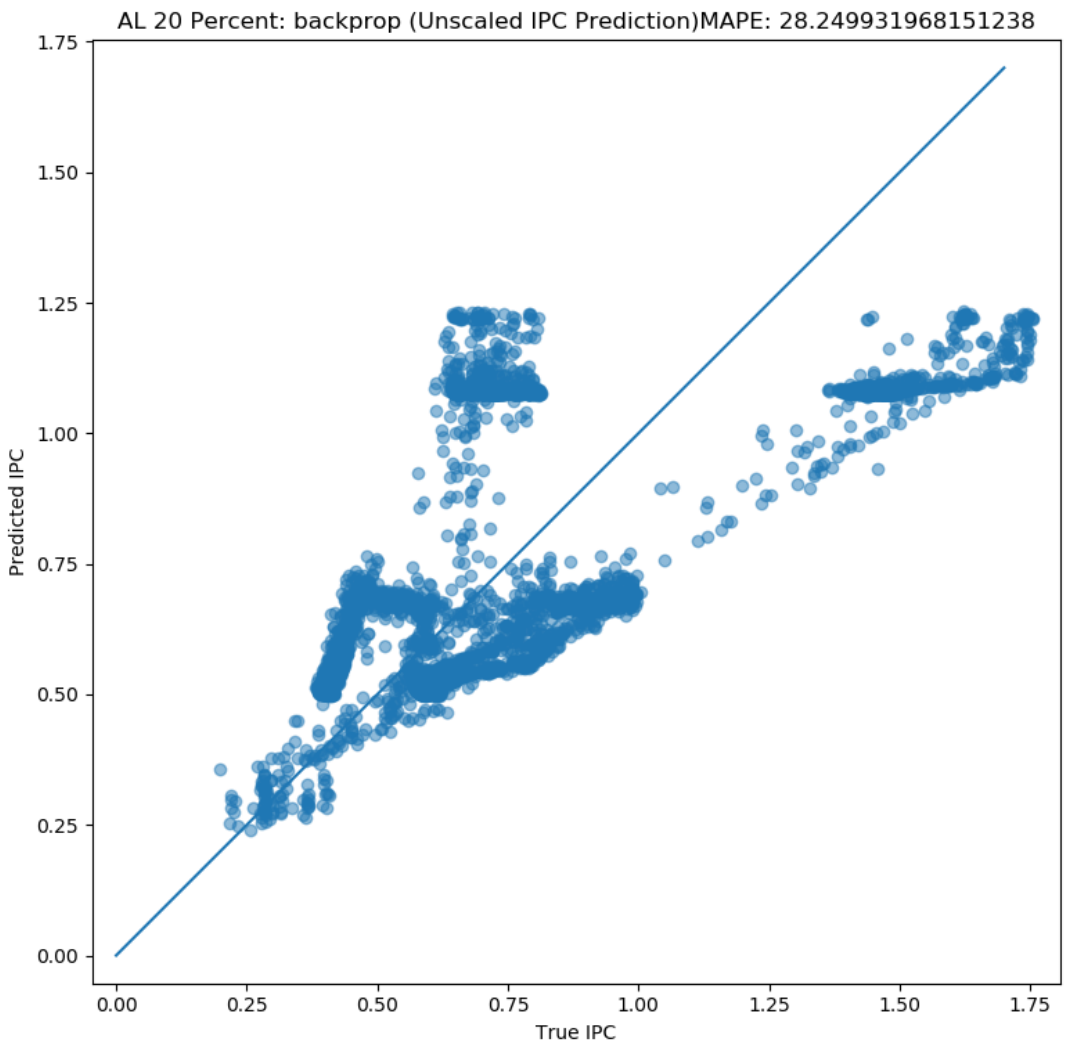


Figure 2.26: Prediction of backprop IPC using DeepHyper model with active learning chosen training set.

last set of bars. Among these applications, srad has the highest MAPE error variation across the data. Figure 2.22 shows the MAPE scores of the models using the full training set and Figure 2.24 shows the error bar among the same tested applications. Similar to the models trained with 20% of the training data, there is notable variation in the srad application and little to no variation in stream.

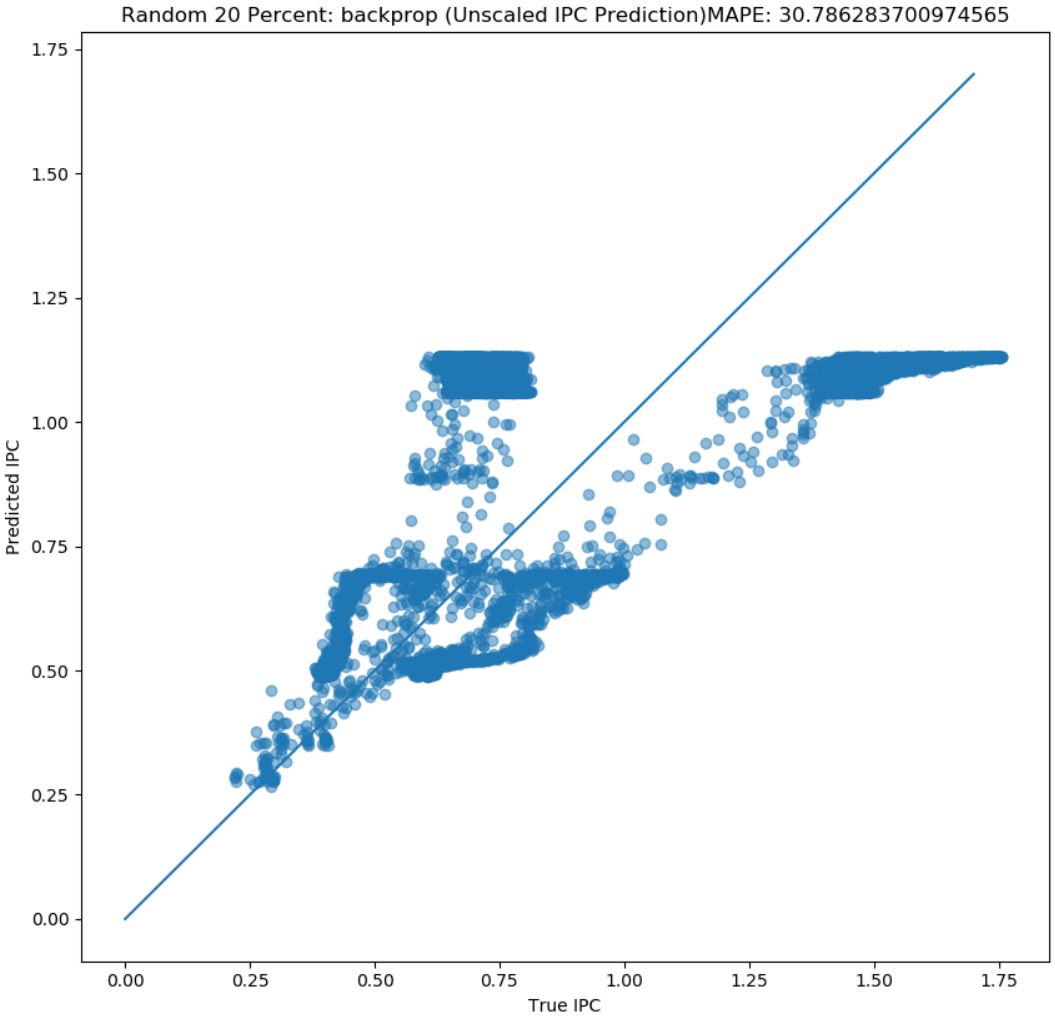


Figure 2.27: Prediction of backprop IPC using DeepHyper model a randomly chosen training set.

For the simplest baseline, we look at mapping the current architecture IPC value to the target IPC value : 'OldNew'. In other words, we assume that if the IPC is x on P100, then it will also be x on V100. Certain applications, such as stream, would not do well with this mapping, but applications such as backprop would not fair any worse. The 'Random Forest' bar corresponds to the prediction of V100 IPC using P100 metrics as features. The

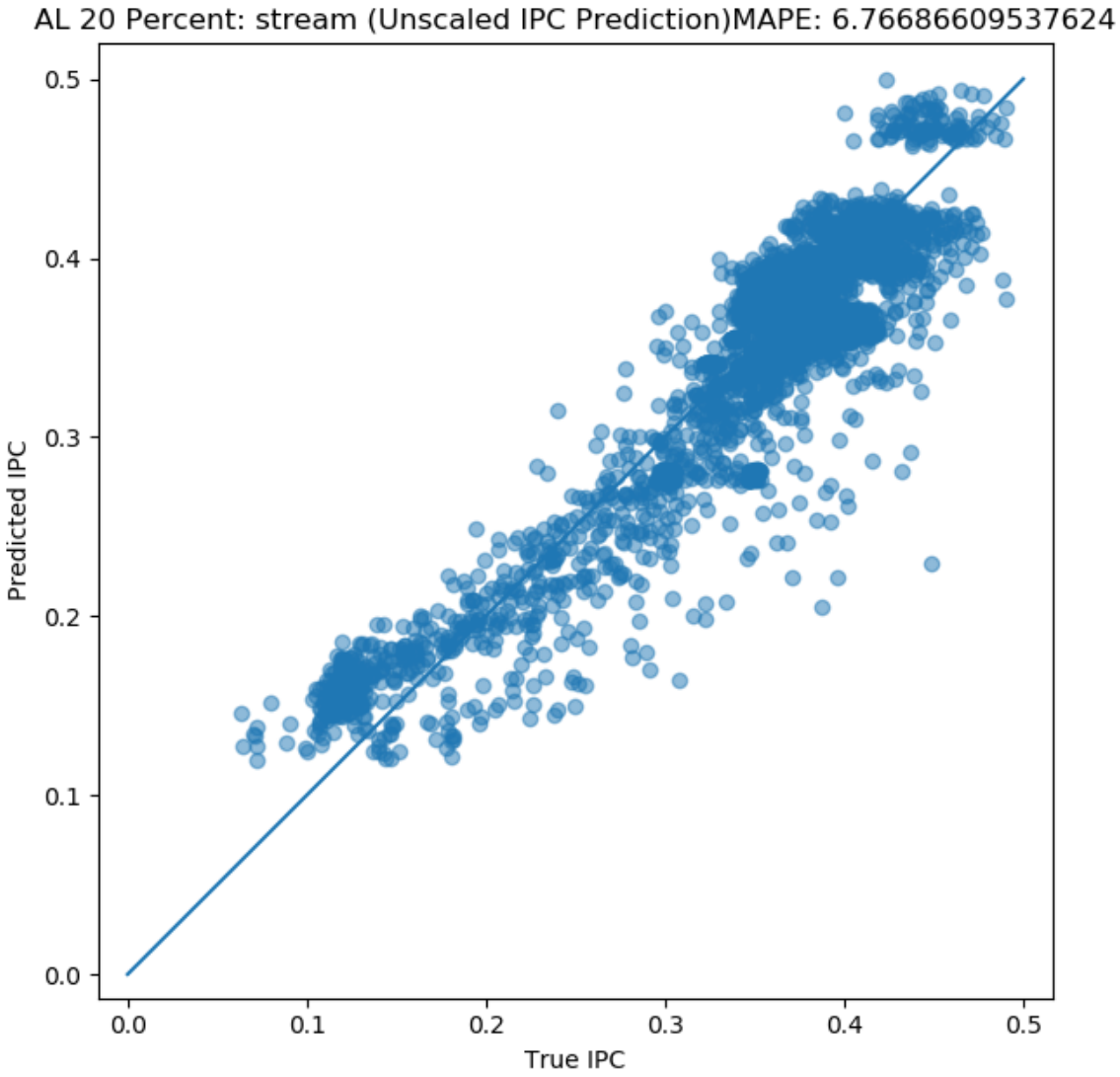


Figure 2.28: Prediction of Stream application using model returned from Deephyper using a training set curated by the active learning model

random forest performance is on par with the deep learning model performance or obtains lower error, such as the error on hybridsort. The 'Random Forest + AL' bar corresponds to a random forest model that uses a training set created by the active learner discussed above. This particular model does not do well in applications such as kmeans and hybridsort, in

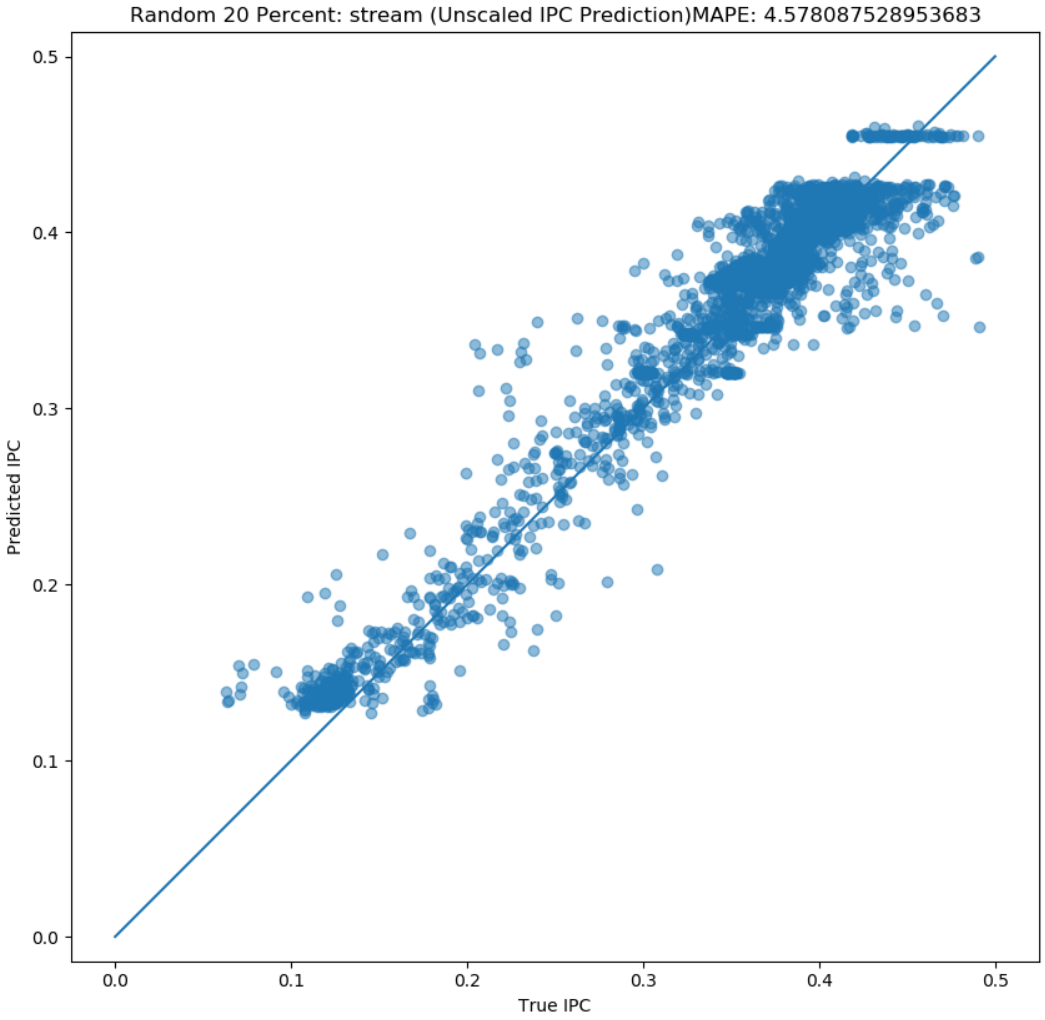


Figure 2.29: Prediction of Stream application using model returned from DeepHyper with random selection.

comparison to random forest without a curated training set by an active learner. This could be due to the fact that the active learning selection has a concentrated focus on the backprop and stream application data in comparison to these other applications, and thus does not focus on acquiring points in the training set for these applications.

The 'Conv_DL' bar corresponds to a conventionally developed deep learning model. The

simple, sequential structure of this neural network architecture is shown in Figure 2.30. We explored various network sizes, activation functions, and regularization approaches. We discovered that deeper models tended to overfit the data, while wider models achieved better results. The 'DH' bar corresponds to the NAS-generated model using a randomly sampled data set. The 'DL + AL' bar corresponds to a NAS-generated model using an actively learned training set. Figure 2.31 shows the diagram of a neural architecture chosen by NAS. This is the architecture returned by DeepHyper using a training data set created by an active learner, showing skip connections and a variation of activation functions chosen. Every model created by DeepHyper is as complex if not more complex than the one shown in Figure 2.31. Overall, when looking at Figure 2.25, which shows the Harmonic mean across applications for both models trained with the full training set and the partial training set, random forest outperforms all other models. Furthermore, tripling the data used to train the random forest shows very little improvement in error, compared to the conventional deep learning model.

Finally, considering all models use mean square error (MSE) as the loss metric, it is only fair to look at the results of mean squared error with respect to training data size and models used. Figure [**?**] shows a small decrease, in MSE for models using an active learning queried training set. There is about a 36% decrease in error when comparing a DeepHyper returned model with an active learn queried dataset to a randomly selected dataset. That being said, though there is a decrease in this error metric, when looking at the MAPE, it is not significant enough to warrant success for any of these models.

2.0.6 Summary

The initial results of the intra-architecture IPC prediction, using limited data, suggest that the available nvprof-based features are sufficient for accurate intra-architecture performance prediction. In-line with those results, the memory bound cross-architecture prediction, with

accuracy of 99%, can be particularly useful when identifying applications that become memory bound from one GPU architecture to another. This information can be useful to application developers and end users. Advanced knowledge of a memory-bound transition allows developers to focus on different performance optimizations and users to avoid the use of certain chip architectures altogether.

Overall the final DeepHyper model prediction does better than some models but still needs improvement, if limited data is a requirement. Given that a majority of the data corresponds to the backprop and stream application, shown in Figure 2.10, we can focus on these applications for our overall evaluation. Unfortunately, as shown in Figure 2.22, both random forest and NAS-optimized deep learning models fail to outperform a simple old to new mapping. The overall poor accuracy of IPC for backprop data suggests that this particular application is very difficult to predict, and that domain specific feature engineering engineering is most likely required to further improve the model.

In contrast to backprop, all models were successful at predicting performance of the stream application. While a simple old to new mapping results in a 33 percent error, both random forest and deep learning models reduced this error to 3 percent when using the full training set.

When comparing DeepHyper created models, there are some cases where the random selection does significantly better than the active learning selection. There are also cases where active learning out performs random selection. If we say a MAPE of under 5% is excellent, only one application prediction falls into that category - stream. These results show that active learning did not always give beneficial improvements to the model and at times reduced the accuracy tremendously. These results also show that even after training and testing over a million models, a more complicated heuristic and furthe feature engineering is needed to identify the relationship between these architectures.

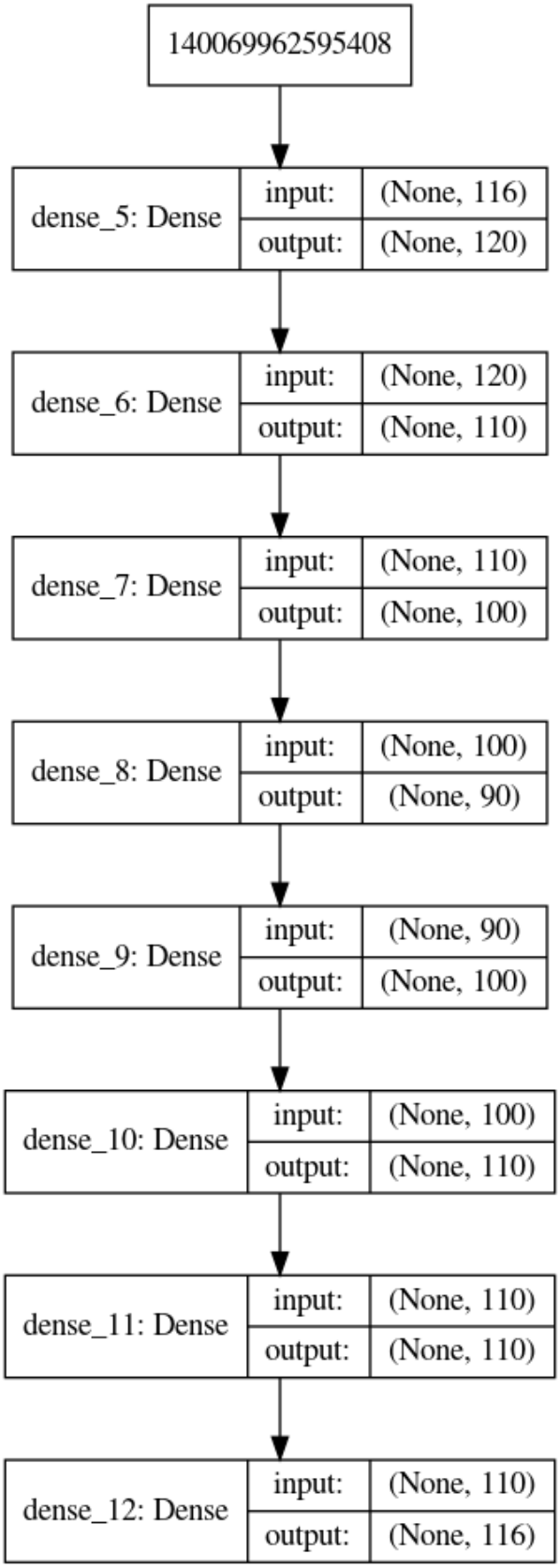


Figure 2.30: Conventional deep learning architecture layout.

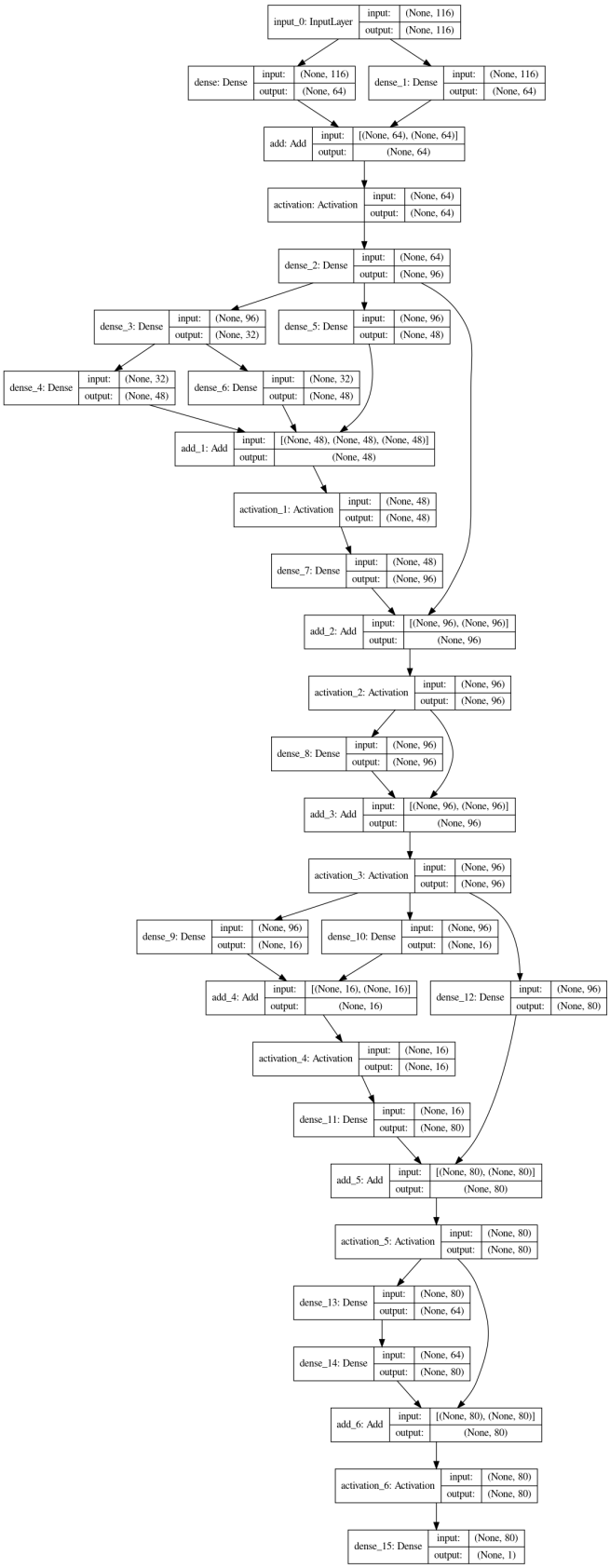


Figure 2.31: DeepHyper + Active learning architecture layout.

CHAPTER 3

SUMMARY AND FUTURE WORK

3.1 Proxima

Molecular dynamics often requires the use of high-cost functions, such as density functional theory. With the recent advances in machine learning, these high-cost function calls have been able to be replaced/modeled with deep-learning neural networks. The typical workflow has a fixed parameter on when the pre-trained model is used, replacing high-cost function without guarantees in accuracy or error.

In this thesis, we presented Proxima, a library that allows the integration of using a controller based decision engine into a molecular dynamics simulation to dynamically decide when to use the machine learned model and it's two implementations in Chapters 1 and 2. Proxima guarantees a given error within an approximate threshold. This library enables the on-the-fly automatic and dynamic use of the machine learned surrogate during a molecular dynamics simulation. This workflow will enable larger simulations as well as increase overall performance while staying within a given target error bound. Additionally, this is a step forward in creating a workflow that can more easily carried out to scale.

3.2 Performance Prediction

With the significant investment in performance prediction across architectures, we propose two case studies to empower research in performance projection on modern architectures. The two architectures we use in this work are NVIDIA'S P100 and V100. Though they are only one generation apart, the differences in the architectures is apparent in the the non-linear relationship among the performance metrics across a range of kernels. By being able to predict total memory throughput and IPC, the application developer can gain insights into future performance, such as whether or not their application will be memory bound on their

target architecture. Understanding this efficiency can help the developer decide whether or not to port their application onto said target architecture or if there's room for increase architecture utility. Therefore this insight the potential to help developers and practitioners optimize application development time.

3.3 Future Work

The insights illuminated by this work, open the door to a variety of follow-on research:

1. With the increasing complexity in molecular dynamics applications, integrating machine learning requires addressing the improvement of said machine learning model, often at simulation time. As a supplement and further improvement of Proxima, we can look at retraining the models on-the-fly. The challenges we are trying address are: reducing both the number of times the model needs to be retrained, along with limiting the amount of data the model needs. To do so, we can use a similar Proxima workflow, focused on retrain intervals. For example, the initial retrain interval will start with a guess on what the new input data set size needs to be before the models are retrained. After retraining, the controller will either increase or decrease the data window size dependent on whether or not it increases or decreases the prediction error. If there is little improvement in prediction error, a larger dataset is needed. As the model starts improving or the simulation keeps visiting states where it has sufficient training data, future dataset sizes will get smaller (speeding up model training as the simulation goes on). Eventually, the model will no longer need retraining or retraining becomes sparse. We would have a model manager on one end (with it's own controller), database of atom data on disk, and Proxima on the other end, controlling surrogate usage on the application. Having a model manager, disjoint to Proxima, will allow for many models to train and the application to run asynchronously among many processes. These processes will place new datapoints into the core atom database where the model manager

can check the database has reached the optimal size and retrain the models. Proxima will have a low cost check if the models on disk have been updated and use the new models when they have, accelerating the application, while still producing scientifically usable results. Finally, we will want to incorporate Proxima in to the ASE library system. ASE supports over a dozen atomistic calculators and with a set of tools that allow for quick set-up of a variety of atomistic simulations. Having the availability of Proxima in such a widely used library will not only increase usability of surrogate models in molecular dynamics simulations.

2. We can leverage the findings from the performance prediction work to study performance portability. Calculating performance portability often requires compute heavy runs across several architectures. Leveraging low-cost machine learning, such as the memory throughput prediction model, can reduce the time needed to acquire such numbers.

REFERENCES

[1] Profiler :: Cuda toolkit documentation. `https://docs.nvidia.com/cuda/profiler-users-guide/index.html`. (Accessed on 12/05/2019).

[2] Programming guide :: Cuda toolkit documentation. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`. (Accessed on 12/02/2019).

[3] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, 76:243–297, 2021.

[4] Mohamad Mahmoud Al Rahhal, Yakoub Bazi, Haikel AlHichri, Naif Alajlan, Farid Melgani, and Ronald R Yager. Deep learning approach for active classification of electrocardiogram signals. *Information Sciences*, 345:340–354, 2016.

[5] 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.

[6] Newsha Ardalani, Clint Lestourgeon, Karthikeyan Sankaralingam, and Xiaojin Zhu. Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance. In *Proceedings of the 48th International Symposium on Microarchitecture*, pages 725–737. ACM, 2015.

[7] Rajive Bagrodia, Ewa Deeljman, Steven Docy, and Thomas Phan. Performance prediction of large parallel applications using parallel simulations. In *ACM SIGPLAN Notices*, volume 34, pages 151–162. ACM, 1999.

[8] Prasanna Balaprakash, Romain Egele, Misha Salim, Stefan Wild, Venkatram Vishwanath, Fangfang Xia, Tom Brettin, and Rick Stevens. Scalable reinforcement-learning-based neural architecture search for cancer deep learning research. *arXiv preprint arXiv:1909.00311*, 2019.

[9] Prasanna Balaprakash, Robert B Gramacy, and Stefan M Wild. Active-learning-based surrogate models for empirical performance tuning. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 1–8. IEEE, 2013.

[10] Prasanna Balaprakash, Karl Rupp, Azamat Mametjanov, Robert B Gramacy, Paul D Hovland, and Stefan M Wild. Empirical performance modeling of gpu kernels using active learning. In *ParCo*, pages 646–655. Citeseer, 2013.

[11] Prasanna Balaprakash, Michael Salim, Thomas Uram, Venkat Vishwanath, and Stefan Wild. Deephyper: Asynchronous hyperparameter search for deep neural networks. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 42–51. IEEE, 2018.

[12] 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.

[13] Michael Balmer, Marcel Rieser, Konrad Meister, David Charypar, Nicolas Lefebvre, and Kai Nagel. Matsim-t: Architecture and simulation times. In *Multi-agent systems for traffic and transportation engineering*, pages 57–78. IGI Global, 2009.

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

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

[16] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

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

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

[19] 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.

[20] 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.

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

[22] Michael Boyer, Jiayuan Meng, and Kalyan Kumaran. Improving gpu performance prediction with data transfer modeling. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 1097–1106. IEEE, 2013.

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

[24] 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.

[25] 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.

[26] Laura Carrington, Allan Snavely, Xiaofeng Gao, and Nicole Wolter. A performance prediction framework for scientific applications. In *International Conference on Computational Science*, pages 926–935. Springer, 2003.

[27] Laura Carrington, Nicole Wolter, Allan Snavely, and Cynthia Bailey Lee. Applying an automated framework to produce accurate blind performance predictions of full-scale hpc applications. In *Department of Defense Users Group Conference*, 2004.

[28] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE international symposium on workload characterization (IISWC)*, pages 44–54. Ieee, 2009.

[29] Tuck C Choy. *Effective medium theory: principles and applications*, volume 165. Oxford University Press, 2015.

[30] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

[31] Matthieu Courbariaux, Jean-Pierre David, and Yoshua Bengio. Low precision storage for deep learning. *arXiv preprint arXiv:1412.7024*, 2014.

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

[33] CJ Cramer. Molecular mechanics. *Essentials of Computational Chemistry. Theories and Models, 2nd ed., John Wiley and Sons Ltd., England*, pages 36–37, 2004.

[34] Gabor Csányi, T Albaret, MC Payne, and Alessandro De Vita. "learn on the fly": A hybrid classical and quantum-mechanical molecular dynamics simulation. *Physical review letters*, 93(17):175503, 2004.

[35] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI*, volume 5, pages 746–751, 2005.

[36] 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.

[37] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[38] Yi Ding, Nikita Mishra, and Henry Hoffmann. Generative and multi-phase learning for computer systems optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, pages 39–52, New York, NY, USA, 2019. ACM.

[39] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] Soraya Ghiasi, Thomas Walter Keller Jr, Ramakrishna Kotla, and Freeman Leigh Rawson III. Scheduling processor voltages and frequencies based on performance prediction and power constraints, June 10 2008. US Patent 7,386,739.

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

[46] 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.

[47] 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.

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

[49] Dilek Hakkani-Tür, Giuseppe Riccardi, and Allen Gorin. Active learning for automatic speech recognition. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages IV–3904. IEEE, 2002.

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

[51] Tomas Hansson, Chris Oostenbrink, and WilfredF van Gunsteren. Molecular dynamics simulations. *Current opinion in structural biology*, 12(2):190–196, 2002.

[52] Stephen Lien Harrell, Joy Kitson, Robert Bird, Simon John Pennycook, Jason Sewall, Douglas Jacobsen, David Neill Asanza, Abaigail Hsu, Hector Carrillo Carrillo, Hessoo Kim, et al. Effective performance portability. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 24–36. IEEE, 2018.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] Scott A Hollingsworth and Ron O Dror. Molecular dynamics simulation for all. *Neuron*, 99(6):1129–1143, 2018.

[59] Qi-Jun Hong and Axel Van De Walle. A user guide for SLUSCHI: Solid and liquid in ultra small coexistence with hovering interfaces. *Calphad*, 52:88–97, 2016.

[60] Sunpyo Hong and Hyesoon Kim. An integrated gpu power and performance model. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 280–289. ACM, 2010.

[61] 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.

[62] Connor Imes, Huazhe Zhang, Kevin Zhao, and Henry Hoffmann. CoPPer: Soft real-time application performance using hardware power capping. In *IEEE International Conference on Autonomic Computing*, pages 31–41. IEEE, 2019.

[63] Engin Ipek, Bronis R De Supinski, Martin Schulz, and Sally A McKee. An approach to performance prediction for parallel applications. In *European Conference on Parallel Processing*, pages 196–205. Springer, 2005.

[64] 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.

[65] Rosie Jones, Rayid Ghani, Tom Mitchell, and Ellen Riloff. Active learning for information extraction with multiple view feature sets. *Proc. of Adaptive Text Extraction and Mining, EMCL/PKDD-03, Cavtat-Dubrovnik, Croatia*, pages 26–34, 2003.

[66] Takafumi Kanamori. Pool-based active learning with optimal sampling distribution and its information geometrical interpretation. *Neurocomputing*, 71(1-3):353–362, 2007.

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

[68] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[69] Elias Konstantinidis and Yiannis Cotronis. A practical performance model for compute and memory bound gpu kernels. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 651–658. IEEE, 2015.

[70] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[71] Benjamin C Lee and David M Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 185–194. ACM, 2006.

[72] Benjamin C Lee, David M Brooks, Bronis R de Supinski, Martin Schulz, Karan Singh, and Sally A McKee. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 249–258. ACM, 2007.

[73] David D Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning Proceedings*, pages 148–156. Elsevier, 1994.

[74] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *SIGIR'94*, pages 3–12. Springer, 1994.

[75] 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):096405, 2015.

[76] 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.

[77] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[78] 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.

[79] Gabriel Marin and John Mellor-Crummey. Cross-architecture performance predictions for scientific applications using parameterized models. In *ACM SIGMETRICS Performance Evaluation Review*, volume 32, pages 2–13. ACM, 2004.

[80] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S Vetter. Nvidia tensor core programmability, performance & precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531. IEEE, 2018.

[81] Andrew Kachites McCallumzy and Kamal Nigamy. Employing em and pool-based active learning for text classification. In *Proc. International Conference on Machine Learning (ICML)*, pages 359–367. Citeseer, 1998.

[82] John D McCalpin. Stream benchmark. *Link: www. cs. virginia. edu/stream/ref. html# what*, 22, 1995.

[83] Jiayuan Meng, Vitali A Morozov, Kalyan Kumaran, Venkatram Vishwanath, and Thomas D Uram. Grophecy: Gpu performance projection from cpu code skeletons. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 14. ACM, 2011.

[84] 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.

[85] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. Caloree: Learning control for predictable latency and low energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 184–198, New York, NY, USA, 2018. ACM.

[86] Nikita Mishra, John D Lafferty, and Henry Hoffmann. Esp: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*, pages 125–134. IEEE, 2017.

[87] Nikita Mishra, Huazhe Zhang, John D Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *ACM SIGPLAN Notices*, volume 50, pages 267–281. ACM, 2015.

[88] Christoph Morbitzer, PA Strachan, Brian Spires, David Cafferty, and Jim Webster.

Integration of building simulation into the design process of an architectural practice. 2001.

[89] 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.

[90] Shubhendu S Mukherjee, Sarita V Adve, Todd Austin, Joel Emer, and Peter S Magnusson. Performance simulation tools. *Computer*, (2):38–39, 2002.

[91] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Toward accelerating deep learning at scale using specialized hardware in the datacenter. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–38. IEEE, 2015.

[92] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. Marss: a full system simulator for multicore x86 cpus. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1050–1055. IEEE, 2011.

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

[94] Dennis C Rapaport and Dennis C Rapaport Rapaport. *The art of molecular dynamics simulation*. Cambridge university press, 2004.

[95] Giuseppe Riccardi and Dilek Hakkani-Tur. Active learning: Theory and applications to automatic speech recognition. *IEEE transactions on speech and audio processing*, 13(4):504–511, 2005.

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

[97] 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.

[98] 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.

[99] Michael A Salim, Thomas D Uram, J Taylor Childers, Prasanna Balaprakash, Venkatram Vishwanath, and Michael E Papka. Balsam: Automated scheduling and execution of dynamic, data-intensive hpc workflows. *arXiv preprint arXiv:1909.08704*, 2019.

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

[101] 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.

[102] Mark R Segal. Machine learning benchmarks and random forest regression. 2004.

[103] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

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

[105] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times using

historical information. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 122–142. Springer, 1998.

[106] Warren Smith, Valerie Taylor, and Ian Foster. Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Workshop on Job scheduling strategies for Parallel Processing*, pages 202–219. Springer, 1999.

[107] 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.

[108] Masashi Sugiyama and Shinichi Nakajima. Pool-based active learning in approximate linear regression. *Machine Learning*, 75(3):249–274, 2009.

[109] Lukasz G Szafaryn, Kevin Skadron, and Jeffrey J Saucerman. Experiences accelerating matlab systems biology applications. In *Proceedings of the Workshop on Biomedicine in Computing: Systems, Architectures, and Circuits*, pages 1–4, 2009.

[110] Cynthia A Thompson, Mary Elaine Califf, and Raymond J Mooney. Active learning for natural language parsing and information extraction. In *ICML*, pages 406–414. Citeseer, 1999.

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

[112] 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.

[113] 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.

[114] 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.

[115] Stefan Wager, Trevor Hastie, and Bradley Efron. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *The Journal of Machine Learning Research*, 15(1):1625–1651, 2014.

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

[117] Chao Wang, Lei Gong, Qi Yu, Xi Li, Yuan Xie, and Xuehai Zhou. Dlau: A scalable deep learning accelerator unit on fpga. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3):513–517, 2016.

[118] Han Wang, Linfeng Zhang, Jiequn Han, and E Weinan. Deepmd-kit: A deep learning package for many-body potential energy representation and molecular dynamics. *Computer Physics Communications*, 228:178–184, 2018.

[119] 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.

[120] 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.

[121] Leo T Yang, Xiaosong Ma, and Frank Mueller. Cross-platform performance prediction of parallel applications using partial execution. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 40. IEEE Computer Society, 2005.

[122] Lingyun Yang, Jennifer M Schopf, and Ian Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 31. ACM, 2003.

[123] Qi Yu, Chao Wang, Xiang Ma, Xi Li, and Xuehai Zhou. A deep learning prediction process accelerator based fpga. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 1159–1162. IEEE, 2015.

[124] Jidong Zhai, Wenguang Chen, and Weimin Zheng. Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node. In *ACM Sigplan Notices*, volume 45, pages 305–314. ACM, 2010.

[125] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170. ACM, 2015.

[126] Yi Zhang, Wei Xu, and James P Callan. Exploration and exploitation in adaptive filtering based on bayesian active learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 896–903, 2003.

[127] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.