Generative and Multi-phase Learning for Computer Systems Optimization

Yi Ding
University of Chicago
dingy@uchicago.edu

Nikita Mishra
University of Chicago
nmishra@cs.uchicago.edu

Henry Hoffmann
University of Chicago
hankhoffmann@cs.uchicago.edu

ABSTRACT

Machine learning and artificial intelligence are invaluable for computer systems optimization: as computer systems expose more resources for management, ML/AI is necessary for modeling these resources' complex interactions. The standard way to incorporate ML/AI into a computer system is to first train a learner to accurately predict the system's behavior as a function of resource usage—*e.g.*, to predict energy efficiency as a function of core usage—and then deploy the learned model as part of a system—*e.g.*, a scheduler. In this paper, we show that (1) continued improvement of learning accuracy may not improve the systems result, but (2) incorporating knowledge of the systems problem into the learning process improves the systems results even though it may not improve overall accuracy. Specifically, we learn application performance and power as a function of resource usage with the systems goal of meeting latency constraints with minimal energy. We propose a novel *generative model* which improves learning accuracy given scarce data, and we propose a *multi-phase sampling* technique, which incorporates knowledge of the systems problem. Our results are both positive and negative. The generative model improves accuracy, even for state-of-the-art learning systems, but negatively impacts energy. Multi-phase sampling reduces energy consumption compared to the state-of-the-art, but does not improve accuracy. These results imply that learning for systems optimization may have reached a point of diminishing returns where accuracy improvements have little effect on the systems outcome. Thus we advocate that future work on learning for systems should de-emphasize accuracy and instead incorporate the system problem's structure into the learner.

CCS CONCEPTS

• **Computing methodologies → Machine learning**; • **Computer systems organization → Heterogeneous (hybrid) systems**; **Embedded systems**; **Real-time system architecture**; • **Hardware → Chip-level power issues**.

KEYWORDS

Machine learning; real-time systems; energy; heterogeneous architectures; resource allocation

ACM Reference Format:
Yi Ding, Nikita Mishra, and Henry Hoffmann. 2019. Generative and Multi-phase Learning for Computer Systems Optimization. In *The 46th Annual International Symposium on Computer Architecture (ISCA '19), June 22–26, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3307650.3326633

1 INTRODUCTION

Computer systems optimization is increasingly multidimensional: systems must deliver reliable performance (*e.g.*, quality-of-service or latency guarantees) while minimizing energy consumption. To meet these conflicting goals, computer architects expose resources for software management. System software is then responsible for configuring these resources to operate at an optimal point in the performance-energy tradeoff space.

Systems expose a wide variety of resources—including, but not limited to heterogeneous core types, multiple sockets, configurable memory hierarchies, adjustable clockspeeds—and these resources have complex, non-linear effects on performance and energy. For example, on little cores clockspeed might uniformly increase performance, while on big cores high clockspeeds might induce thermal throttling, causing performance to decline. These types of resource interactions create local optima and make it difficult (or impossible) for gradient-based optimization and other heuristics to find a true optimal resource allocation. Indeed, several studies show that the increasing variety and complexity of configurable resources has rendered venerable heuristics ineffective [9, 24, 32, 37, 45, 47].

In recent years, machine learning techniques have shown the potential to increase system robustness by replacing resource management heuristics. Machine learning can model resources' complicated, non-linear interactions to avoid local optima and deliver a true optimal solution. Indeed, as systems complexity has grown researchers have proposed a variety of machine learning methods for system resource management [6, 12, 17, 18, 21, 25, 31, 33, 45, 47, 49, 50, 52, 67]. While this prior work shows that machine learning is effective for modeling complicated tradeoffs, there are several challenges that must be addressed to continue improving learning for computer system resource management, including:

- *Scarce Data:* To increase learning *accuracy*—*i.e.*, the learned model's ability to predict ground truth for some unseen application and system configuration—a robust set of training data is required. Collecting this training data is expensive: it requires observing a benchmark set in many different configurations, during which the machine to be modeled is not doing useful work. Additionally, the training benchmarks must exhibit a wide range of behavior, so that they can make accurate predictions for previously unseen applications.

ISCA '19, June 22–26, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6669-4/19/06…$15.00
https://doi.org/10.1145/3307650.3326633

- *Asymmetric Benefits:* Most learning problems require equal accuracy for all inputs. Furthermore, achieving better results for one input class represents a *biased* learner, a condition to be avoided, in general. In learning for systems, however, not all resource configurations are useful, in the sense of representing optimal tradeoffs (*e.g.*, between performance and power).[1] Ideally, a computer system would only use configurations on the optimal frontier and ignore all non-optimal configurations; *i.e.*, unlike general learning problems, biasing the learner towards configurations representing optimal tradeoffs is beneficial. The challenge is that we do not know which configurations are optimal to begin.

We address these two challenges by presenting two techniques that improve a variety of learning methods for computer system management. First, we propose a novel *generative model* that addresses the scarce data challenge by generating training data that improves learning accuracy. The key insight for the generative model is determining how to generate data that is sufficiently different from the training set, but still realistic enough to predict unseen behavior. Second, we propose *multi-phase sampling* to address asymmetric benefits by splitting sampling into two phases: first, separating the optimal configurations from the rest and second, improving the prediction accuracy of the optimal points.

We test these techniques by implementing them for both an ARM big.LITTLE mobile and an Intel x86 server. We use five different published learning systems to both predict performance and power consumption for unseen benchmark applications and to schedule resources to meet application latency requirements with minimal energy. We compare the results of these published learners to the same learners augmented with our proposed generative model and multi-phase sampling. Our results show the following:

- The generative model improves predictions for all learners on both the mobile and server systems. The average increase in prediction accuracy is 8 percentage points.
- Multi-phase sampling improves energy savings on both the mobile and server systems. On average—across all learners and systems—this technique produces energy that is 26% closer to optimal than the published learners for system resource allocation and energy management [17, 18, 47].

Additionally, our data support the following observations:

- While increasing learning accuracy generally reduces energy, *there is a point of diminishing returns.*
- Thus, even though the generative model improves even the best prior learner's accuracy, *the continued accuracy improvement does not reduce energy.*
- Because multi-phase sampling biases the learner towards optimal configurations it *reduces overall accuracy, yet significantly improving energy savings.*

This study is strong evidence that after achieving a certain level of accuracy, it is no longer profitable for systems researchers to improve learning systems *without accounting for the structure*—i.e.*, the geometry of optimal tradeoffs for the systems problem to be solved.* In our example of meeting latency requirements with minimal energy, the learners only need to produce accurate results for the configurations on the frontier of optimal performance and power tradeoffs. Once we have separated the optimal configurations, further accuracy improvements provide no additional energy savings and simply waste resources learning for no advantage. This problem is exacerbated because for any one given application most points are not on the optimal frontier. Thus, optimizing for learning accuracy improves predictions for points that are not practically useful. In summary, this work makes the following contributions:

- Proposing a novel generative model for improving predictions of performance and power given scarce data.
- Demonstrating how the structure of constrained optimization problems in computing systems creates asymmetric benefits: accurately predicting optimal configurations is essential while accuracy for other points is of little value.
- Proposing multi-phase sampling for biasing learners towards the useful points.
- Demonstrating the generality of the proposed techniques by using them to further improve existing predictive modeling approaches for allocating resources to meet latency requirements with minimal energy on two different hardware platforms.

2 RELATED WORK AND MOTIVATION

Several studies provide evidence that heuristic-based resource management can break down as the underlying computing systems become more complicated [9, 32, 37, 45, 47, 65]. For example, a popular heuristic is *race-to-idle*, which meets latency constraints by completing computations as fast as possible and then transitioning to a low-power idle state (or sleep state) until the next piece of work is available. Race-to-idle is especially effective on hardware that lacks *energy-proportionality*; *i.e.*, the hardware is most energy-efficient when it runs as fast as possible and slowing down reduces power, but actually increases energy [3, 23, 30]. Recent work demonstrates that newer processor designs—especially heterogeneous processors with a mix of both high-performance and low-power cores—have poor energy behavior under this heuristic [9, 37]. This work demonstrates that if it were possible to accurately model the performance and power tradeoffs of all possible resource assignments, a true optimal resource assignment can reduce energy consumption by large integer factors compared to heuristic methods which often get stuck in local optima. In other words, resource allocation heuristics are brittle and not portable across different hardware devices, as demonstrated in both academic studies [32] and in commercial examples [24].

The increasing complexity of computing systems creates a need for principled approaches to replace heuristics and motivates the study of machine learning and artificial intelligence within resource management systems—especially those concerned with performance and energy. Due to its merits in modeling complicated tradeoffs and avoiding locally optimal resource configurations, machine learning is an attractive alternative to heuristic solutions. We begin this section by finding commonalities in recent work applying ML/AI to computing systems optimization problems. We then show an example of a system/application pair which is difficult

[1] The set of optimal tradeoffs could either be Pareto-optimal, or more strictly, the set of inputs on the lower (or upper) convex hull of the tradeoff space, depending on the specific problem formulation. We simply use the term *optimal* tradeoffs as the ideas in this paper are common to both cases.

to model and demonstrate how important it is for the learner to capture the structure of the systems problem.

2.1 ML/AI-based Systems Management

This section details many recent examples of learning approaches for system management. The common thread for all examples is that the learning system is just a part of the overall solution: the learning components produce models which are used to augment solutions to typical systems problems; *e.g.*, resource allocation [14, 15, 17, 39, 42, 44, 45, 47, 49, 51–53, 61, 63, 64, 67], configuration [1, 11, 19, 21, 25, 40, 41, 43, 50, 55–57, 60, 62, 66], scheduling [17, 18, 31, 54, 58, 59]. Several of these approaches are for *offline* system design—*e.g.,* determining the optimal number and type of cores in a heterogeneous processor [55, 56]—but this paper focuses primarily on building models appropriate for dynamic resource allocation in a fixed processor design.

A common approach to integrating learning into systems management is to use low-level features (*e.g.*, caches misses, instruction per clock) to predict high-level behavior (*e.g.*, throughput, power, latency) [6, 12, 13, 21, 33, 40, 41, 46, 50, 52, 67]. For example, Koala uses regression to transform such features of mobile phones into predictions of application performance and power, which are used to allocate resources to meet performance, energy, or power goals [52]. Similarly, Dubach et al. use low-level features to train a model of a configurable super-scalar that is then used to minimize energy by adapting to application phases [21]. Finally, the Flicker architecture has configurable lanes and uses learned models to configure those lanes in a way that dynamically maximizes performance for a given power budget [50].

An alternative approach learns models of high-level behavior from observing similar applications [17, 18, 45, 47], hardware configurations [58, 63], or both [19, 49]. For example, Paragon [17] and its follow-up, Quasar [18], use the Netflix algorithm [4] to determine efficient schedules for a new application from models of similar applications [17]. The Performance Impact Estimation (PIE) system uses an application's performance on one core type to predict the same application's behavior on a different microarchitecture [58]. Finally, Carat models combinations of applications and mobile devices to determine energy savings opportunities for other combinations [49].

Whether using low- or high-level metrics, data scarcity can be a challenge to deploying learning methods in computer systems. *Generative* methods represent a class of learning techniques that produce new data to improve learning during the training phase [26, 35]. Some generative techniques have been proposed for learning in systems; *e.g.*, generating code to produce better learning systems for optimized compilation [16]. The generative technique proposed in this paper is a novel modification of Gaussian Mixture Models, which is designed specifically for amplifying rare behavior within our training sample.

In all examples, the learning approach is just part of the overall solution. The methodologies in these approaches can be divided into two stages: (1) learn a model optimizing for accuracy, then (2) use that model to solve a systems problem. While all approaches are primarily concerned with solving a systems problem, the general approach appears to be training a learning model to maximize accuracy, then using the learned model to solve a systems problem. The primary contribution of this paper is to show that (1) techniques that improve accuracy (even by quite a lot) do not necessarily produce better solutions to constrained optimization problems arising in computing systems and (2) making the learner aware of the system optimization problem's structure produces better results, even if it does not produce the best accuracy. Thus, we advocate *not training a model for maximum accuracy, but using feedback to improve the ultimate metric of interest—possibly producing less accurate predictions, but better overall systems solutions.*

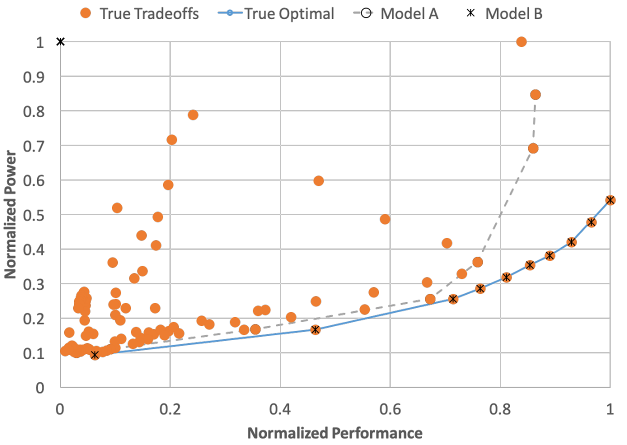


Figure 1: Learning performance/power tradeoffs for SRAD on an ARM big.LITTLE system. The dots show the true tradeoffs. The solid line shows the true optimal frontier. Model A is a learner that is accurate except for the optimal frontier. Model B captures the optimal frontier and gets all other tradeoffs wrong.

2.2 Motivational Example

We demonstrate how high learning accuracy does not guarantee a good systems outcome. We run a real application on a real system and construct two hypothetical learned models. We then use those models in an open-source scheduler ([32]) to meet latency constraints with minimal energy. Both models predict the application's true power and performance tradeoffs as a function of the system configuration. In this example and the rest of the paper, a *configuration* is an allocation of specific hardware resources to an application. The specific resources and allowable settings will vary for different hardware platforms, but they can include things like how many physical cores are available, the clockspeed of those cores, whether hyperthreading is enabled or not, etc. Model A perfectly predicts every point *except* those on the frontier of optimal tradeoffs. Those true optimal tradeoffs are predicted to be just far enough away from their true values to be viewed as non-optimal. The second model perfectly predicts the frontier of optimal tradeoffs, but predicts all other points as having minimal performance and maximum power. The first model gets near perfect accuracy, but high energy; the second model has poor accuracy and optimal energy.

Specifically, we consider the SRAD application (from Rodinia [10]) running on an ARM big.LITTLE system with four LITTLE cores (with 13 clockspeeds) and four big cores (with 19 clockspeeds).

This application is interesting for several reasons. First, it performs well with four cores, but does not scale down. Thus, it is hard to predict performance on four cores from two core samples. Second, SRAD gets high energy efficiency using four LITTLE cores at maximum speed, but when using four big cores at maximum speed thermal throttling drops performance dramatically. Thus, the relationship between clockspeed and performance on LITTLE cores is not a good predictor of that relationship on big cores.

Figure 1 shows the normalized performance (x-axis) and power (y-axis) tradeoffs. Each point is a *configuration* (combination of core allocation and clockspeed) and its position represents its performance and power tradeoffs. The dots show all possible configurations, while the solid line shows the frontier of true optimal tradeoffs (found through exhaustive search). This figure illustrates a key intuition behind the insights presented in this paper: *for any one application most points do not represent optimal tradeoffs.* In this example, only 10 of 128 configurations are on the optimal frontier.

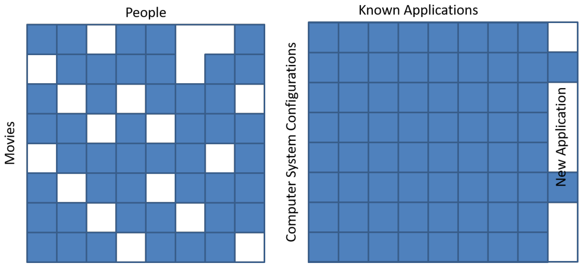
We now construct our first example learner: Model A, which uses the true, measured data for the non-optimal points and then deliberately moves the optimal points just far enough that they will not be selected by the scheduler. We measure Model A's accuracy using *goodness-of-fit* (see Section 6.5) and find it is 99% accurate. We measure energy by feeding this model to a scheduler (from [32]) and having it select combinations of configurations to meet latency requirements and minimize energy. We vary latency requirements across the range of possible behaviors and ensure they are met at least 99% of the time. We then measure the energy and compare to optimal—*i.e.*, that obtained with a perfect model. We find that Model A uses 22% more energy than optimal.

Model B uses the true data for the optimal configurations, all others are assigned minimal performance and maximum power. Model B's goodness-of-fit is essentially 0—not surprising, as most points are inaccurately predicted. All the optimal points are predicted with no error, however, so the energy is the same as optimal.

This example illustrates the most important intuition behind the remainder of the paper. First, high accuracy does not necessarily imply a good systems result. Second, low accuracy does not necessarily mean a bad systems result. These observations demonstrate the problem of *asymmetric benefits*: the system disproportionately benefits from improving accuracy of the small set of configurations on the optimal frontier.

3 LEARNING BY EXAMPLE

This paper builds off prior systems work that *learns by example* [17, 18, 47, 49, 58]. The Paragon project applies this idea to resource management in cloud computing [17]. Paragon takes inspiration from *recommender systems*, specifically the solution to the Netflix challenge [4]. The Netflix approach learns movie recommendations by finding people with similar taste and using those similarities to predict how people would respond to new movies—*i.e.*, it recommends movies based on scores for common movies. More formally, the problem is structured as a matrix (shown in Figure 2a) where each row represents a movie, each column is a person and the entry at a particular row and column is the person's numerical rating for that movie. Many entries are missing, and the learner's job is to estimate them. The intuition is that if two people have similar ratings for some common movies, they will likely have similar ratings for movies that one person has not seen.



(a) Movie Recommendations **(b) Computer Systems**

Figure 2: Matrix Formulation of Learning by Example. Shaded regions represent known data, while blank entries represent missing data. In the movie recommendations system, the rows are movies and the columns are users. Each entry represents a user's score for a given movie; many entries will be empty and the learner's goal is to use known scores to fill in the missing entries. In the computer systems example, the rows are resource configurations and the columns are applications. Each entry represents an application's performance (or power) for a given resource assignment. In this case, most entries are known from running common benchmarks. When a new application arrives, the learner's goal is to sample the new application in a small number of configurations and fill in the missing entries. In that way, the learner uses the known applications' responses to resources to predict the new application's response to the same resources. In this problem formulation, learning techniques that work well for recommender systems can be easily adapted to computer system resource management.

Paragon shows that this structure can be applied to computer systems where we have a configurable computer system and many benchmark applications [17]. Given some new, unseen application, we want to sample a small number of configurations and estimate the behavior in all others. Having done so, we can use those estimates to perform configuration, scheduling or resource allocation optimally. As shown in Figure 2b now each row is a system configuration (*e.g.*, an assignment of cores and clockspeed to an application) and each column is an application. In this case, we may have complete information about some set of benchmarks, but incomplete information about the new application that we can only sample in a small number of configurations.

The main benefit of this approach is generality. The configurations can be quite broad, including assignment of resources to an application in a server [47] or mobile system [45], assignment of a request to a node in a heterogeneous data center [17], or a combination of resource assignment and application co-scheduling in the data center [18]. These techniques can be applied to a broad range of computer systems without deep knowledge of the underlying architecture, but simply listing what configurations are available.

Two downsides of this approach are mentioned in the introduction: scarce data and asymmetric benefits. The effort needed to fill in the known data is much larger for computer systems than it is for making movie recommendations, meaning computer systems must work with scarce data. Also, for recommender systems all data points are equally useful—there is no reason to favor one

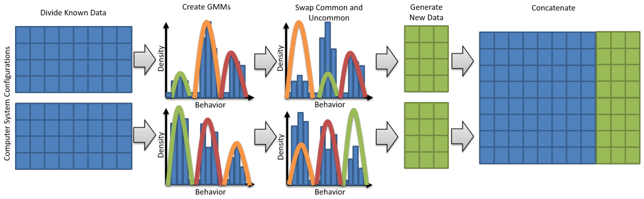


Figure 3: Workflow of proposed generative model.

movie over another. In computer systems, however, typically only a handful of configurations are actually useful, and Figure 1 is a concrete example of this asymmetric response. The vast majority of configurations do not fall on the frontier of optimal tradeoffs and their estimations can be highly inaccurate without affecting the optimality of the computer system.

In Section 4 we present a technique to generate representative data and deal with the challenge of scarce data. Section 5 shows how to divide a sample budget to bias the learner towards those configurations that are most important for the system problem to be ultimately solved.

4 GENERATING DATA FOR ACCURACY

We describe a general way to improve the accuracy of learning by example for computing systems (as formulated in the previous section). Specifically, we use the statistical properties of the known data to generate new "known" data. While generating data is easy (*e.g.*, it could be trivially done with a random number generator), the challenge is to generate data that is both different from the known applications and yet still realistic. The generated data must be different to increase the learner's accuracy when it encounters an application with new behavior. The generated data must be realistic to ensure it captures some plausible behavior that might be exhibited by an unseen application.

To generate different, yet plausible behavior, we propose the use of a Gaussian Mixture Model (GMM) as part of the workflow illustrated in Figure 3. We first divide the data set into disjoint *chunks*. We then use Gaussian mixture models (GMMs) to capture the density of different behaviors in those chunks. A GMM is a weighted sum of Gaussians. A standard GMM would capture the data in our known data set. To generate new data, we swap the highest and lowest weights. This swap amplifies behavior that was present, but rare, in our original data set, while damping the impact of common behavior. The intuition is that this swap should meet the challenge of generating different, but plausible data by amplifying the importance of existent, but rare behavior. We then use this modified GMM to generate new data, represented as extra columns in the matrix formulation of Figure 2b. This larger matrix is trivially compatible with existing matrix completion algorithms.

We give a brief overview of GMMs (which can be skipped for familiar readers). We then provide more detail on how to use a GMM to generate realistic, but diverse data.

4.1 GMM Overview

When analyzing a data set, a common assumption is that each observation comes from a Gaussian distribution, but assuming a single distribution is restrictive and may not make intuitive sense. Alternatively, we can model each observation as being drawn from a finite set—or *mixture*—of models. For example, consider the height of an adult. Since men are typically taller than women, we capture height as a mixture of two components. If we randomly choose an adult, there is a 50% chance of choosing a man or woman, and these proportions are called *mixture proportions/weights*. In this example, the model is a weighted combination of two Gaussian distributions—hence, the name Gaussian mixture model (GMM).

In a GMM, each observation is generated by first choosing one of the Gaussians, and then sampling from it. Given N data points $\mathbf{x}_i \in \mathbb{R}^D, i = 1, \dots, N$, a GMM assumes K components, where the proportion (or weight) of the k-th component is w_k. Notice that the proportions/weights represent the probability that x_i belongs to the k-th component. Thus:

$$p(\mathbf{x}_i) = \sum_{k=1}^{K} w_k g(\mathbf{x}_i | \mu_k, \Sigma_k) \tag{1}$$

where $\mathbf{x}_i$ is the observation, w_k is the mixture weight, and $g(\mathbf{x}_i|\mu_k, \Sigma_k)$ is the component Gaussian. Each component Gaussian is a D-variate Gaussian function of the form

$$g(\mathbf{x}_i|\mu_k, \Sigma_k) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma_i|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu_k)^\top \Sigma_i^{-1} (\mathbf{x}_i - \mu_k) \right),$$

where the mixture weights are non-negative and sum to one.

The GMM is parameterized by mean μ_k, co-variance Σ_k and mixture weights w_k from all components, which cannot be written in closed form. Therefore, the Expectation Maximization algorithm is used to find these values [28].

4.2 Generating Data with a GMM

We now address the scarce data challenge with a generative model based on a GMM. Given a data matrix where rows are configurations and columns are applications, we take the following steps.

Divide. The data matrix is split into smaller chunks according to their configurations. The data within each chunk should have similar distribution. For example, the rows with same number of cores can belong to the same chunk. The chunk size can be determined by any clustering algorithm such as k-means [2] or BSCAN [22]. Given the known configuration distribution, we use k-means clustering since it allows us to determine k—*i.e.,* the number of chunks—by incorporating this prior knowledge.

Learn. For each chunk, fit a GMM to obtain each mixture component. The number of components is decided by cross validation: the data matrix is split into training and validation sets, and different numbers of components are selected to find the best based on the corresponding training and validation likelihood values.

Swap. For each chunk, find the components with minimum and maximum proportions (weights) and swap those extreme values to create a new GMM. This step is crucial since it aims to increase data variations. Intuitively, a component with maximal proportion has actually been reflected substantially from the original data, whereas a component with minimum proportion represents rare behavior. As such, the data generated from this new GMM should amplify the data that have little influence in the original data.

Generate. For each chunk, generate new data based on the GMM with with new mixture proportions from the above step.

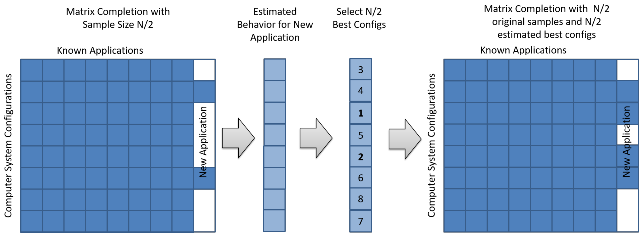

Matrix Completion with Sample Size N/2
Known Applications
Computer System Configurations
New Application
Estimated Behavior for New Application
Select N/2 Best Configs
3
4
1
5
2
6
8
7
Matrix Completion with N/2 original samples and N/2 estimated best configs
Known Applications
Computer System Configurations
New Application

Figure 4: Workflow of proposed multi-phase sampling scheme.

Concatenate. Append the new data chunks with the original data to complete the process.

4.3 Discussion and Limitations

We propose a novel modification of existing GMM methods that amplifies rare behavior in our training set. We do not use the GMM to make predictions itself, however, because GMMs are not good tools for predicting unseen behavior. Rather, their value is in characterizing existing data and especially for finding sub-populations within a larger population. Our novelty is using this method to find rare behavior and then changing the weights to amplify the rare behavior, generating data that is realistic, but different from the measured data. This process diversifies the training set (without additional measurement) and improves learning accuracy.

There are limitations to this approach, of course. The proposed method will make it more likely that the learners pick up on rare behavior in the original data set. If the test data has behavior that is non-existent in the training set, then this approach is not expected to significantly improve accuracy. In future work, this approach could perhaps be improved by generating completely new data that is still realistic. One approach might be capturing existing behavior and "phase shifting" it so that trends remain, but the peak behaviors occur at different configurations that never exhibit peak behavior in the original training set.

5 MULTI-PHASE SAMPLING

Figure 4 illustrates multi-phase sampling, which biases the learner towards the most important points for the system optimization problem that we ultimately care about. We assume the same setup as the prior sections: there is a known body of applications for which we have performance and power data in all configurations and the goal is to observe a new application in a small number of configurations and then predict the behavior in all other configurations. Configurations are again an assignment of system resources—*e.g.,* a specific allocation of cores, core types, and clockspeeds—to an application.

In the first phase, given a sampling budget N and a new, unknown application, we take $N/2$ samples. Each learning algorithm can have its own sampling strategy for this phase and we use the sampling strategies established in the literature for each learner studied. After obtaining the sampled configurations, we run each learner to get an initial estimation of performance and power for the target application. With these estimations, we compute the estimated energy efficiency for each configuration as:

$$\text{efficiency} = \frac{\text{estimated performance}}{\text{estimated power}}. \tag{2}$$

```
Input: Known and unknown applications, sampling budget $N$.

while $True$ do
    Phase-1:
        • Sample half of the budget $N/2$ configurations.
        • Run learner to get an initial estimation.
        • Rank configurations by estimated energy efficiency.
    Phase-2:
        • Sample the $N/2$ most energy efficient configs.
        • Run learner again to obtain the final estimation.
end while
Output: Estimation of performance and power.
```

Algorithm 1: Multi-phase sampling approach.

Then, we rank the unseen application's configurations in terms of their energy efficiency.

In the second phase, we collect additional $N/2$ samples from the most energy efficient configurations. We then use all N samples to run the learner again to obtain the final performance and power estimates for all configurations for the new application.

The intuition behind this approach is that the $N/2$ configurations sampled in Phase 2 will be biased towards the optimal frontier of performance/power tradeoffs. Thus, Phase 2's learning should favor these points (where the sampling concentrated) compared to traditional approaches.

More formally, meeting latency constraints with minimal energy can be written as a linear optimization problem where the decision variables are the amount of time to spend in any configuration [37]. Furthermore, the optimal solution to this problem will only consider configurations that appear on the optimal frontier of the performance/power tradeoff space (as illustrated in Figure 1). Thus, the first phase of the proposed approach is designed to create an initial estimate of the optimal frontier and, therefore, an estimate of the small number of states that may be used in an optimal solution. We rank configurations by energy efficiency because we do not know the true optimal frontier, but we do know that the points on the frontier must be the most energy efficient. This two-phase approach we evaluate here could be extended to have arbitrarily many stages, at a cost of additional overhead for each stage. It is also possible to extend the initial phases to use metrics other than energy efficiency. For example, it may produce better results if we used a metric that balanced energy efficiency with an attempt to spread the sampled configurations out across the range of possible performance. We leave these explorations for future work.

6 EXPERIMENTAL SETUP

Our goal is to show: (1) the generative model improves learning accuracy while (2) multi-phase sampling improves energy. Furthermore, we want to demonstrate that these two results generalize to multiple systems, applications, and learners.

6.1 Systems

Our mobile device is an ODROID-XU3 with a Samsung Exynos 5 Octa processor, based on an ARM big.LITTLE architecture, running Ubuntu 14.04. The 4 big cores support 19 clockspeeds, the 4 LITTLE ones have 13. Thus the mobile configurations are combinations of cores, core types, and clockspeed for the cores.

The server is a dual-socket Linux 3.2.0 system with two Intel Xeon E5-2690 processors, supporting hyperthreads and TurboBoost. Each socket has 8 cores/16 hyperthreads and a 20 MB last-level cache. Because the server system is dual socket, it also has two memory controllers. Therefore, configurations on this system represent a combination of the number of sockets, the cores per socket, whether hyperthreads are used, the clockspeed for each socket, and the number of memory controllers. We allow the learners to use TurboBoost, but it is seen—through the `cpufrequtils` package—as just the highest clockspeed available.

Through their presentation of different resources and different resource types, these two systems stress the learners' abilities to handle a wide variety of configurations. We only consider systems configurations and not application-level ones. In other words, the configurations are assignments of available system resources to an application. For example, on the mobile system a configuration is an assignment of big or LITTLE cores, plus the clockspeed of those cores. On the server system a configuration is a number of sockets, memory controllers, cores per socket, hyperthreads (on or off), and clockspeed.

6.2 Applications

We test a variety of applications on both the mobile and server systems using benchmarks drawn from Parsec [5], Rodinia [10], ParMiBench [34], and MineBench [48]. Parsec and Rodinia contain a number of general purpose workloads. ParMiBench contains multithreaded versions of the embedded MiBench benchmarks [29], and represent workloads common to mobile computing. MineBench contains ML and data mining workloads that represent analytics problems for server systems. All applications are multi-threaded. We use four threads on the mobile and thirty-two threads on the server system. We choose four threads for mobile because the big and LITTLE clusters on our platform each have four cores and the use of four threads stresses the learners' abilities to determine when to migrate between clusters. We strike a balance between achieving some overlap of the application sets on each processor type and stressing the different use cases for each. The primary limiting factor on application testing is establishing ground truth to evaluate the learners, which is done by running each application in all possible configurations for all inputs. For many applications it takes days to establish ground truth on mobile (and never less than hours).

On mobile we use a variety of typical mobile and embedded workloads including video encoding (`x264`), route planning `bfs`, and encryption `sha`. We also use some more computationally intensive tasks which are not typically run on mobile systems today, but stress the learning algorithms and demonstrate workloads that will likely be pushed to mobile (or the edge) in the near future. These include advanced signal processing (`lud`), image processing (`srad`), video analytics (`bodytrack`), machine learning (`backprop` and `kmeans`) . The server system includes a number of exemplary workloads like data analytics (`apr, btree, kmeans, svmrfe`, and many others). We also include a search webserver (`swish++`) and some scientific computing benchmarks (`cfd, nn`).

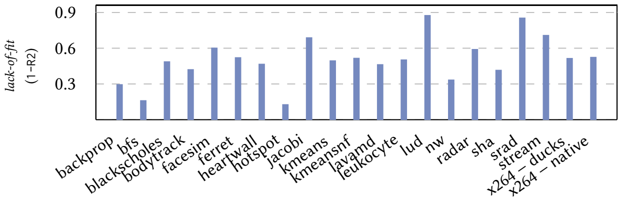


Figure 5: ***Lack-of-fit*** **for performance vs clock-speed on mobile.**

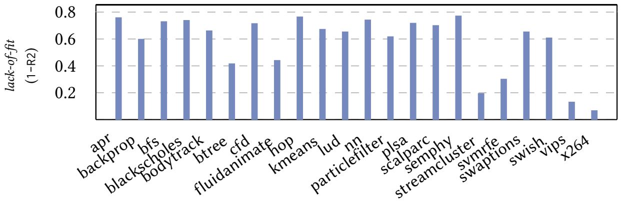


Figure 6: ***Lack-of-fit*** **for performance vs clock-speed on server.**

In our experiments we consider only a single, multi-threaded application at a time. Whether on mobile or server, our goal is to meet an application latency target with minimal energy by configuring system resources appropriately. Given the latency requirements, we currently consider a single application at a time and leave multi-programmed scheduling for future work.

6.3 Application and System Diversity

To demonstrate the diversity in workloads we compute a simple linear regression for each application's performance and clockspeed on both systems. The intuition is that if an application scales linearly with clockspeed, it is likely compute-intensive, while no scaling would indicate memory intensity. We quantify this intuition using *lack-of-fit*, which is simply $1 - R^2$, where R is the correlation coefficient for this simple linear model. Low numbers mean the linear model fits well, indicating compute-intensive workloads. High numbers indicate the opposite.

Figures 5 and 6 show the results for the applications on mobile and server, respectively. The figures show a wide variety of behavior from very compute-intensive to very memory-intensive, with several examples falling in between these extremes.

In addition, we demonstrate the diversity of benchmarks and the difference between machines by finding the true optimal performance/power tradeoffs for all benchmarks (on average across all inputs) on both systems. Figures 7 and 8 show the frequency with which each configuration appears on the frontier of optimal tradeoffs. We note that 63 (out of 128 total) configurations appear on at least one mobile application's optimal frontier, while 96 (out of 1024) configurations appear on at least one server application's optimal frontier. The shape of these distributions is further evidence of the different qualities of the architectures. Mobile has two clusters of configurations that appear on the optimal frontier while the server has a more uniform distribution.

We further demonstrate the difficulty of the learning problem by measuring the number of configurations that appear on the optimal frontier for each application on each system. Figure 9 shows this data for the mobile while Figure 10 shows the data for the server. The largest number of configurations for any application on mobile

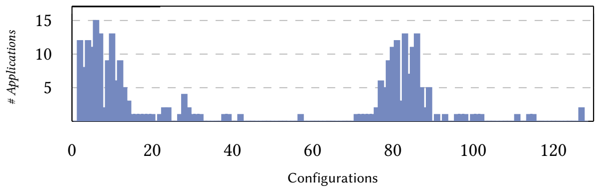


Figure 7: Distribution of optimal configurations for mobile.

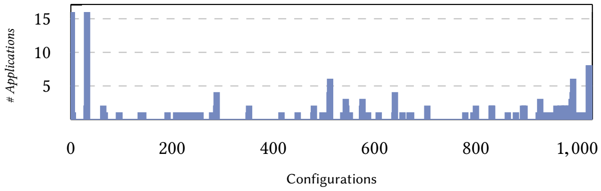


Figure 8: Distribution of optimal configurations for server.

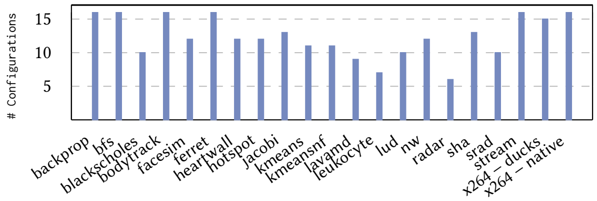


Figure 9: Optimal configuration count for mobile applications.

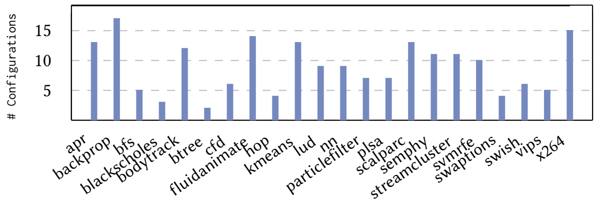


Figure 10: Optimal configuration count for server applications.

is 16, 15 for the server. This data indicates that while a large number of configurations appear on *some* optimal frontier, for any *one* application just a fraction of this total are relevant.

In summary, this data shows that these two systems will stress the learning algorithms' generality. The applications exhibit a wide range of behaviors, the two systems have different behavior, and of the large number of possible configurations, only a small number appear on any one application's optimal frontier.

6.4 Learning Models Studied

We evaluate our proposed framework on five following learning models, where the first four are matrix completion based algorithms, and the last one is a Bayesian approach:

(1) *MCGD*: an approximate matrix completion algorithm solved via gradient descent [36].
(2) *MCMF*: a matrix completion algorithm solved by factorizing the matrix into bi-linear form [38].
(3) *Nuclear*: an exact matrix completion algorithm by minimizing the matrix's nuclear norm [7, 8]. This technique has been demonstrated to provide good systems outcomes in scheduling for data centers [17, 18].
(4) *WNNM*: an exact matrix completion algorithm by minimizing the matrix's weighted nuclear norm [27].
(5) *HBM*: a hierarchical Bayesian model for recovering optimal performance/power tradeoffs [47].

To the best of our knowledge, this is the first comprehensive evaluation of matrix completion algorithms for systems.

For each learner, we evaluate four variations:

(1) *Vanilla*: the vanilla framework that only uses the basic learners to estimate the missing entries for speed/power.
(2) *GM*: uses the generative model to augment the data matrix and then apply the learners to perform estimation.
(3) *MP*: use multi-phase sampling framework to perform estimation.
(4) *MP-GM*: use generative model to augment the data matrix and then apply the multi-phase sampling framework to perform estimation.

6.5 Evaluation Metrics

For each application, we evaluate prediction accuracy by using adjusted R^2 [20], which measures the *goodness-of-fit* between the ground-truth $\mathbf{y}$ and estimated value $\hat{\mathbf{y}}$:

$$R^2 = \max\left(0, 1 - \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}{\|\mathbf{y} - \bar{\mathbf{y}}\|^2}\right), \quad (3)$$

where n is the number of configurations (length of vector $\mathbf{y}$) and $\bar{\mathbf{y}}$ is the mean vector of $\mathbf{y}$.

We evaluate energy savings by running every application in every resource configuration. To compare across applications, we normalize energy:

$$\texttt{Normalized energy} = 100\% * \left(\frac{\mathrm{e_{measured}}}{\mathrm{e_{optimal}}} - 1\right), \quad (4)$$

where $\mathrm{e_{measured}}$ is measured energy and $\mathrm{e_{optimal}}$ is the optimal energy. This metric shows the percentage of energy over optimal by subtracting 1.

6.6 Evaluation Methodology

We collect the true performance and power for all applications in all configurations for both the mobile and server systems. All accuracy and energy evaluations are all done with respect to this data, which is collected through exhaustive measurement. When evaluating the accuracy and energy savings, we use *leave-one-out* cross validation. To test application i, we form a set of all other applications excluding i, so all other applications form the full columns of the data matrix from Figure 2b. We then sample i in several configurations to form the partial, last column from Figure 2b.

We then use the algorithms mentioned above with combinations of our proposed techniques to estimate each application's behavior in unsampled configurations. These estimations are passed to an open-source resource allocator, which assigns resources to meet goals [32]. For each application we vary the performance goal to require from 10-95% utilization to meet the goal in the worst case. For this work, we assume we know the worst case timing for any input and application if processed with all available resources. Thus, we are assured (and we manually verify) that the scheduler will meet the performance requirements and we focus on the energy savings. This methodology prevents a learner from "cheating" by passing the scheduler a model that reduces energy by delivering low performance.

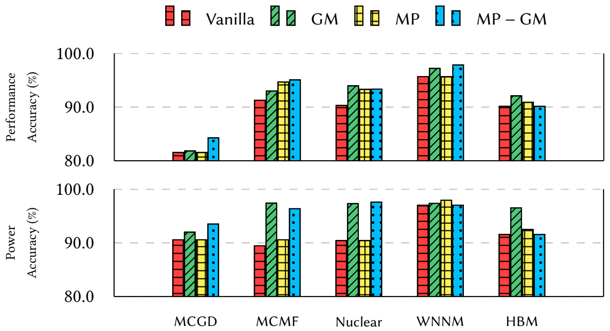


Figure 11: Performance and power accuracy on mobile (higher is better).

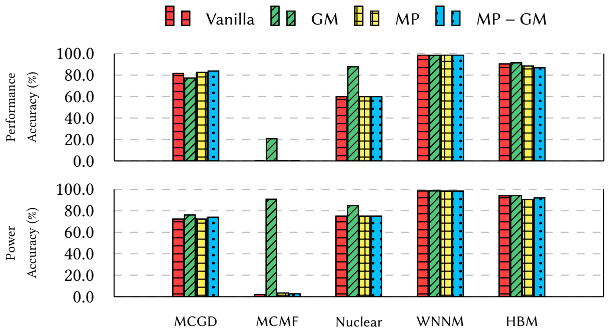


Figure 12: Performance and power accuracy on server (higher is better).

7 EVALUATION

We start this section with some high-level summary results. We then present detailed results for learning accuracy and energy. We then perform a sensitivity analysis to show how the learners behave as a function of their sampling budget. We finally evaluate the overhead of all techniques.

7.1 Summary Results

7.1.1 Accuracy. We show the estimation accuracy for performance and power on both mobile and server in Figure 11 and Figure 12. The x-axes show the learner, while the y-axes show the average accuracy across all applications. There is a bar for each variation of: *Vanilla*, *GM*, *MP*, and *MP-GM*.

The results for the vanilla Nuclear and HBM learners are basically equivalent to published work using those same learners in similar scenarios [45, 47], which gives us confidence that our improvements are representative. Indeed, we see that the GM method greatly improves accuracy for all learners on mobile. On server, GM improves accuracy for MCMF from, effectively, zero to something non-zero. multi-phase sampling does not, in general improve accuracy. Somewhat counter-intuitively, the combination of GM and multi-phase sampling also does not improve accuracy. Table 1 shows the average improvement in percentage points for each technique. From this table it is clear that GM has a large effect on accuracy—more than 8 percentage points on average—while the other techniques have little effect.

Table 1: Average percentage points of accuracy improvement.

		GM	MP	MP – GM
Mobile	Performance	1.8	1.4	2.3
	Power	4.3	0.6	3.4
Server	Performance	9.0	−0.2	−0.3
	Power	20.5	−0.4	0.1
Average		8.9	0.4	1.4

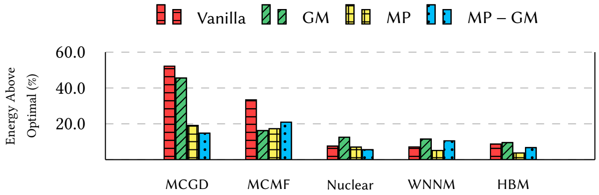


Figure 13: Energy compared to optimal on mobile (lower is better).

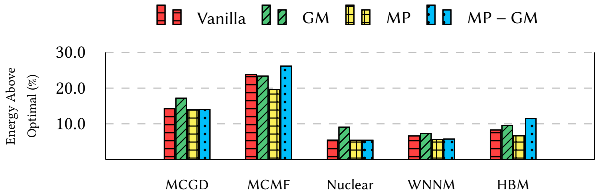


Figure 14: Energy compared to optimal on server (lower is better).

Weighted Nuclear Norm Minimization is the best vanilla learner, but—to our knowledge—this technique has not been applied to computer systems optimization before. However, the proposed GM method improves even this best-of-class technique. In addition, the GM method brings other techniques, which have been applied to systems (Nuclear [17, 18] and HBM [47]) to the similar levels of accuracy. *We conclude that GM can have a dramatic effect on the accuracy of performance and power predictions.*

7.1.2 Energy. Figures 13 and 14 show the average energy over optimal (y-axis) for each technique (x-axis) on the mobile and server.

These figures show two key points: (1) despite the GM method's much higher accuracy, it often has a higher energy than the vanilla learner and (2) even though it is generally lower in accuracy, the multi-phase method has lower energy than the vanilla methods. These trends are starkly visible in Table 2, which shows the improvement in energy compared to the vanilla learners. This table compares the energy of the augmented method (GM, MP, MP-GM) to the energy of the vanilla method and shows how much closer the augmented method is to idle. The results are expressed as a percentage so that we can compare the effects of poor learners to good ones. Negative numbers show that the energy is worse using the method than just using the plain learner. The table shows that the multi-phase sampling method gets much closer to optimal than GM or even the combination of GM and MP. In fact, on average, the GM method has a substantial negative effect on energy.

Table 2: Average energy improvement. (Higher is better).

	GM	MP	MP – GM
Mobile	−14%	41%	22%
Server	−22%	11%	−6.5%

7.2 Detailed Accuracy Results

We find that power is, generally, much easier to predict than performance. Therefore we present only performance accuracy detailed

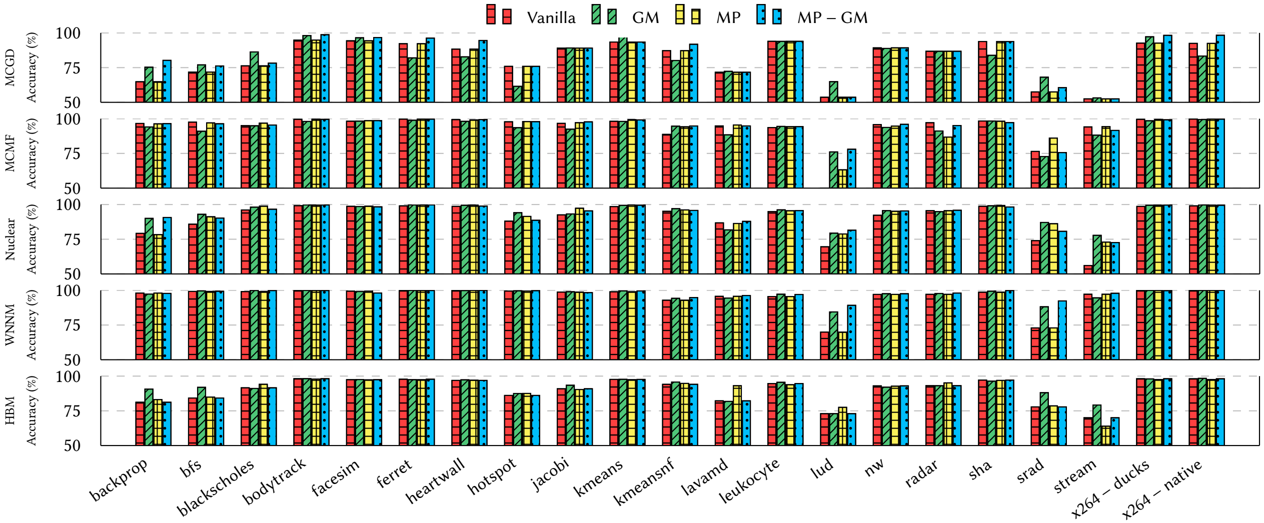


Figure 15: Comparison of performance accuracy per application for mobile system (higher is better).

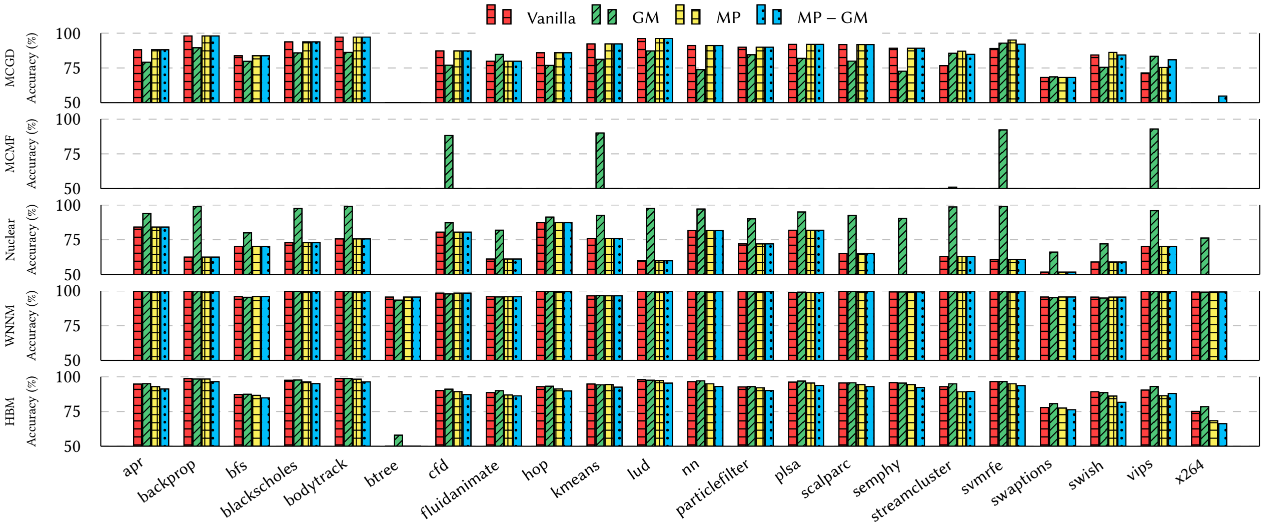


Figure 16: Comparison of performance accuracy per application for server system (higher is better).

results to save space. Figures 15 and 16 show the performance estimation accuracy for all 5 learning algorithms on the mobile and server system, respectively. The x-axis shows each benchmark and the y-axis shows the accuracy. Each benchmark has four bars, one for each of Vanilla, GM, MP, and MP-GM.

These figures show the prediction accuracy is over 80% for all learners on mobile, with a wider range on server. In particular, MCMF and Nuclear have lower prediction accuracy than the other learners for both performance and power on the server. This finding is consistent with the characterizations in Figures 9 and 10 where the mobile data is clustered, but the server data is evenly distributed.

7.3 Detailed Energy Results

As can be seen in the accuracy results, some applications are much easier to predict than others. To save space we show just the energy savings for the hardest to estimate applications, which we define as those that have lack-of-fit greater than 0.5 for mobile and greater than 0.6 for server. Figures 17 and 18 show these energy results.

These results not only provide detail showing the energy behavior for the toughest applications, they also demonstrate that multi-phase sampling is robust to these difficult applications. While the energy for the worst applications is, not surprisingly, higher than the average energy, multi-phase sampling still saves considerable energy. For example, the average energy over optimal for HBM-MP on mobile is 3.7%, while the average energy for these hardest applications is 4.3%.

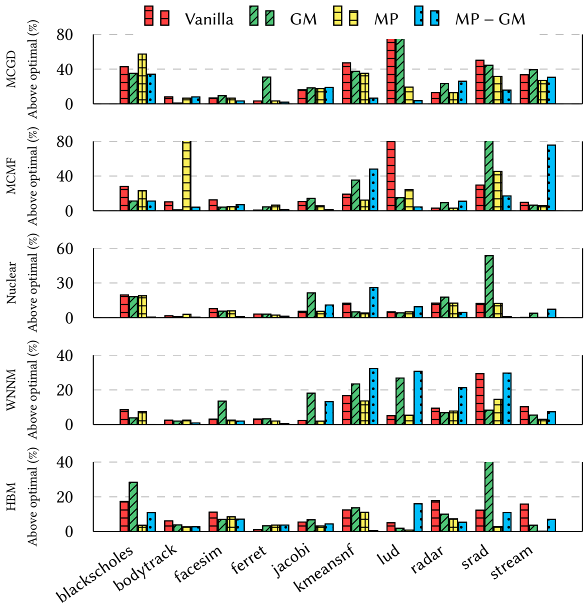


Figure 17: Energy savings per application for mobile system (lower is better).

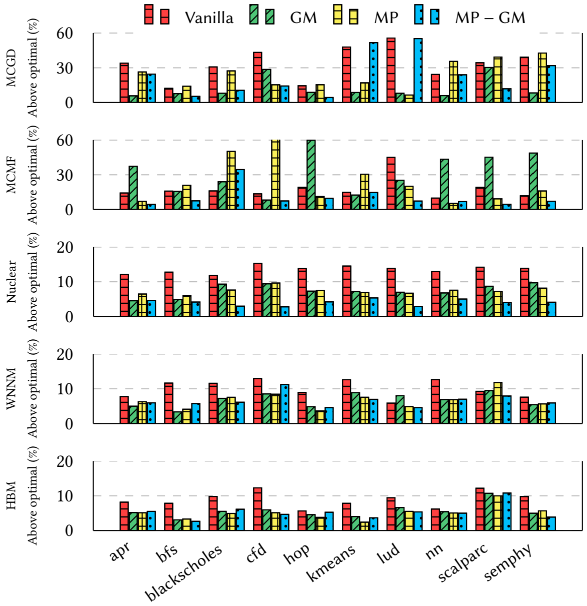


Figure 18: Energy savings per application for server system (lower is better).

7.4 Accuracy and Energy for Best Learners

As mentioned above, MCGD and MCMF are clearly weaker learners than the other three. These results are not terribly surprising as Nuclear [17, 18] and HBM [45, 47] have both been used in prior systems work, while WNNM only recently appeared in the ML literature [27]. In this section we evaluate the accuracy and energy results considering only these, best-in-class learners and omitting MCGD and MCMF.

When removing MCGD and MCMF the numbers change, but the broad conclusions are even stronger. Specifically, GM improves performance/power accuracy by on mobile and server as shown in Table 3. The energy savings for GM are worse when removing MCGD and MCMF as shown in Table 4. MP's accuracy is almost the same, but the energy improvements are 30% and 12% on mobile and server, respectively. The MP-GM results are not significantly different in this scenario. Thus, when focusing on the most sophisticated learners, the accuracy improvements are slightly smaller. MP's energy improvements are smaller in magnitude, but relatively much more significant. We believe these results support the conclusion that improving state-of-the-art learners' accuracy does not improve systems outcomes—for the constrained optimization problems explored in this paper—but accounting for the problem structure (in this case, by modifying the sampling procedure) does improve the system outcome.

Table 3: Accuracy improvement for best learners.

		GM	MP	MP – GM
Mobile	Performance	2.4	1.2	1.7
	Power	4.1	0.6	2.4
Server	Performance	9.6	−0.6	−1.2
	Power	3.4	−1.2	−0.7
Average		4.8	0.0	0.6

Table 4: Energy improvement for the best learners.

	GM	MP	MP – GM
Mobile	−45%	31%	0.0%
Server	−31%	12%	−8.3%

7.5 Sensitivity to Sample Size

7.5.1 Sample-Complexity Results. One of the key parameters of all learners is the number of samples it must measure to produce accurate estimates. All of the above measurements were taken with the mobile and server systems configured to sample 20 and 120 configurations, respectively. In Figure 19, we show the accuracy (averaged over all benchmarks) for performance (power is omitted for space) estimation as a function of sample size. Since our accuracy measurement is adjusted R^2, it is not surprising to see zeros in MCGD and MCMF on server for small sample sizes. We also observe that GM and MP-GM always perform better than Vanilla and MP across different sample sizes, which justifies the ability of generative models to improve accuracy.

7.5.2 Energy Savings for Reduced Samples. We now explore energy savings for reduced samples. While it is not feasible to measure energy savings at every possible sample size, we reduce the sample sizes by half and rerun all the above experiments. The results for mobile are very similar to the results already presented: WNNM-GM has highest accuracy, while HBM-MP has the lowest energy.

Table 5: Average energy improvement with reduced samples.

	GM	MP	MP – GM
Average Case	17%	15%	2%
Worst Apps	15%	21%	−8%

Table 5 shows the results for the server with half the sample size. The table shows the energy improvement relative to vanilla for each of our proposed techniques. There is one row for the average case and another for the worst applications (again those that had the worst energy for WNNM-Vanilla).

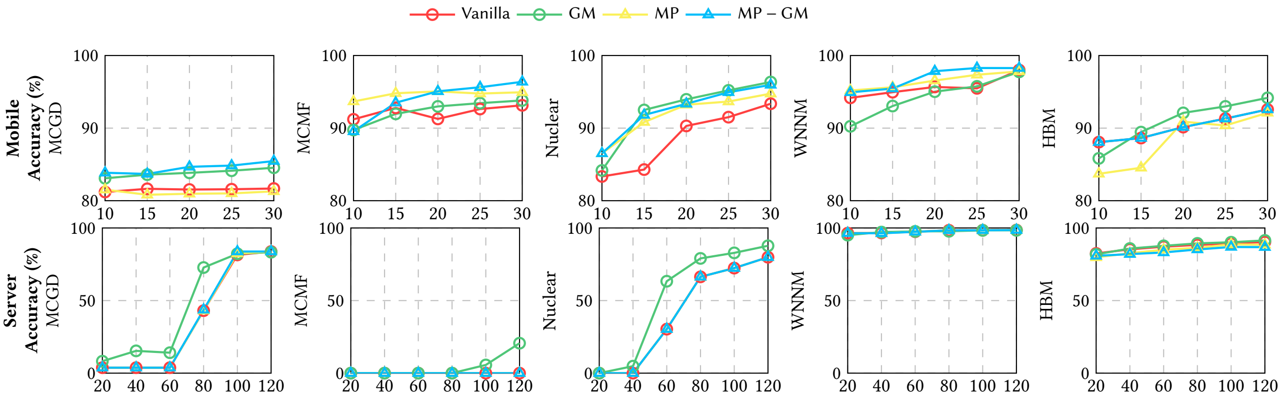


Figure 19: Sensitivity analysis of performance accuracy for mobile and server systems. The x-axis is the sample size.

These results illustrate a complicated aspect of our study. The sample complexity plots show that, while all learners have a point of diminishing returns, they occur at different locations for different learners. In this section, we are investigating sample sizes where some learners have sufficient samples and others do not. At this point, for example, GM really improve MCGD's energy because it was not at the point of diminishing returns. Other learners are beyond that point though, and even in this case we see that multi-phase sampling significantly improves energy on the hardest apps. These results indicate that multi-phase sampling is robust to reduced sample sizes.

7.6 Overhead

We collect the overhead for all combinations of systems, learners, and proposed augmentations. These overheads are listed in Table 6 and Table 7 for the mobile and server, respectively. The tables show the time amortized over all configurations. These results show that the GM and multi-phase methods add some overhead compared to the vanilla learning systems, but these results are not surprising and we believe the overhead is tolerable for the benefits.

For example, the GM method is expensive because the EM algorithm on which it relies is quite expensive. However, GM creates new columns in the known data matrix. An alternative to approach is to find a new application to add to the known data by exhaustively characterizing it in all configurations. On the systems we study, this exhaustive characterization takes hours up to a day, and there is no guarantee that the new application will exhibit significantly different behavior than the common case in the data set. Compared this approach of manually adding data to the known applications, GM is orders of magnitude faster and provides better statistical guarantees that it will generate useful data.

Multi-phase sampling also adds overhead because it runs these fairly computationally expensive learners twice, once in each phase. One direction of future work is to look at further optimizations to this approach. Perhaps different learning techniques can be combined to greater effect. For example WNNM is the highest accuracy and less expensive than HBM, so using multi-phase sampling with WNNM in Phase 1 and HBM in Phase 2 might produce better accuracy and energy savings with lower overhead.

Table 6: Learner overhead for mobile system (in ms).

	Vanilla	GM	MP	MP – GM
MCGD	1.1	1.9	2.3	2.9
MCMF	0.1	0.7	0.1	0.7
Nuclear	2.2	3.0	3.4	4.6
WNNM	1.2	2.7	2.3	3.8
HBM	2.5	3.8	4.8	6.1

Table 7: Learner overhead for server system (in ms).

	Vanilla	GM	MP	MP – GM
MCGD	0.9	1.2	1.8	2.0
MCMF	0.1	0.2	0.3	0.2
Nuclear	0.7	0.8	1.0	1.1
WNNM	0.3	0.6	0.6	0.9
HBM	13.2	15.4	19.0	23.9

7.7 Discussion

So far, detailed results have been presented using predictive modeling to assign system-level resources to applications such that latency constraints are met with minimal energy. We have compared existing approaches to this problem to the same approaches augmented with generative and multi-phase enhancements. For this specific resource management problem the results suggest that:

- *There is a point of diminishing returns in applying learning.* The MCGD and MCMF methods are clearly worse (in accuracy and performance) than the other three. However, even the best vanilla learner (WNNM) shows little energy improvement over the others in its class (Nuclear and HBM).
- *The generative model improves accuracy.* It is significant that we can generate data (which has no measurement cost and is many orders of magnitude faster than exhaustively measuring an application) and improve learning accuracy.
- *The multi-phase method improves energy.* By biasing the learner to the configurations (resource allocations) that are likely to be most energy efficient, this approach improves energy consumption, as long as the learning technique to which it is being applied is accurate enough. For example, our initial results show that this technique has significant accuracy savings, but we find that the energy savings can diminish for some learners with reduced sample size.
- *Improving accuracy does not necessarily improve energy consumption.* Because of asymmetric response and diminishing

returns, it is possible to greatly improve accuracy by improving estimations of the configurations that are not on the optimal frontier of performance and power.

- *The systems outcome can be improved without improving the learner's accuracy.* Multi-phase sampling does not improve overall accuracy, but has a significant effect on energy even for the best in class learners.

We expect the same broad behavior for any learners whose output is used to solve a constrained optimization problem in computer systems. The nature of such problems means that, for any application, only a small number of configurations will appear on the polytope of possible optimal solutions.

The key insight from this study is that only the small subset of optimal configurations matter for systems outcomes—improving accuracy for the non-optimal configurations is not helpful, but getting the optimal set correct is essential. For straight optimization problems–without constraints; *e.g.*, find the most energy efficient configuration—the results might not hold. Similarly, if we could build computing systems such that all configurations were on the frontier of optimal tradeoffs for all applications then the results might not hold, as the accuracy of each configuration's predicted behavior would directly affect the solution to the optimization problem.

8 CONCLUSION

As machine learning and AI researchers continue to produce astonishing results, it is natural to simply apply each new learning techniques to computing systems and reap the benefit. This process typically follows an approach of training a learner for maximum accuracy and then deploying it to build a model that some computer management system (*e.g.*, scheduler, configuration management, or resource allocation) can use to improve its system outcomes.

We argue that the above process has reached a point of diminishing returns. Our example from Section 2 shows a hypothetical counter example where a very inaccurate learner produces better systems results than an accurate learner. The key difference between those learners is that one learner is aware of the structure of the systems problem (it is a constrained optimization problem in the performance/power space) and is accurate only for the configurations that affect that structure.

We have shown how to build learners that produce better systems results by acquiring knowledge of the systems problem. In the multiphase learning we propose, the first phase finds the likely most significant points for the systems problem, while the second phase explicitly samples those points. This technique represents one way to incorporate knowledge of the systems problem into training the learner and it leads to empirically better outcomes, even for state-of-the-art systems from the literature.

This work is just one example of how to incorporate more systems knowledge into training learners. While we have applied it to the problem of meeting latency constraints with minimal energy, we believe the ideas would translate to any system that has to balance multiple, competing constraints. We hope this work inspires other systems researchers to consider techniques for incorporating system knowledge into learning solutions.

Acknowledgments

We thank our anonymous shepherd for both patience and the detailed feedback that greatly improved this final version of the paper. We also thank the anonymous reviewers for their helpful feedback and Erik Altman for patiently serving as the point of contact between us and our shepherd. This research is supported by NSF (CCF-1439156, CNS-1526304, CCF-1823032, CNS-1764039). Additional support comes from the Proteus project under the DARPA BRASS program and a DOE Early Career award.

REFERENCES

[1] Jason Ansel, Maciej Pacula, Yee Lok Wong, Cy Chan, Marek Olszewski, Una-May O'Reilly, and Saman Amarasinghe. 2012. Siblingrivalry: online autotuning through local competitions. In *CASES*.

[2] David Arthur and Sergei Vassilvitskii. 2007. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1027–1035. http://dl.acm.org/citation.cfm?id=1283383.1283494

[3] L.A Barroso and U. Holzle. 2007. The Case for Energy-Proportional Computing. *Computer* 40, 12 (Dec 2007), 33–37. https://doi.org/10.1109/MC.2007.443

[4] R. M. Bell, Y. Koren, and C. Volinsky. 2008. *The BellKor 2008 solution to the Netflix Prize.* Technical Report. ATandT Labs.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *PACT*.

[6] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. 2008. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *MICRO*.

[7] J. Cai, E. Candès, and Z. Shen. 2010. A Singular Value Thresholding Algorithm for Matrix Completion. *SIAM Journal on Optimization* 20, 4 (2010), 1956–1982. https://doi.org/10.1137/080738970

[8] Emmanuel J Candès and Benjamin Recht. 2009. Exact matrix completion via convex optimization. *Foundations of Computational mathematics* 9, 6 (2009), 717.

[9] Aaron Carroll and Gernot Heiser. 2013. Mobile Multicores: Use Them or Waste Them. In *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower '13)*. ACM, New York, NY, USA, Article 12, 5 pages. https://doi.org/10.1145/2525526.2525850

[10] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *IISWC*.

[11] Chi-Ou Chen, Ye-Qi Zhuo, Chao-Chun Yeh, Che-Min Lin, and Shih-Wei Liao. 2015. Machine Learning-Based Configuration Parameter Tuning on Hadoop System. In *BigData Congress*.

[12] Jian Chen and Lizy Kurian John. 2011. Predictive coordination of multiple on-chip resources for chip multiprocessors. In *ICS*.

[13] Jian Chen, Lizy Kurian John, and Dimitris Kaseridis. 2011. Modeling Program Resource Demand Using Inherent Program Characteristics. *SIGMETRICS Perform. Eval. Rev.* 39, 1 (June 2011), 1–12.

[14] Seungryul Choi and Donald Yeung. 2006. Learning-Based SMT Processor Resource Distribution via Hill-Climbing. In *ISCA*.

[15] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. 2011. Pack & Cap: adaptive DVFS and thread packing under power caps. In *MICRO*.

[16] Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. 2017. Synthesizing benchmarks for predictive modeling. In *Proceedings of the 2017 International Symposium on Code Generation and Optimization, CGO 2017, Austin, TX, USA, February 4-8, 2017*. 86–99.

[17] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *ASPLOS*.

[18] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware Cluster Management. In *ASPLOS*.

[19] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Fred Chong. 2017. Memory Cocktail Therapy: A General Learning-Based Framework to Optimize Dynamic Tradeoffs in NVM. In *MICRO*.

[20] N.R. Draper and H. Smith. 1998. *Applied regression analysis.* Number v. 1 in Wiley series in probability and statistics: Texts and references section. Wiley.

[21] Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F. P. O'Boyle. 2010. A Predictive Model for Dynamic Microarchitectural Adaptivity Control. In *MICRO*.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231. http://dl.acm.org/citation.cfm?id=3001460.3001507

[23] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. 2007. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual*

international symposium on Computer architecture (ISCA '07). ACM, New York, NY, USA, 13–23. https://doi.org/10.1145/1250662.1250665

[24] Andrei Frumusanu. 2018. Improving the Exynos 9810 Galaxy S9: Part 2 - Catching Up With the Snapdragon. *AnandTech* (April 2018). https://www.anandtech.com/show/12620/improving-the-exynos-9810-galaxy-s9-part-2

[25] Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson. 2009. A case for machine learning to optimize multicore performance. In *First USENIX Workshop on Hot Topics in Parallelism (HotParŠ09)*.

[26] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[27] Shuhang Gu, Qi Xie, Deyu Meng, Wangmeng Zuo, Xiangchu Feng, and Lei Zhang. 2017. Weighted Nuclear Norm Minimization and Its Applications to Low Level Vision. *Int. J. Comput. Vision* 121, 2 (Jan. 2017), 183–208.

[28] Maya R. Gupta and Yihua Chen. 2011. Theory and Use of the EM Algorithm. *Found. Trends Signal Process.* 4, 3 (March 2011), 223–296.

[29] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop (WWC '01)*. IEEE Computer Society, Washington, DC, USA, 3–14. https://doi.org/10.1109/WWC.2001.15

[30] Urs Hoelzle and Luiz Andre Barroso. 2009. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines* (1st ed.). Morgan and Claypool Publishers.

[31] Henry Hoffmann. 2015. JouleGuard: energy guarantees for approximate applications. In *SOSP*.

[32] Connor Imes, David H. K. Kim, Martina Maggio, and Henry Hoffmann. 2015. POET: A Portable Approach to Minimizing Energy Under Soft Real-time Constraints. In *RTAS*.

[33] E. Ipek, O. Mutlu, J. F. MartŠnez, and R. Caruana. 2008. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *ISCA*.

[34] Syed Muhammad Zeeshan Iqbal, Yuchen Liang, and Hakan Grahn. 2010. ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems. *IEEE Comput. Archit. Lett.* 9, 2 (July 2010).

[35] Tony Jebara. 2003. *Machine Learning: Discriminative and Generative (Kluwer International Series in Engineering and Computer Science)*. Kluwer Academic Publishers, Norwell, MA, USA.

[36] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. 2010. Matrix completion from noisy entries. *Journal of Machine Learning Research* 11, Jul (2010), 2057–2078.

[37] David H. K. Kim, Connor Imes, and Henry Hoffmann. 2015. Racing and Pacing to Idle: Theoretical and Empirical Analysis of Energy Optimization Heuristics. In *CPSNA*.

[38] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37. https://doi.org/10.1109/MC.2009.263

[39] B.C. Lee, J. Collins, Hong Wang, and D. Brooks. 2008. CPR: Composable performance regression for scalable multiprocessor models. In *MICRO*.

[40] Benjamin C. Lee and David M. Brooks. 2006. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *ASPLOS*.

[41] Benjamin C. Lee and David M. Brooks. 2010. Applied inference: Case studies in microarchitectural design. *TACO* 7, 2 (2010), 8:1–8:37. https://doi.org/10.1145/1839667.1839670

[42] Benjamin C. Lee, David M. Brooks, Bronis R. de Supinski, Martin Schulz, Karan Singh, and Sally A. McKee. 2007. 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, PPOPP 2007, San Jose, California, USA, March 14-17, 2007*. 249–258.

[43] J. Li and J.F. Martinez. 2006. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *HPCA*.

[44] J. F. Martinez and E. Ipek. 2009. Dynamic Multicore Resource Management: A Machine Learning Approach. *IEEE Micro* 29, 5 (Sept 2009), 8–17. https://doi.org/10.1109/MM.2009.77

[45] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. 2018. CALOREE: Learning Control for Predictable Latency and Low Energy. In *ASPLOS*.

[46] Nikita Mishra, John D. Lafferty, and Henry Hoffmann. 2017. ESP: A Machine Learning Approach to Predicting Application Interference. In *2017 IEEE International Conference on Autonomic Computing, ICAC 2017, Columbus, OH, USA, July 17-21, 2017*. 125–134.

[47] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. 2015. A Probabilistic Graphical Model-based Approach for Minimizing Energy Under Performance Constraints. In *ASPLOS*.

[48] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. 2006. MineBench: A Benchmark Suite for Data Mining Workloads. In *IISWC*.

[49] Adam J. Oliner, Anand P. Iyer, Ion Stoica, Eemil Lagerspetz, and Sasu Tarkoma. 2013. Carat: Collaborative Energy Diagnosis for Mobile Devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys '13)*. ACM, New York, NY, USA, Article 10, 14 pages. https://doi.org/10.1145/2517351.2517354

[50] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, and Christine A. Shoemaker. 2013. Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems. In *ISCA*.

[51] Dmitry Ponomarev, Gurhan Kucuk, and Kanad Ghose. 2001. Reducing Power Requirements of Instruction Scheduling Through Dynamic Allocation of Multiple Datapath Resources. In *MICRO*.

[52] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. 2009. Koala: A Platform for OS-level Power Management. In *EuroSys*.

[53] Srinath Sridharan, Gagan Gupta, and Gurindar S. Sohi. 2013. Holistic Run-time Parallelism Management for Time and Energy Efficiency. In *ICS*.

[54] G. Tesauro. 2007. Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies. *IEEE Internet Computing* 11 (2007). Issue 1.

[55] Erik Tomusk, Christophe Dubach, and Michael F. P. O'Boyle. 2016. Four Metrics to Evaluate Heterogeneous Multicores. *TACO* 12, 4 (2016), 37:1–37:25. https://doi.org/10.1145/2829950

[56] Erik Tomusk, Christophe Dubach, and Michael F. P. O'Boyle. 2016. Selecting Heterogeneous Cores for Diversity. *TACO* 13, 4 (2016), 49:1–49:25. https://doi.org/10.1145/3014165

[57] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. 2017. Automatic Database Management System Tuning Through Large-scale Machine Learning. In *SIGMOD*.

[58] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. 2012. Scheduling Heterogeneous Multi-cores Through Performance Impact Estimation (PIE). In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*. IEEE Computer Society, Washington, DC, USA, 213–224. http://dl.acm.org/citation.cfm?id=2337159.2337184

[59] Jonathan A. Winter, David H. Albonesi, and Christine A. Shoemaker. 2010. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *PACT*.

[60] Weidan Wu and Benjamin C Lee. 2012. Inferred models for dynamic and sparse hardware-software spaces. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*. IEEE, 413–424.

[61] Joshua J. Yi, David J. Lilja, and Douglas M. Hawkins. 2003. A Statistically Rigorous Approach for Improving Simulation Methodology. In *HPCA*.

[62] Nezih Yigitbasi, Theodore L Willke, Guangdeng Liao, and Dick Epema. 2013. Towards machine learning-based auto-tuning of mapreduce. In *MASCOTS*.

[63] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *ASPLOS*.

[64] Xiao Zhang, Rongrong Zhong, Sandhya Dwarkadas, and Kai Shen. 2012. A Flexible Framework for Throttling-Enabled Multicore Management (TEMM). In *ICPP*.

[65] Yanqi Zhou, Henry Hoffmann, and David Wentzlaff. 2016. CASH: Supporting IaaS Customers with a Sub-core Configurable Architecture. In *ISCA*.

[66] Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. BestConfig: tapping the performance potential of systems via automatic configuration tuning. In *SoCC*.

[67] Yuhao Zhu and Vijay Janapa Reddi. 2013. High-performance and energy-efficient mobile web browsing on big/little systems. In *HPCA*.