THE UNIVERSITY OF CHICAGO

LEARNING STRUCTURE FOR COMPUTER SYSTEMS MANAGEMENT

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
YI DING

CHICAGO, ILLINOIS
DECEMBER 2020

Dedicated to my parrents, for always believing in me.

To improve is to change; to be perfect is to change often.

— Winston Churchill

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I have to begin by thanking my marvelous, wise, patient, and kind advisor Hank Hoffmann, who has been extremely supportive of my unusual PhD journey. I met Hank in the Computer Architecture course when I was a late third year PhD student struggling with my machine learning (ML) research. Taking this course was not my intent of interest but to fulfill the departmental requirements. Three month later after I completed this course, Hank emailed me asking if I wanted to develop my course project into a paper level project. I replied yes and this is how I started research on computer architecture and systems. Eight months later, we submitted an ISCA paper and it got accepted, which is my first publication with Hank. When my fourth year ended, Hank became my advisor officially. I appreciate that he saw something in me and encouraged me to apply my ML expertise to solve systems problem. Later, he noticed my work in causal inference, a unique domain that most ML researchers had not paid attention to. Again, he encouraged and helped me to develop a holistic and coherent thesis statement, which led to the next two projects for the rest of my PhD. During the process of working with Hank, he helped me understand how to develop my own research problems, how to write high quality papers, and how to give academic presentations. His efforts have helped me become an independent researcher with my own tastes and unique ideas. I cannot imagine a better advisor than Hank.

Hank is always open-minded to new research topics. He introduced me to psychologists at UChicago so that I could apply my causal inference expertise to solve environmental neuroscience problems. He also introduced me to chemists from the University of Pittsburgh so that I could apply my ML expertise to solve protein-binding problems. These opportunities let me know more people and feel more confident in my chosen field.

Hank also advocates for diversity and inclusion. I was extremely touched when he flew to Houston *alone* to represent UChicago CS to support women in computing at Grace Hopper Conference in 2018. He did not work at all for three days even when no one visited the booth because he was afraid no one would come to the booth if he was seen working. Hank's research group is

the most diversified among the department, and he treats everyone equally in the group. He really has done and is doing something for his belief, not just talking.

Nearly the end of my PhD, a global pandemic COVID-19 occurred, which hurt the academic job market badly. CRA and CCC responded rapidly to this and launched the CIFellows 2020 program. I took this opportunity and Hank introduced me to his friend Michael (Mike) Carbin from MIT, who is also an amazing and kind person. Mike was extremely helpful in writing application materials and I was finally selected to be a CIFellow. Thank Hank and Mike both for helping me with my academic career in such difficult times.

The last and most important lesson that I learn from Hank is to become a kind person. Kindness is the most important quality as a human being, and thus Hank is the best human being I have ever met in academia so far. Finally, I am truly happy for him that he won a PECASE and earned a black belt in Tang Soo Do.

The rest acknowledgments are for other people in my PhD journey. I would be remiss without thanking Panos Toulis from Chicago Booth, who introduced me to causal inference. I reached out to Panos in Summer 2017 when I was very lost in my machine learning research, and he nicely responded and started meeting with me. After finishing our first project, he introduced me to his colleague Guillaume Basse from Stanford University, from whom I learned lots of statistics in experimental design. Panos also always supported my travel to difference conferences very generously. I would not have encountered causal inference and then developed my unique PhD research topics without Panos.

I would like to thank Bryon Aragam from Chicago Booth, who taught me lots of rigorous statistics in causal inference and graphical models. I reached out to Bryon in Fall 2019 after finishing my internship at Google because I noticed his remarkable work in learning causal graphs. Fun fact is that I actually first met Bryon in Panos's office coincidentally when Bryon was here for job interview. Bryon is a very well-organized and approachable researcher and I enjoyed working with him every step for our NeurIPS paper.

I would like to thank Risi Kondor, who was my first PhD advisor at UChicago. Nothing above would happen if Risi did not admit me as his student. I am extremely grateful that Risi introduced group theory and representation theory to me, which are highly advanced mathematical tools that even many ML researchers are not aware of. I also appreciate his relentless help when I worked on my first NeurIPS spotlight paper, which gave me confidence in pursing PhD degree.

I also met many amazing people at Chicago. I am fortunate to have you as my friends: Dixin Tang, Horace Pan, Zechao Shang, Renyu Zhang, Guangpu Li, Mingzhe Hao, Min Xu, Huaicheng Li, and Gushu Li. I know that I cannot name you all. I am also fortunate to have you in the same research group: Ahsan Pervaiz (thanks for helping me collect data), Muhammad Husni Santriaji, Yuliana Zamora, Tejas Kannan, Anne Farrel, Bernard Dickens, and Huazhe Zhang. I want to thank extraordinary Jean Salac for founding the Graduate Women in Computer Science (GWiCS) and inviting me to co-chair with her, which is quite an leadership experience and so much fun when hanging out with all women students. I want to thank multi-talented Kartik Singhal for founding the graduate Ministry and inviting me to apply for and become the Prime Minister to serve the whole department for two years. What an incredible leadership experience! I want to thank my lifelong friends, Hanyue Pu, Zhu Sun, and Tao Chen, who are always there for me outside academia. Special thanks to literate and witty Tianqi Tang for sharing fun and knowledgeable conversations with me, especially during this difficult pandemic time.

Finally, I would like to express my tremendous thanks and gratitudes to my greatest father. You are always there for me no matter when, where, and how. You have spent your entire life accompanying, supporting, and encouraging me. I am extremely fortunate to receive the best education in the world from you. I hope I will spend more time with you for the rest of my life, making you the proudest and happiest father.

ABSTRACT

Modern computer systems expose diverse configurable parameters whose complicated interactions have surprising effects on performance and energy. This puts a great burden on systems designers and researchers to manage such complexity. Machine learning (ML) creates an opportunity to alleviate this burden by modeling resources' complicated, non-linear interactions and deliver an optimal solution to scheduling and resource management problems. However, naively applying traditional ML methods, such as deep learning, creates several challenges including generalization [112] (ability to guarantee good prediction performance for future input), robustness [63] (ability to guarantee learning models robust to errors and adversaries), and interpretability [47] (ability to explain why ML models make such decisions).

This dissertation contains two projects that tackle the fundamental challenges described above in learning for systems by incorporating the underlying system structure, which is defined as the *geometry* of the system problems we solve. Figure 1 illustrates the structure for the two projects.

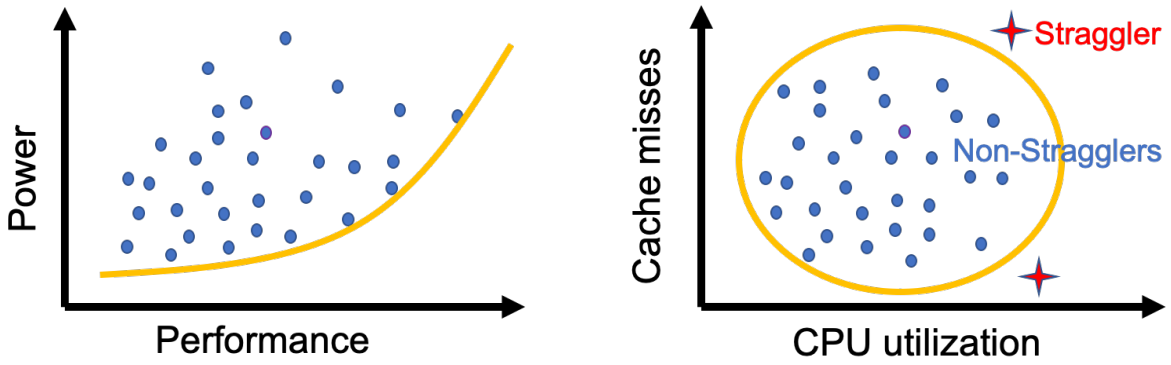


Figure 1: **Illustrations of system structure for Chapter 2 (left) and Chapter 3 (right).**

The first project describes learning for systems optimization in Chapter 2. We propose a novel *generative model* to address the data scarcity issue and a *multi-phase sampling* approach by exploiting system structure, i.e., the optimal frontier of performance and power tradeoff space. Our results are both positive and negative. The generative model addresses the data scarcity issue and improves accuracy by enhancing the generalizability and robustness of learning models, but negatively impacts energy. Multi-phase sampling reduces energy consumption by incorporating system

structure, but does not improve accuracy. 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—*e.g.*, geometry of optimal tradeoffs for the systems problem to be solved. *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.*

The second project describes Sherlock in Chapter 3, a causal straggler prediction framework for datacenters. Stragglers are rare events that exhibit extreme tail latencies, which lead to imbalance—structure—in the training data. To address the data imbalance issue, Sherlock augments correlation-based learning with *causal* analysis without prior knowledge. This eliminates the burden of carefully curating a training set while allowing Sherlock to generalize to never-before-seen workloads. To effectively mitigate stragglers, Sherlock applies permutation feature importance (PFI) to gain insights into the straggling behavior for further system intervention. Sherlock's combination of PS and PFI allows it to make accurate, interpretable predictions from imbalanced training data. *This work is evidence that causal analysis is effective in delivering more generalizable, robust, and interpretable systems.*

ML has influenced much recent systems research including providing systems support for ML and applying ML to solve systems problems. To further produce generalizable, robust, and interpretable results, it is crucial to understand the underlying system structure. This dissertation constitutes a new understanding of ML for systems that opens the way to a novel class of methodology and application by incorporating causal inference techniques.

CHAPTER 1

INTRODUCTION

1.1 Background

Modern computer systems expose diverse configurable parameters whose complicated interactions have surprising effects on performance and energy. This puts a great burden on systems designers and researchers to manage such complexity. Machine learning (ML) creates an opportunity to alleviate this burden by modeling resources' complicated, non-linear interactions and deliver an optimal solution to scheduling and resource management problems. 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) [17, 135, 49, 119, 29, 30, 81, 92, 94, 163, 105]. An alternative approach learns models of high-level behavior from observing similar applications [40, 41, 107, 104], hardware configurations [156, 143], or both [113, 43].

However, naively applying traditional ML methods without considering system structure creates several challenges including generalization [112] (ability to guarantee good prediction performance for future input), robustness [63] (ability to guarantee learning models robust to errors and adversaries), and interpretability [47] (ability to explain why ML models make such decisions). In particular, several challenges must be addressed to continue improving learning for computer system resource management, including:

- **Training set sensitivity.** 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. This

requires the training set to be *balanced*; i.e., reflect both all possible outcomes that will need to be predicted and all possible causes of those outcomes with roughly equal representation. However, curating an appropriately balanced training set is hard when the task is predicting rare events like stragglers in datacenters.

- **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.
- **Poor interpretability.** Linear models are interpretable and have been used to understand learned system results like straggling behavior in distributed systems [149, 150]. However, most learning models applied to systems are black-box and thus are difficult to interpret despite high prediction accuracy [160]. Lack of interpretability is a problem, because without an indication of the reasons for certain system behavior it is difficult to intervene effectively.

1.2 Thesis

This dissertation tackles the abovementioned fundamental challenges of applying ML techniques to systems. We focus on two specific goals: (1) learning for systems optimization with scarce data and system structure (Chapter 2), and (2) learning for straggler prediction with imbalanced data (Chapter 3). In particular, Chapter 2 addresses the challenges of training set sensitivity *in a single machine* and asymmetric benefits, and Chapter 3 addresses the challenges of training set

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 are common to both cases.

sensitivity *in datacenters* and poor interpretability. Both chapters characterize the system structure in different formats—single machine and datacenters—and present new methodologies that are robust, generalizable, real-time, and resource-efficient to improve computer systems outcomes.

1.3 Summary of Results

Learning for Systems Optimization with Scarce Data and System Structure (Chapter 2): One difficulty of procuring training data necessary to apply ML to systems problems is data scarcity: either the training set size is not big enough, or the training data fails to exhibit a wide range of behavior. To overcome this difficulty, we propose a novel *generative model* by generating data that improves learning accuracy (Section 2.4). The key insight is determining how to generate data that is sufficiently different from the training set, but still realistic enough to predict unseen behavior. By augmenting the original training data with the newly generated data, the learning models better generalize to the unseen applications so as to produce robust results.

Despite the improved prediction accuracy after data augmentation, we find that it does not improve the systems outcome—*e.g.,* energy consumption. The reason is that not all resource configurations are useful in learning for systems in the sense of representing optimal tradeoffs (*e.g.*, between performance and power). Ideally, a computer system would only use configurations on the optimal frontier and ignore all non-optimal configurations, an example of how the underlying problem structure affects systems outcomes. However, it is not known a priori which configurations are optimal to begin. To address this, we propose a *multi-phase sampling* approach by exploiting system structure (Section 2.5). The idea is to bias the learner towards optimal configurations to improve energy savings. By incorporating system structure, the multi-phase sampling approach significantly improves energy savings, yet reduces learning accuracy.

Our results are both positive and negative (Section 2.7). The generative model addresses the data scarcity issue and improves accuracy by enhancing the generalizability and robustness of learning models, but negatively impacts energy. Multi-phase sampling reduces energy consump-

tion by incorporating system structure, but does not improve accuracy. 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—*e.g.*, geometry of optimal tradeoffs for the systems problem to be solved. *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.*

Learning for Straggler Prediction with Imbalanced Data (Chapter 3): Structure not only exists in the data generated from a single machine, but also those from datacenters. A notable example is stragglers—computations that exhibit extreme tail latencies, which pose a major difficulty of delivering predictable performance in datacenters [6]. Accurately predicting stragglers would enable efficient, proactive intervention. While ML methods have been applied to predict computer system performance, they have limitations that are problematic in the context of straggler mitigation.

As another form of data scarcity, stragglers are rare events in the training data. However, achieving accurate predictions requires the training set to be *balanced*; i.e., reflect both all possible outcomes that will need to be predicted and all possible causes of those outcomes with roughly equal representation. Therefore, curating an appropriately balanced training set is hard when the task is predicting rare events like stragglers. The state-of-the-art to address imbalance is *oversampling* by counting the rare stragglers more than once. Unfortunately, this technique will fail if a new workload is unlike any workload in the training set. Also, oversampling makes the trained model likely to overfit to the oversampled data.

To address data imbalance issue, we propose Sherlock (Section 3.3), a straggler prediction framework that augments correlation-based learning with *causal* analysis [79]. The key insight is that rather than try to adjust the training set (e.g., by oversampling stragglers), Sherlock adjusts the predictions to correct any bias due to imbalanced data. This approach eliminates the burden of carefully curating a training set while allowing Sherlock to generalize to never-before-seen workloads. Specifically, Sherlock starts with a correlation-based learner and then weights this

learner's predictions by their *propensity score* (PS), a statistical method for dealing with imbalanced data from causal inference [128]. Sherlock uses PS to quantify how different a particular sub-computation's features are from those that are known *not* to straggle, which corrects the bias in predictions aroused from data scarcity at training. This approach is a novel application of causal inference for predictions, not just understanding causes once outcomes are known.

To effectively mitigate stragglers, it is also crucial to understand the straggling causes for further system intervenation. To do so, we propose permutation feature importance (PFI) to gain insights into the reasons for straggling behavior (Section 3.3.3). Different from prior work that used linear models for interpretation at the cost of prediction accuracy [149], PFI is designed to interpret highly accurate black-box models on live data, immediately after a straggler is predicted. Sherlock's combination of PS and PFI allows it to make accurate, interpretable predictions from imbalanced training data.

We evaluate Sherlock on datacenter traces from Google and Alibaba and find that, compared to prior work that is heavily reliant on the training set, Sherlock more accurate, earlier, and more interpretable predictions (Section 3.5). Sherlock achieves these improved predictions by exploiting the underlying structure—training data imbalance—and then applying key technique from causal analysis. *This work is evidence that causal analysis is effective in delivering more generalizable, robust, and interpretable systems.*

CHAPTER 2

GENERATIVE AND MULTI-PHASE LEARNING FOR COMPUTER SYSTEMS OPTIMIZATION

2.1 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 [56, 86, 22, 80, 107, 104].

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 [40, 41, 104, 107, 135, 49, 57, 113, 163, 81, 17, 119, 29, 78]. 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.
- *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

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 work are common to both cases.

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 [40, 41, 107].

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.

2.2 Related Work and Motivation

Several studies provide evidence that heuristic-based resource management can break down as the underlying computing systems become more complicated [80, 86, 22, 107, 104, 162]. 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 [12, 52, 77]. 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 [86, 22]. 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 differ-

ent hardware devices, as demonstrated in both academic studies [80] and in commercial examples [56].

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

2.2.1 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 [40, 153, 107, 91, 104, 135, 163, 113, 156, 95, 136, 103, 157, 32, 31, 122], configuration [96, 142, 154, 28, 164, 49, 92, 94, 119, 57, 8, 147, 43, 139, 140], scheduling [40, 41, 143, 78, 146, 138]. Several of these approaches are for *offline* system design—*e.g.*, determining the optimal number and type of cores in a heterogeneous processor [139, 140]—but this work 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) [17, 135, 49, 119, 29, 30, 81, 92, 94, 163, 105]. 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 [135].

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 [49]. 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 [119].

An alternative approach learns models of high-level behavior from observing similar applications [40, 41, 107, 104], hardware configurations [156, 143], or both [113, 43]. For example, Paragon [40] and its follow-up, Quasar [41], use the Netflix algorithm [14] to determine efficient schedules for a new application from models of similar applications [40]. 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 [143]. Finally, Carat models combinations of applications and mobile devices to determine energy savings opportunities for other combinations [113].

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 [84, 64]. Some generative techniques have been proposed for learning in systems; *e.g.*, generating code to produce better learning systems for optimized compilation [35]. The generative technique proposed in this work 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 work 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

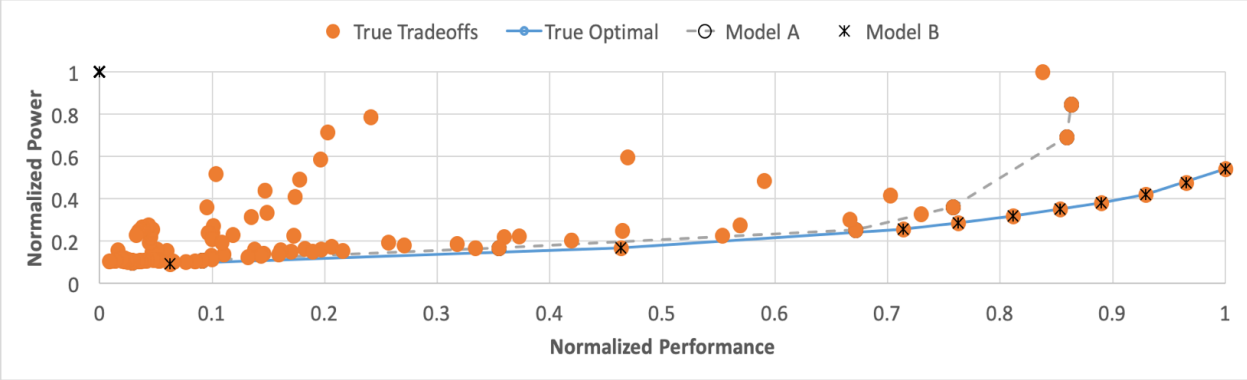


Figure 2.1: **Learning performance/power tradeoffs for SRAD on an ARM big.LITTLE system. The dots show the true tradeoffs. The solid line shows the true optimal frontier. Model A is a learner that is accurate except for the optimal frontier. Model B captures the optimal frontier and gets all other tradeoffs wrong.**

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*.

2.2.2 *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 ([80]) 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 work, 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 [27]) 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 2.1 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 work: *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 2.6.5) and find it is 99% accurate. We measure energy by feeding this model to a scheduler (from [80]) 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

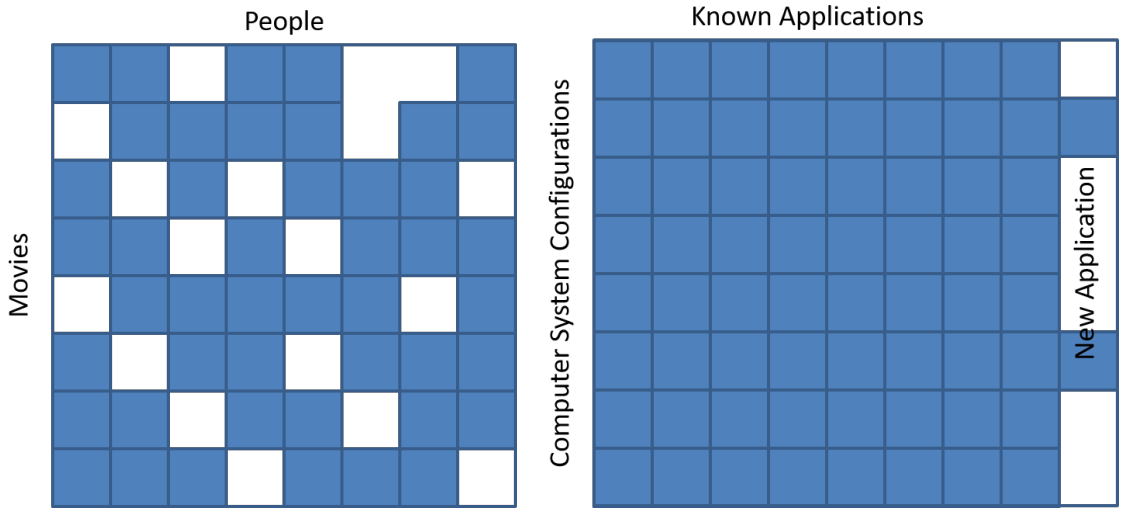
performance and maximum power. Model B's goodness-of-fit is essentially 0—not surprising, as most points are inaccurately predicted. All the optimal points are predicted with no error, however, so the energy is the same as optimal.

This example illustrates the most important intuition behind the remainder of the work. 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.

2.3 Learning by Example

This work builds off prior systems work that *learns by example* [40, 41, 107, 143, 113]. The Paragon project applies this idea to resource management in cloud computing [40]. Paragon takes inspiration from *recommender systems*, specifically the solution to the Netflix challenge [14]. 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 2.2a) 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.

Paragon shows that this structure can be applied to computer systems where we have a configurable computer system and many benchmark applications [40]. 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 2.2b now each row is a system configuration (*e.g.*,



(a) Movie Recommendations (b) Computer Systems

Figure 2.2: **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.**

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 [107] or mobile system [104], assignment of a request to a node in a heterogeneous data center [40], or a combination of resource assignment and application co-scheduling in the data center [41]. 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 movie over another. In computer systems, however, typically only a handful of configurations are actually useful, and Figure 2.1 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 2.4 we present a technique to generate representative data and deal with the challenge of scarce data. Section 2.5 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.

2.4 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 2.3. 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

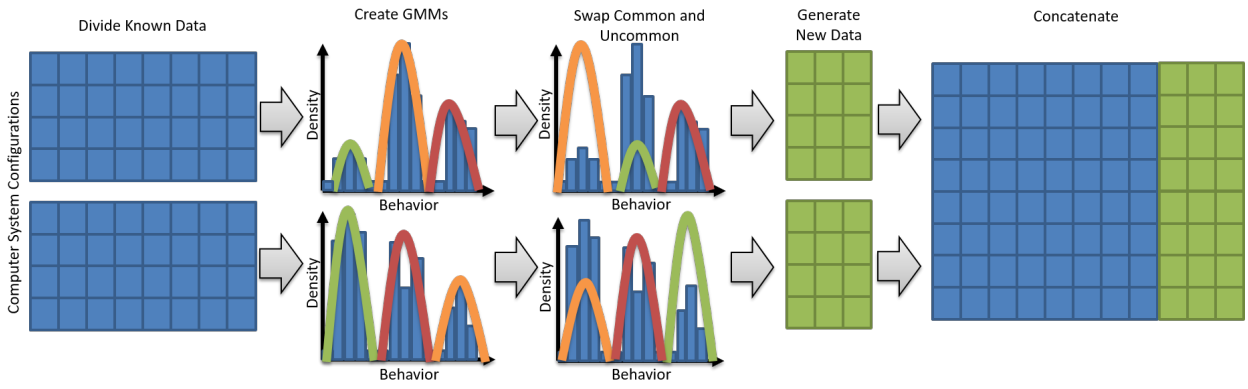


Figure 2.3: **Workflow of proposed generative model.**

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 2.2b. 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.

2.4.1 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 $\mathbf{x}_i \in \mathbb{R}^D, i = 1, \ldots, N$, 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:

$$p(\mathbf{x}_i) = \sum_{k=1}^{K} w_k g(\mathbf{x}_i|\mu_k, \Sigma_k) \tag{2.1}$$

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

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

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 [69].

2.4.2 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 [9] or BSCAN [51]. 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.

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

2.4.3 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.

2.5 Multi-phase Sampling

Figure 2.4 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:

$$\text{efficiency} = \frac{\text{estimated performance}}{\text{estimated power}}. \tag{2.2}$$

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 config-

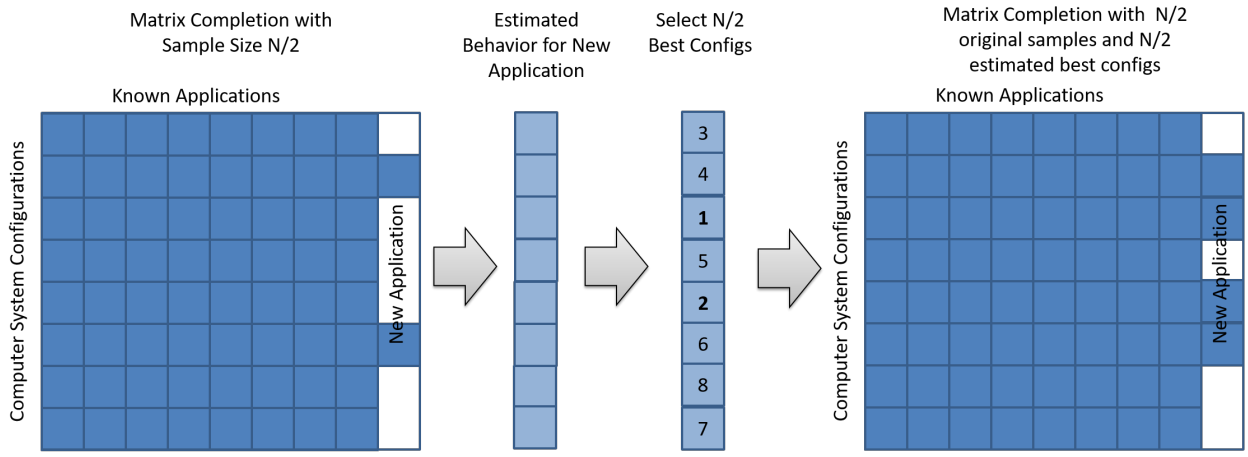


Figure 2.4: **Workflow of proposed multi-phase sampling scheme.**

uration [86]. 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 2.1). 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.

2.6 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.

Algorithm 1 Multi-phase sampling approach.

Require: Known and unknown applications
Require: Sampling budget N
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.

Output: Estimation of performance and power.

2.6.1 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.

2.6.2 Applications

We test a variety of applications on both the mobile and server systems using benchmarks drawn from Parsec [16], Rodinia [27], ParMiBench [82], and MineBench [110]. Parsec and Rodinia contain a number of general purpose workloads. ParMiBench contains multithreaded versions of the embedded MiBench benchmarks [70], 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`).

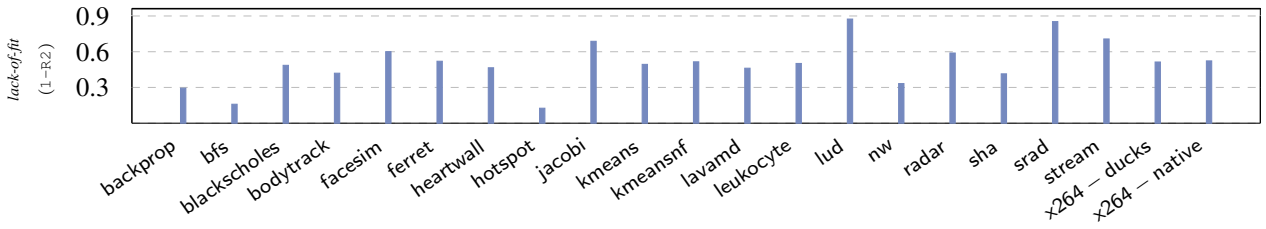


Figure 2.5: ***Lack-of-fit* for performance vs clock-speed on mobile.**

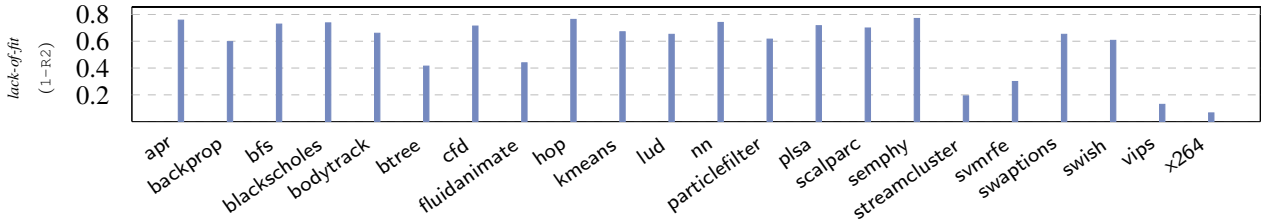


Figure 2.6: ***Lack-of-fit* for performance vs clock-speed on server.**

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.

2.6.3 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 2.5 and 2.6 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.

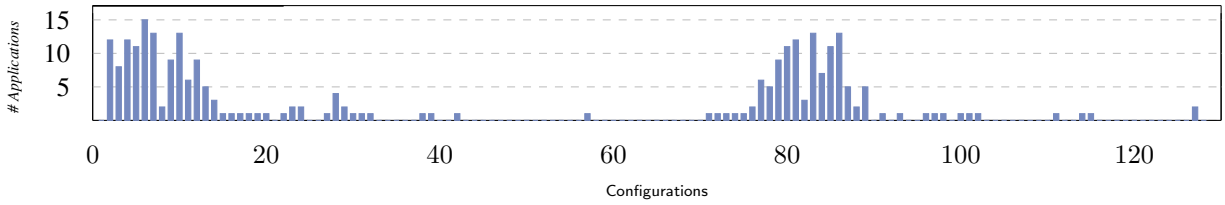


Figure 2.7: **Distribution of optimal configurations for mobile.**

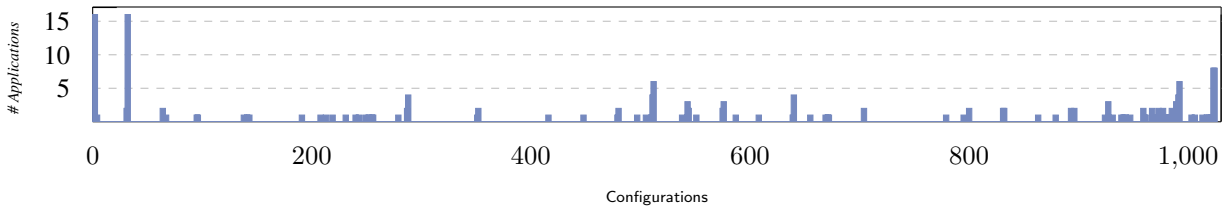


Figure 2.8: **Distribution of optimal configurations for server.**

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 2.7 and 2.8 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 2.9 shows this data for the mobile while Figure 2.10 shows the data for the server. The largest number of configurations for any application on mobile 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 appli-

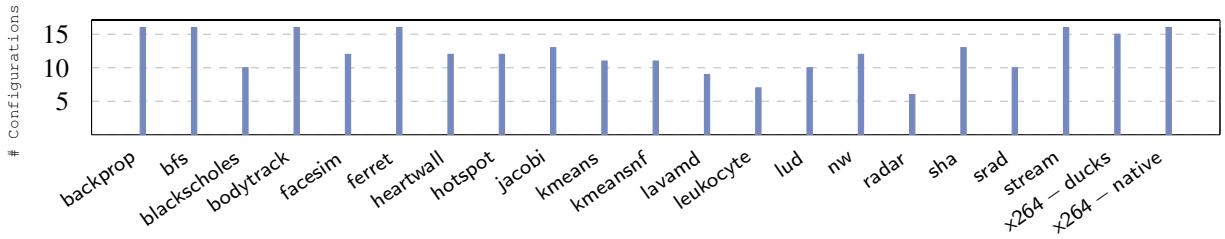


Figure 2.9: **Optimal configuration count for mobile applications.**

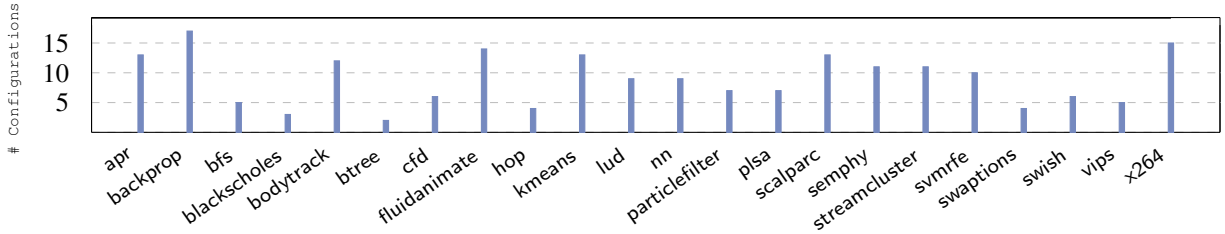


Figure 2.10: **Optimal configuration count for server applications.**

cation's optimal frontier.

2.6.4 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 [85].
2. *MCMF*: a matrix completion algorithm solved by factorizing the matrix into bi-linear form [88].
3. *Nuclear*: an exact matrix completion algorithm by minimizing the matrix's nuclear norm [20, 19]. This technique has been demonstrated to provide good systems outcomes in scheduling for data centers [40, 41].
4. *WNNM*: an exact matrix completion algorithm by minimizing the matrix's weighted nuclear norm [65].
5. *HBM*: a hierarchical Bayesian model for recovering optimal performance/power tradeoffs [107].

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.

2.6.5 Evaluation Metrics

For each application, we evaluate prediction accuracy by using adjusted R^2 [48], which measures the *goodness-of-fit* between the ground-truth $\mathbf{y}$ and estimated value $\hat{\mathbf{y}}$:

$$R^2 = \max\left(0, 1 - \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}{\|\mathbf{y} - \bar{\mathbf{y}}\|^2}\right), \tag{2.3}$$

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:

$$\texttt{Normalized energy} = 100\% * (\frac{\mathrm{e}_{\mathrm{measured}}}{\mathrm{e}_{\mathrm{optimal}}} - 1), \tag{2.4}$$

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

2.6.6 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 en-

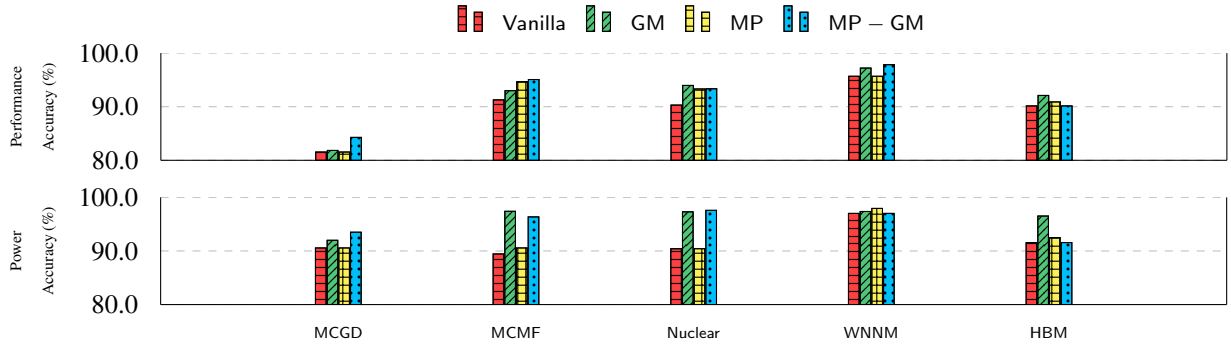


Figure 2.11: **Performance and power accuracy on mobile (higher is better).**

ergy 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 2.2b. We then sample i in several configurations to form the partial, last column from Figure 2.2b.

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 [80]. 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.

2.7 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.

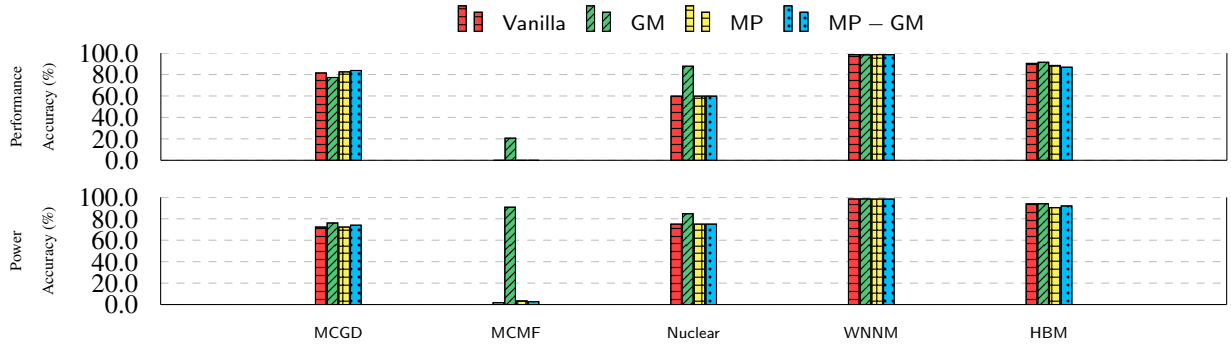


Figure 2.12: **Performance and power accuracy on server (higher is better).**

2.7.1 Summary Results

Accuracy

We show the estimation accuracy for performance and power on both mobile and server in Figure 2.11 and Figure 2.12. 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 [107, 104], 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 2.1 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 2.1: **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

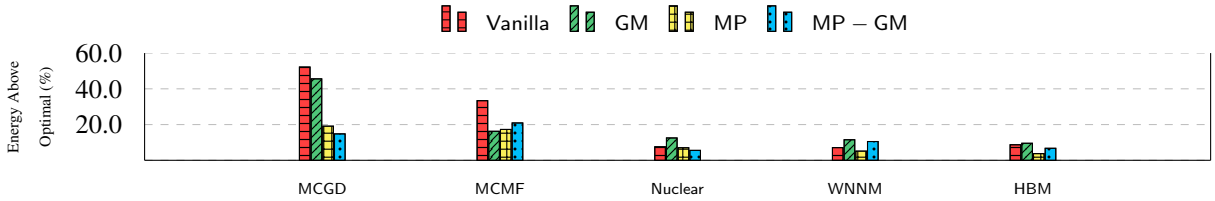


Figure 2.13: **Energy compared to optimal on mobile (lower is better).**

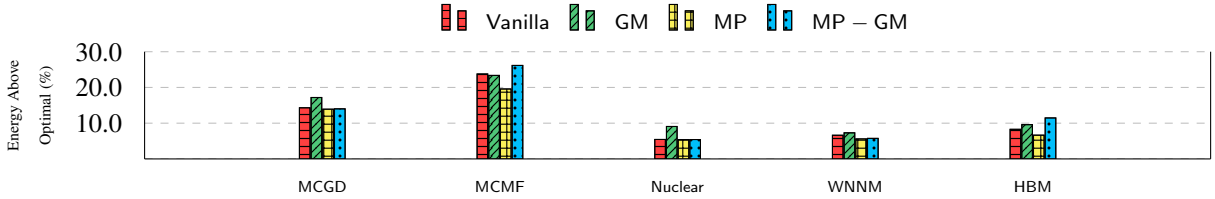


Figure 2.14: **Energy compared to optimal on server (lower is better).**

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 [40, 41] and HBM [107]) to the similar levels of accuracy. *We conclude that GM can have a dramatic effect on the accuracy of performance and power predictions*.

Energy

Figures 2.13 and 2.14 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 2.2, 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

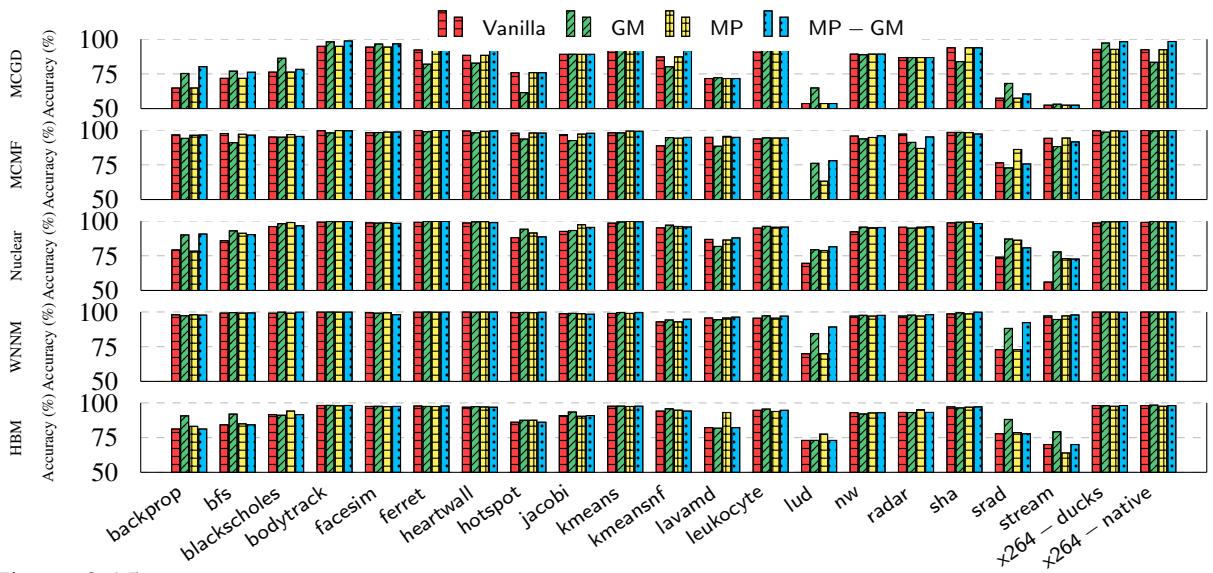


Figure 2.15: **Comparison of performance accuracy per application for mobile system (higher is better).**

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 2.2: **Average energy improvement. (Higher is better).**

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

2.7.2 Detailed Accuracy Results

We find that power is, generally, much easier to predict than performance. Therefore we present only performance accuracy detailed results to save space. Figures 2.15 and 2.16 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

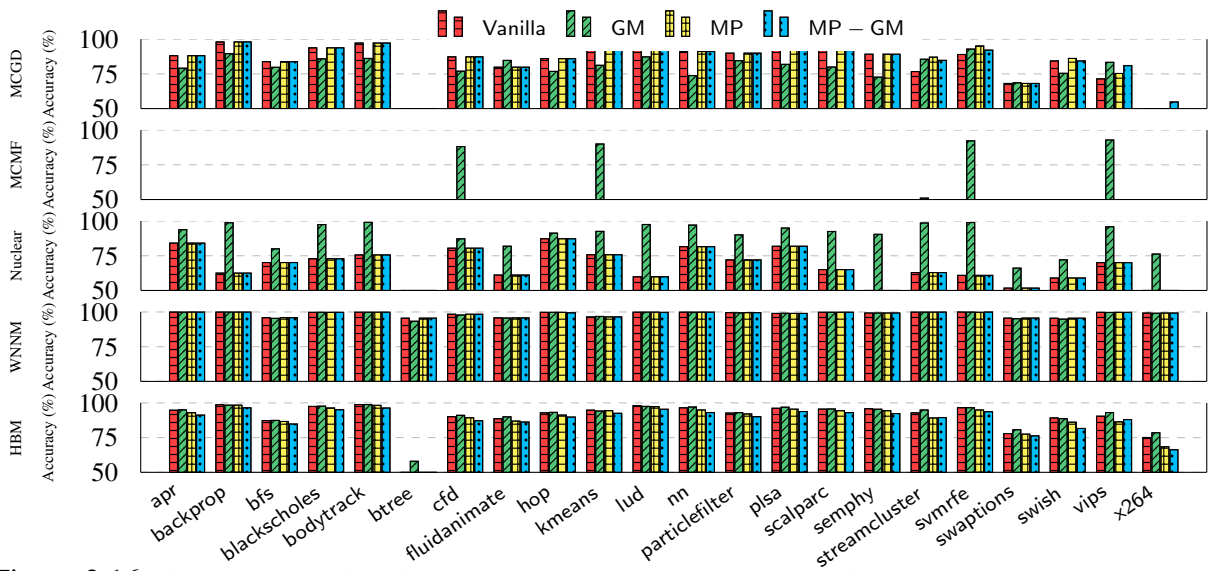


Figure 2.16: **Comparison of performance accuracy per application for server system (higher is better).**

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 2.9 and 2.10 where the mobile data is clustered, but the server data is evenly distributed.

2.7.3 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 2.17 and 2.18 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%.

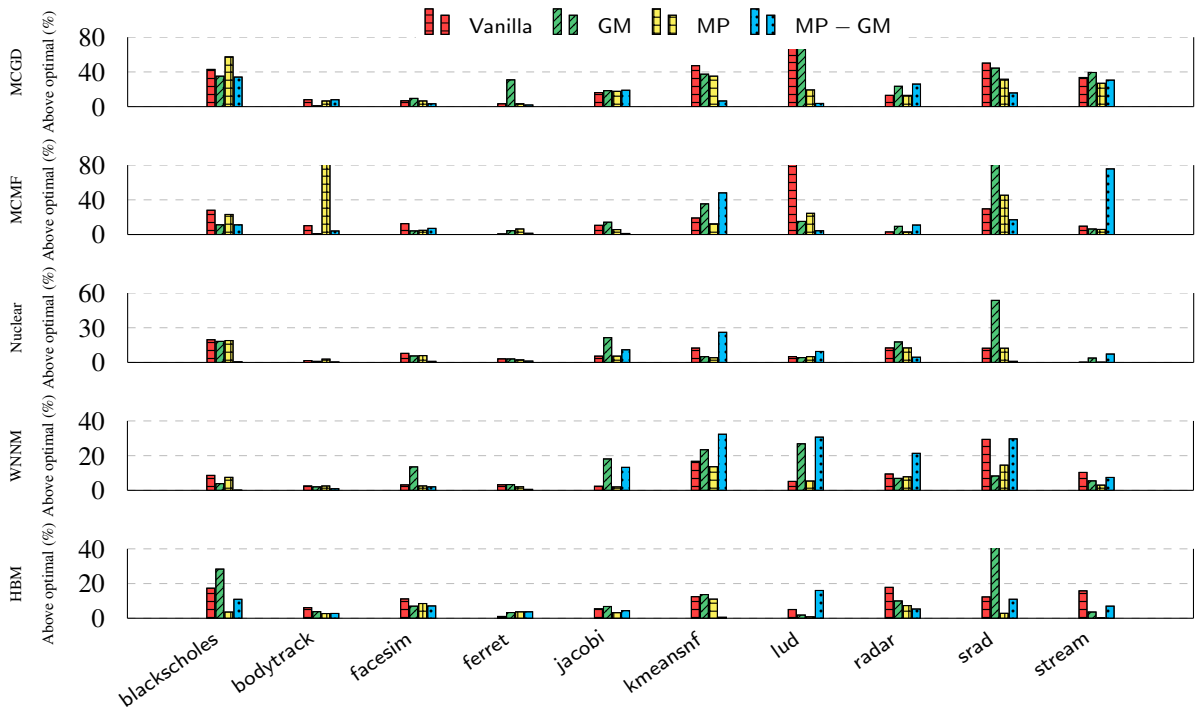


Figure 2.17: **Energy savings per application for mobile system (lower is better).**

2.7.4 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 [40, 41] and HBM [107, 104] have both been used in prior systems work, while WNNM only recently appeared in the ML literature [65]. 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 2.3. The energy savings for GM are worse when removing MCGD and MCMF as shown in Table 2.4. 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

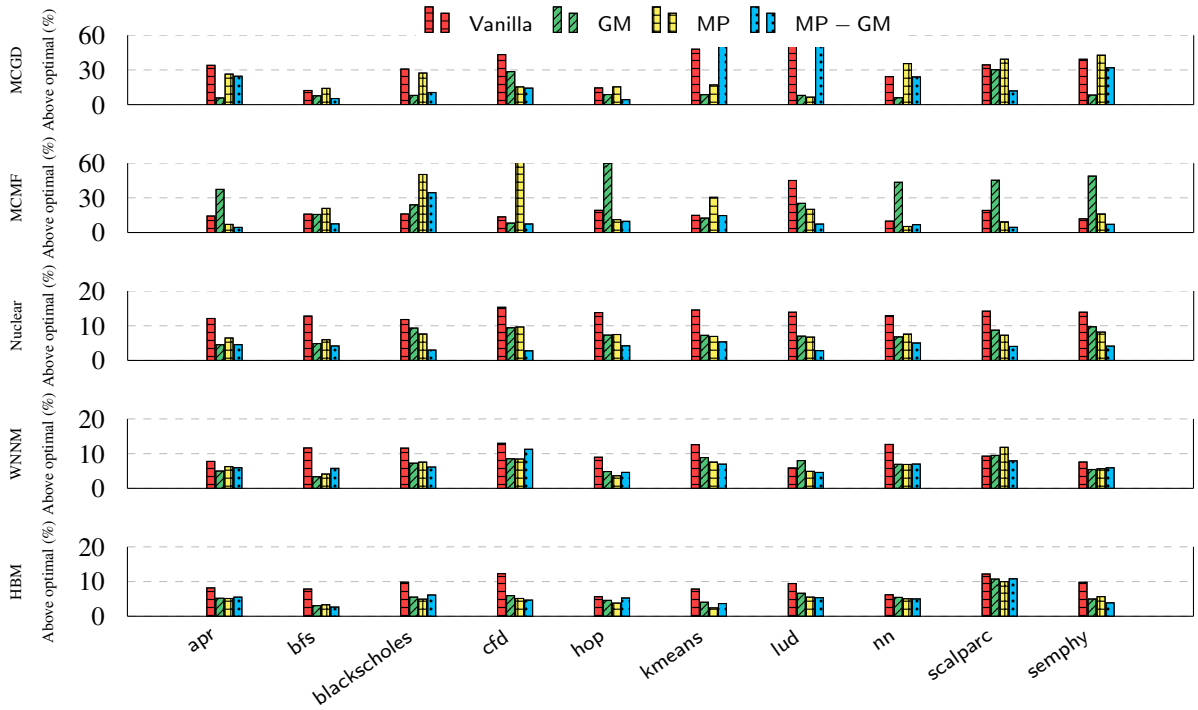


Figure 2.18: **Energy savings per application for server system (lower is better).**

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 work—but accounting for the problem structure (in this case, by modifying the sampling procedure) does improve the system outcome.

Table 2.3: **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 2.4: **Energy improvement for the best learners.**

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

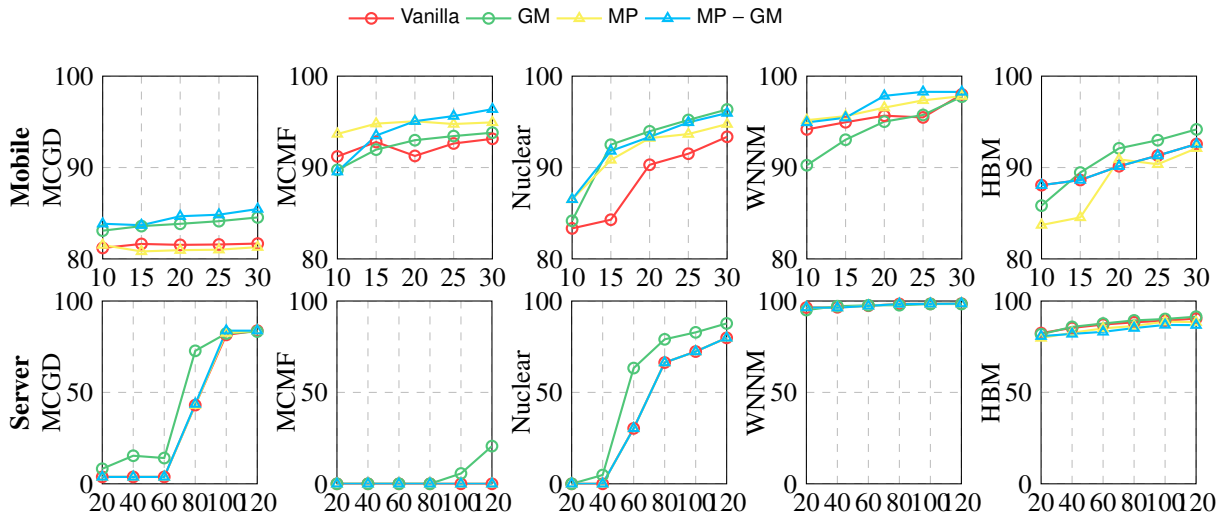


Figure 2.19: **Sensitivity analysis of performance accuracy for mobile and server systems. The x-axis is the sample size. The y-axis is the performance accuracy.**

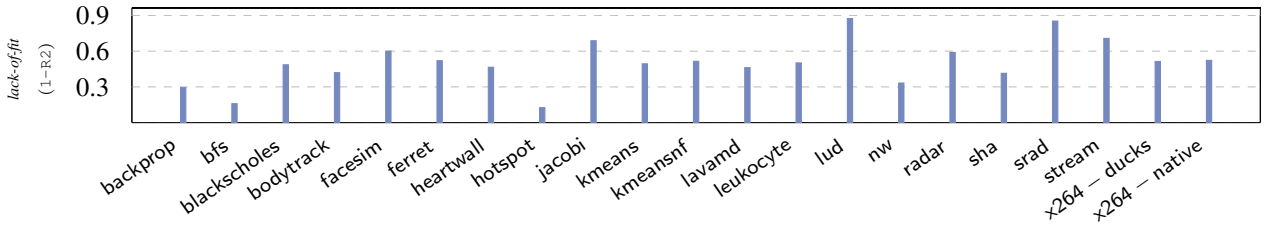


Figure 2.20: ***Lack-of-fit* for performance vs clock-speed on mobile.**

2.7.5 Sensitivity to Sample Size

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 2.19, 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.

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 2.5: **Average energy improvement with reduced samples.**

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

Table 2.5 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).

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.

2.7.6 Overhead

We collect the overhead for all combinations of systems, learners, and proposed augmentations. These overheads are listed in Table 2.6 and Table 2.7 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 2.6: **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 2.7: **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

2.7.7 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.

2.8 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 2.2 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 multi-phase 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.

CHAPTER 3

CAUSAL AND INTERPRETABLE LEARNING FOR DATACENTER LATENCY PREDICTION

3.1 Introduction

Reducing completion time for latency-critical applications is a fundamental problem in datacenter-scale computing [13, 1, 99, 123, 72, 74, 75, 97, 151, 24, 111, 11]. A major impediment is the presence of *stragglers*, exceptionally slow sub-computations within some larger computation. Several studies show that, while rare, stragglers degrade overall performance by as much as 30 to 50% [6, 124, 161].

Existing straggler mitigation techniques fall into two broad classes: performance-oblivious and performance-aware. Performance-oblivious approaches reduce straggling penalties without predicting which tasks will straggle. These include methods like replicating computations [155] and speculative execution [39, 125]. Performance-aware techniques monitor executing computations and predict straggling *before* it reveals itself with long run times [73, 149]. Performance-aware methods promise greater resource utilization: if straggling is accurately predicted, extra resources will only be used for the rare stragglers. Thus, performance-aware methods critically depend on prediction accuracy.

Predicting performance is useful in many scheduling problems and there is a large body of work using machine learning (ML) to predict computer system performance (see recent surveys for a broad overview [118, 159]). The vast majority of these are based on finding *correlations* between some set of input features (like CPU utilization) and computation latency. Unfortunately, these learning techniques have limitations that are particularly problematic in the context of straggler mitigation:

- **Training set sensitivity** Achieving accurate predictions requires the training set to be *balanced*; i.e., reflect both all possible outcomes that will need to be predicted and all possible

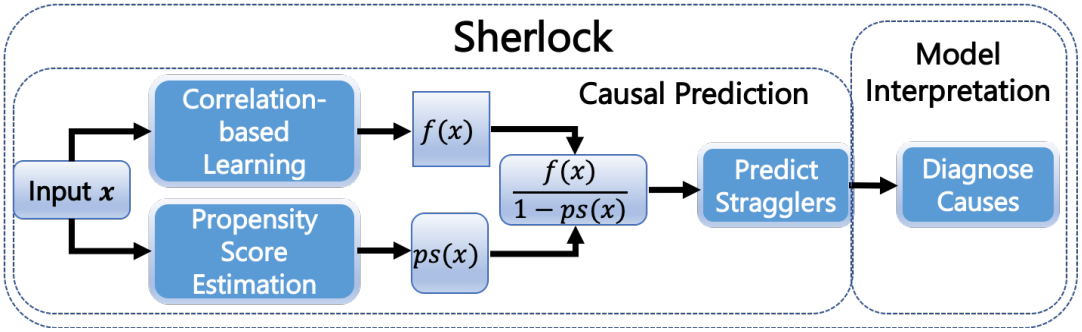


Figure 3.1: **Sherlock, a causal straggler prediction framework.**

causes of those outcomes with roughly equal representation. However, curating an appropriately balanced training set is hard when the task is predicting rare events like stragglers. The state-of-the-art to address imbalance is *oversampling* by counting the rare stragglers more than once [26]. Unfortunately, this technique will fail if a new workload is unlike any workload in the training set. Additionally, oversampling makes it likely that the trained model will overfit to the oversampled data [89].

- **Poor interpretability** Linear models are interpretable and have been used to understand straggling behavior [149, 150]. However, most learning models applied to systems are black-box and thus are difficult to interpret despite high prediction accuracy [160]. Lack of interpretability is a problem, because without an indication of why the task is straggling it is difficult to intervene effectively.

To address these limitations, we present Sherlock (Figure 3.1), a straggler prediction framework that augments correlation-based learning with *causal* analysis [79]. The key insight is that rather than try to adjust the training set (e.g., by oversampling stragglers), Sherlock adjusts the predictions to correct any bias due to imbalanced data. This approach eliminates the burden of carefully curating a training set while allowing Sherlock to generalize to never-before-seen workloads.

Specifically, Sherlock starts with a correlation-based learner. Then Sherlock weights this learner's predictions by their *propensity score* (PS), a statistical method for dealing with imbalanced data [128]. Sherlock uses PS to quantify how different a particular sub-computation's features are from those that are known *not* to straggle. This weighting scheme accounts for training

sensitivity by preserving latency predictions for sub-computations that are similar to known non-stragglers and increasing predicted latency for those that are very different. When a straggler is predicted, Sherlock applies permutation feature importance (PFI) [53] to return a list of features ranked by how much each contributes to latency. Sherlock's combination of PS and PFI allows it to make accurate, interpretable predictions from imbalanced training data.

We implement Sherlock and evaluate it on two production traces from Google [124] and Alibaba [4]. For all evaluations, Sherlock works on *live* data and makes predictions about which sub-computations will straggle without first seeing any stragglers during training. This setup differentiates Sherlock from all prior work that requires appropriate representation of stragglers in training data [149, 150]. Compared to prior work:

- Sherlock's causal model improves the true positive rate by 59% and reduces the false positive rate by 7%.
- Sherlock identifies stragglers 1.46× earlier. In fact, unlike prior work, *Sherlock produces accurate predictions before it has any positive examples of straggling.*
- Sherlock provides reliable interpretations of straggling that are consistent with prior human expert analysis [161, 68].

In summary, this work makes the following contributions:

- Proposing a novel causal framework for straggler prediction. While prior work applies causal analysis to understand straggling after the fact, we believe Sherlock is the first to demonstrate its benefits for predicting performance.
- Proposing permutation feature importance to interpret and gain insights into the straggling behavior on live data.
- Releasing Sherlock and its testing infrastructure.[1]

1. Link to repository removed for double blind review.

3.2 Related Work and Motivation

We discuss prior work mitigating stragglers and predicting system performance. We then illustrate how these approaches struggle if their training sets are not carefully constructed.

3.2.1 Mitigating Stragglers

There is a large body of prior work on understanding and mitigating stragglers [38, 3]. This is effectively a scheduling problem: allocating resources to computations to reduce the longest running tasks' latency [129]. Mitigation techniques can thus be categorized by whether the scheduler is performance-oblivious or performance-aware.

Performance-oblivious schedulers do not predict which computations will straggle but simply schedule such that stragglers will naturally be reduced. For example, speculative execution launches redundant computations and uses the result of the first to complete [39, 155, 5]. Another example removes slow machines as schedulable resources without attempting to determine which computations might be suited for them [37, 6]. Performance-oblivious approaches are effective; e.g., if every computation is replicated, the worst case latency is very likely to be reduced. The downside is that they waste resources on redundant computation.

Performance-aware schedulers predict stragglers and only allocate additional resources to them. For example, MittOS allows IO requests to be submitted with a target latency [73] and the scheduler predicts if the latency can be satisfied before serving it or sending it to a less-loaded replica. Similar performance-aware schedulers predict stragglers before allocating resources [7, 125, 149, 150]. These approaches promise better resource utilization: with perfect predictions, extra resources only go to slow computations. The downside is that producing accurate predictions is challenging.

3.2.2 *ML for Performance Prediction*

ML has influenced much recent research including systems support for ML; e.g., [2, 83, 134, 152, 100, 98, 117, 132, 101, 54, 152, 58, 59, 71, 131, 126, 42]. We, however, are interested in ML for systems, especially scheduling and resource management. Many examples exist, including learning to predict performance from microarchitectural features [17, 49, 81, 87, 93, 106, 120, 135, 15, 60, 109], application features [23, 67, 40, 41, 107, 104, 45, 163, 78], and combinations of the two [113, 90, 158].

Two issues, however, make it difficult to apply these ML techniques to straggler prediction: (1) training set sensitivity and (2) model interpretability. First, the accuracy of the learning models cited above is highly sensitive to the data set used to train the model. Ideally, the training set data should come from controlled, randomized trials [61] and have roughly equal representation of both stragglers and non-stragglers [145]. Furthermore, all possible causes of straggling should be represented. Meeting these requirements is difficult because stragglers exhibit tail behavior and are thus rare by definition [38, 3]. Furthermore, training set curation is a challenging process to debug as it is often unclear when poor prediction is a result of a poor learning algorithm, a poor training set, or both.

The second challenge is that most learners used in the above projects are difficult to interpret as they rely on finding nonlinear relationships between input features and the predicted outcomes. This means that even if stragglers are correctly predicted, the underlying causes are not clear (the learned model is essentially a black-box [66]) and so it is not obvious how to intervene. For example, a scheduler should intervene differently if straggling is due to the underlying hardware rather than some property of the workload [46].

Wrangler [149, 150] is a performance-aware scheduler that tries to overcome the problems of training set sensitivity and interpretability. To ensure proper representation of stragglers in the training set, Wrangler *oversamples* them—counting each more than once. While oversampling is commonly used when dealing with imbalanced training sets [50], it tends to result in overfitting

the training data, which hurts generalization to new tasks that were not seen at training time [26]. To address interpretability, Wrangler uses only linear models so that the linear weights associated with any feature can be interpreted as that feature's importance. Linear models, however, usually come with low predictive power and cannot capture interactions between features [76]; e.g., a task that is memory-limited in some circumstances and compute-limited in others [40, 41, 104].

3.2.3 Learning From Imbalanced Training Data

Sherlock addresses the same challenges as Wrangler—training set sensitivity and black-box model interpretation—but it takes a fundamentally different approach. Rather than carefully curate the training set through oversampling or any other method, *Sherlock is designed from the beginning to produce accurate predictions from imperfect training data.* This approach makes Sherlock robust to potential errors in training set construction while enabling it to generalize to workloads for which it has no prior knowledge.

To do so, we start from existing techniques and then reweight their predictions to account for training set imbalance. This design assumes that initial predictions will be biased due to imbalanced training data. We construct a weighting function that organically corrects this bias using features of the input data. Specifically, we use *propensity scores* [128] from causal analysis, which represent the probability that a subject is in a particular group given its features.

Causal analysis has been proposed to understand computing systems behavior by carefully randomizing experiments [36, 141] and to find causes of performance anomalies after execution [161]. Causal analysis has also been used to understand ML models and guard them against pollution with malicious data [21]. However, we are not aware of any other work that uses causal analysis to predict future stragglers on live computations. In fact, we believe our use of propensity scores for prediction is novel in any field, not just computing.

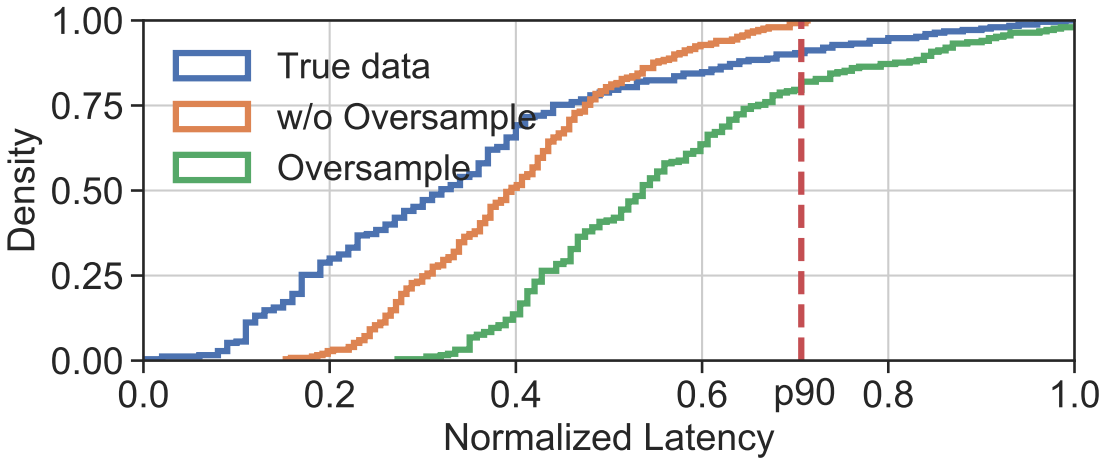


Figure 3.2: **CDF of normalized latency on a real computation: (a) true data; (b) predictions from learning model with representative stragglers (5% of training data); (c) predictions from learning model with oversampled stragglers.**

3.2.4 Motivational Example of Training Bias

We demonstrate how existing learners' straggler predictions are influenced by training set sensitivity. We study a computation from a production trace and construct two models: (1) trained with a representative number of stragglers and (2) trained by oversampling stragglers to balance the straggling and non-straggling data (based on prior work [149, 150]). All models predict latency as a function of features (like CPU utilization). However, they either (1) falsely predict all stragglers to be non-stragglers (due to too few stragglers at training) or (2) have high false positive rate, predicting many non-stragglers to straggle (due to overfitting to oversampled stragglers).

We consider the publicly available Google trace data [124], which has millions of jobs, each with multiple tasks. We conduct analysis on a specific job with ID 6343946350 and use its related tasks to construct a data set whose features are task metrics regarding machine capacity, scheduling, resource request and usage, and microarchitecture (Section 3.4 details the full experimental setup). We define tasks with completion time beyond 90th percentile as stragglers. Our goal is to use the features to predict whether a task will straggle. We use gradient boosting [55] as our learner.

Figure 3.2 shows the results. The blue curve shows the true CDF for tasks in this job. A red vertical line shows the p90 cutoff. The orange curve shows the predicted CDF from the model with

trained with representative stragglers, which predicts all tasks not to straggle (the CDF hits 1 to the left of the true p90 cutoff). The green curve shows the predictions from oversampling stragglers, which predicts a large number (38.7%) of non-stragglers to straggle (the CDF is shifted to the right of the p90 cutoff). This example shows that due to training sensitivity, learning models are either too conservative—due to lack of straggling examples—or too aggressive due to overfitting to stragglers.

The next section presents an alternative: rather than reweighting the training data (as with oversampling), reweight the prediction based on features observed at run time.

3.3 Sherlock Design

Figure 3.1 illustrates Sherlock, which consists of two complementary modules. The first, *Causal Prediction* (CP), monitors runtime behavior and predicts future stragglers. The second, *Model Interpretation* (MI), diagnoses the predictions. For simplicity, we assume that we have a *job* made of *tasks* and we want to predict which tasks will straggle. However, different frameworks use different terminology; Sherlock handles any computation (job) consisting of sub-computations (tasks). Sherlock continually monitors a job, measuring data for each task at fixed time intervals. At each time step, Sherlock runs CP to predict which tasks will straggle; i.e., whether or not they will exceed a user-specified latency threshold. For each predicted straggler, Sherlock runs MI to interpret the prediction. The interpreted results can be sent to either a human operator or another system which can proactively intervene for this task. Therefore, Sherlock will run live so both CP and MI react immediately as they receive new data.

3.3.1 Intuition and Challenges

Straggler prediction is conceptually simple: estimate latency as a function of input features; i.e., readily available metrics like CPU utilization, IO behavior, etc.. Then, tasks predicted to exceed a user-specified latency threshold are stragglers. The difficulty is that imbalanced training data

biases predictions.

Sherlock overcomes this issue by reweighting predictions based on a function of input features. The intuition is that a straggler must behave differently from a typical task and this behavior should be captured by available features. If we think of a task's features as a point in a high-dimensional (feature) space, then we expect the non-straggler points to be relatively close together and the stragglers to be far away. Using this insight, we reweight latency predictions by an amount proportional to the "distance" between the points represented by the features. This weighting will be small for non-stragglers and large for stragglers separating the predicted latencies for the two groups. We note four challenges to applying this insight:

(1) Finding a weighting function. Weighting by propensity scores (PS) has been used in other sciences to estimate causal effects once outcomes are known [10]. We propose a novel approach: applying propensity on live data to predict future stragglers, leading to the next challenge.

(2) Labeling live data. PS can be estimated via a supervised learning method called Logistic Regression [25], but doing so requires labeled training data. Since our goal is to act on live data there are no labels for tasks that have not completed. Sherlock addresses this challenge by first waiting for a small number of tasks to complete. These will not be stragglers because they finish early, and thus Sherlock labels them all non-stragglers. From that point on, Sherlock labels all other tasks stragglers and computes their PS as such. While we know most remaining tasks will not straggle, this approach is feasible because the number of stragglers is so small that the PS will be low unless the new task behaves fundamentally differently than the first tasks to complete. That is, even when mislabeled, the PS will be low because a mislabeled non-straggler's features will be "close" to those of other non-stragglers, while a straggler's features will be very "far".

(3) Correcting the weighting function. As it runs, Sherlock accumulates mislabeled examples and thus the propensity scores (computed through logistic regression) will lose the ability

to distinguish stragglers. Sherlock addresses this challenge by updating its model online once the true latency is known; i.e., when a task finishes, Sherlock relabels it with its known outcome (straggler or not) and updates the logistic regression model. Thus, Sherlock actually improves its models as it collects more data.

(4) Interpreting models. To intervene properly, we want to determine which features have the largest effect on a predicted straggler. As Sherlock uses nonlinear models, it applies *Permutation Feature Importance* (PFI) [53] for interpretation. This technique would typically be applied after the outcomes are known and would evaluate feature importance by measuring the change in prediction error as features are permuted. Since Sherlock operates on live data we make a subtle change of modifying PFI to measure importance as the change in predicted outcome instead of error.

3.3.2 Causal Prediction

Causal prediction (CP) assumes no prior knowledge or pre-training. Therefore, deployment has both a training phase—where labels are collected and the models are built—and a prediction phase, where it identifies stragglers on live data.

Training on Live Data

Sherlock treats straggler prediction as a regression problem by estimating latency first and then comparing it to a user-defined target latency. Latency predictions above the target are stragglers. We choose regression over classification because we need to reweight the latency predictions, while the outputs from classification are binary—0 for non-stragglers and 1 for stragglers. Thus reweighting classification output would result in one class always being 0. Through regression, Sherlock can reweight the predictions effectively.

Sherlock works with any standard correlation-based learner including linear models (although that is not recommended, due to low accuracy), or complicated, black-box models (e.g., Gradient

Boosting [55] or neural networks [62]). We believe Sherlock could even work with future black-box models as they become available. Our empirical analysis finds that Gradient Boosting produces the best results (see Section 3.5.5) and we thus use it as the default learner. Our evaluation also shows the generality of our approach by applying our framework with five different learners with online updates.

At the beginning of deployment, Sherlock waits for the first small number of (non-straggler) tasks to complete. These tasks are used to train a gradient boosting model in which each training sample has input features and output latency. The choice of input features depends on the specific system on which Sherlock is deployed. Ideally, Sherlock would use metrics related to scheduling, resource usage, and microarchitecture. It is important that the recorded features cover a wide range of possible causes for long-latency behavior [108]. For example, if all input features correspond to CPU usage, no framework could identify straggling tasks caused by memory behavior. Sherlock also requires that the metrics can be collected live so that it can predict stragglers in real time.

To assemble its training data on the live job, Sherlock waits for 4% of tasks to finish (this choice is justified empirically in Section 3.5.4). Those that finish early are obviously non-stragglers, and Sherlock trains the gradient boosting model on these tasks. Then at each time checkpoint, it uses the trained model to predict whether the unfinished tasks will exceed the user-specified latency threshold.

Straggler Prediction with Propensity Scores

The CP module records the features for all active tasks within a job and feeds those features into the latency predictor. Sherlock identifies tasks that will straggle using the adjusted latency prediction, which reweights the latency prediction produced by the gradient boosting model. The key (Challenge (1), above) is to mitigate prediction bias since the model is trained on data that has no straggler examples. Sherlock therefore turns differences in input features into a weight that separates the predicted latency of stragglers from non-stragglers.

Formally, we consider the predicted latency from gradient boosting to be $\hat{y}$. We want a weighting function on features such that if the features are close to those of known non-stragglers, the straggler prediction stays almost unchanged, but if the features are vastly different from known non-stragglers then the latency prediction will be scaled proportionally to the difference. Specifically, the goal is to find $ps(x)$ in Equation 3.1:

$$\hat{y}_{\text{adj}} = \frac{\hat{y}}{1 - ps(x)} \tag{3.1}$$

In Sherlock, $ps(x)$ is the probability that a task is a straggler, given features x. Thus, we want $ps(x)$ to be a value between 0 and 1 such that it is larger when the features are less similar to known non-stragglers.

Sherlock finds $ps(x)$ using logistic regression (LR) [25], a supervised learning approach that estimates the probability that a feature vector belongs to a certain binary class. Since it does not have labels for tasks that have not finished, Sherlock assumes that any running task will straggle and labels it as such (Challenge (2) from above). This deliberate mislabeling is feasible because the number of stragglers is so small that PS will be low unless the new task behaves fundamentally differently than known non-stragglers tasks. Since logistic loss is convex, Sherlock trains LR up to a linear rate of convergence and scales to large training sets using stochastic optimization [18, 102].

After estimating $ps(x)$, Sherlock gets $\hat{y}_{\text{adj}}$ from Equation 3.1. For stragglers, $\hat{y}_{\text{adj}}$ will be much longer than $\hat{y}$ since $1 - ps(x)$ will be relatively smaller. For non-stragglers, $\hat{y}_{\text{adj}}$ is almost the same as $\hat{y}$ since $1 - ps(x)$ will be close to 1.

As more tasks are evaluated, Sherlock builds up a number of mislabeled examples—non-stragglers labeled as stragglers—and it will slowly bias the results towards predicting all tasks as non-stragglers (Challenge (3)). Sherlock corrects this by dynamically updating its models once a task's true latency is revealed. Algorithm 2 summarizes causal prediction.

Algorithm 2 CP for a job at each time checkpoint.

Require: h ▷ Trained correlation-based learning model
Require: X_{train} ▷ Features for completed tasks
Require: y_{train} ▷ Latency for completed tasks
Require: X_{run} ▷ Features for running tasks
Require: c ▷ Latency threshold
  for each task $x_{\text{run}} \in X_{\text{run}}$ **do**
    if task completes **then**
      Relabel task, remove it from X_{run}, put it in X_{train} and y_{train}.
      Retrain h on new X_{train} and y_{train}.
      Go to next task.
    else
      Get initial predicted latency $\hat{y} = h(x_{\text{run}})$.
      Get estimated PS using LR $ps(x) = \text{LR}(X_{\text{train}}, y_{\text{train}}, x_{\text{run}})$.
      Get adjusted predicted latency $\hat{y}_{\text{adj}}$ based on Equation 3.1.
      Compare $\hat{y}_{\text{adj}}$ to c:
      if $\hat{y}_{\text{adj}} \geq c$ **then**
        Straggler detected, pass results to MI, trigger intervention.
      else
        Go to next task.

3.3.3 Model Interpretation

After predicting a straggler, Sherlock immediately triggers MI to interpret models. Recent research shows that causal relations can be extracted via interpreting black-box models [160]. Our goal is to find important features contributing to the prediction results. Although logistic regression is naturally interpretable due to its property of linearity, the correlation-based learning approaches Sherlock relies on (e.g., Gradient Boosting) are black-box models, which makes the combination of the correlation-based model and the weight also black-box.

To address this issue, Sherlock applies the Permutation Feature Importance (PFI) [53] to interpret the models from CP and gain insights into the straggling behavior (addressing Challenge (4)). The intuition is that feature permutation breaks the association between a feature and the prediction. A feature is "important" if its prediction changes dramatically after permuting its values because the model relies on this feature for prediction. In contrast, a feature is "unimportant" if its prediction changes very little after permutation because the model ignores that feature.

Figure 3.3 illustrates how Sherlock permutes one feature in the training data matrix, where each row represents a task and each column a feature. Given p features, to get feature j's importance—j=1 in this example—Sherlock fixes all values for other columns and permutes values across all tasks in column j to get a new training matrix. Sherlock then re-trains the correlation-based and LR models, records the new prediction result, and reports the difference between the new and original prediction as feature importance. Unlike the original PFI that computes the increase in prediction error to measure feature importance, Sherlock measures the importance of a feature by calculating the change in prediction after permuting the feature. This approach allows Sherlock to operate on live data, where the true prediction error is unknown until the task completes.

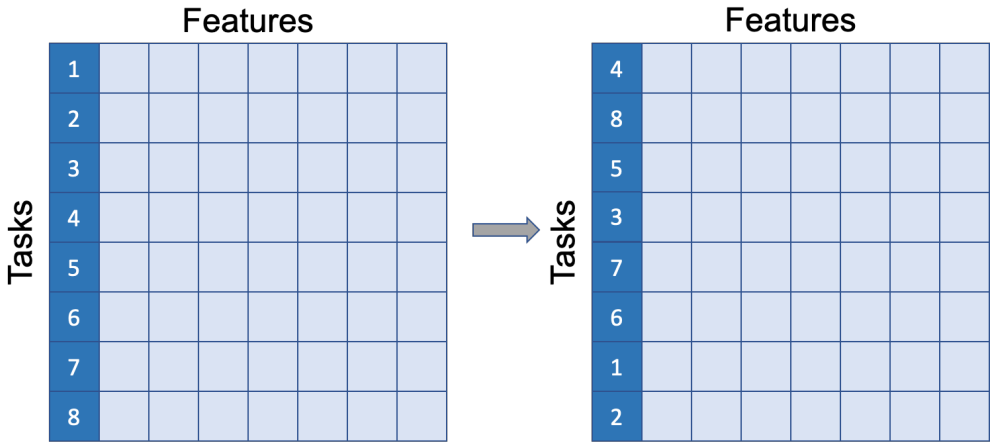


Figure 3.3: **Permuting the feature of the first column.**

Algorithm 3 Permutation feature importance for MI.

Require: h ▷ Trained correlation-based learning model
Require: LR ▷ LR model
Require: X_{train} ▷ Features for completed tasks
Require: y_{train} ▷ Latency for completed tasks
Require: x_{str} ▷ Features for predicted straggler
Require: y_{str} ▷ Predicted latency for predicted straggler
for each feature $j = 1, \ldots, p$ **do**
Get $X^{\text{perm}}_{\text{train}}$ by permuting feature j in data matrix X_{train}.
Retrain h and LR on $X^{\text{perm}}_{\text{train}}$ and y_{train} to get new prediction $y^{\text{perm}}_{\text{str}}$.
Record feature importance by computing $\Delta_j = \left| y^{\text{perm}}_{\text{str}} - y_{\text{str}} \right|$.
Sort feature importance Δ_j, $j = 1, \ldots, p$ in descending order.
Output: Sorted features in descending order.

3.3.4 Discussion and Limitations

Although PS is defined as the probability that a task will straggle given its features, it cannot be directly used to classify stragglers (we demonstrate this in Section 3.5). The reason is that due to pre-labeling, the computed PS for all unfinished tasks are greater than 0.5, which is useless in classification if a traditional threshold 0.5 is taken. Even if we use the computed PS for classification, an appropriate threshold is unknown because it is job-dependent. Therefore, PS only functions as reweighting in Sherlock for final latency prediction.

To interpret models in real-time, Sherlock runs PFI with extra cost, which is p times of model training (p is number of features). An alternative is to wait until a job completes and then interpret it with all tasks available. This strategy will require much less overhead but does not produce interpretations in real-time. This is the trade-off we consider for model interpretability, and we are willing to pay extra computation cost so that we can gain insights immediately when a straggler is identified. We evaluate overhead in Section 3.5.7.

Transfer learning is also a technique to deal with a lack of training data [114], but it is impractical for straggler prediction on live data. Transfer learning between tasks would be useful if tasks are similar to each other. However, it is hard to transfer if a new, unique workload is coming. In contrast, Sherlock would still work with that unique workload, however, so it is more robust than the transfer learning approach.

3.4 Experimental Setup

3.4.1 Evaluation Methodology

We evaluate Sherlock's ability to predict stragglers as computations run. We construct a simulator by parsing publicly available data traces and converting them into a time-series format; i.e., a series of the statistics available for each timestamp. The simulator replicates real execution by sending Sherlock the statistics that would be available at a given time.

We use two traces from different providers to demonstrate generality. Each has different computational structures and records different metrics, demonstrating that Sherlock is not tied to a particular structure or set of metrics. Generally, each trace has data for a number of computations that are divided into sub-computations. For each computation, we set a latency threshold such that any sub-computations higher than 90th percentile latency are considered stragglers (we evaluate sensitivity to this threshold below). For all evaluations, Sherlock works on live data and makes predictions about which sub-computations will straggle without first seeing any stragglers during training. This setup differentiates Sherlock from all prior work that requires appropriate representation of stragglers in training data. The simulator and Sherlock are run on a a dual socket server with two 32-core Intel Xeon Gold 6242 processors, 192GB RAM, and 2.80GHz clock speed.

3.4.2 Converting Traces to Time Series

Our evaluation uses two production cluster traces:

Google Trace. The Google trace includes 29 days of data from 12.5K machines [124]. The computations consist of a number of *jobs*, each of which has numerous *tasks*, from 100 to 9999. We filter to only include production jobs with 100 or more tasks, which reduces the 650K jobs and 25M tasks to 8425 jobs and 1.1M tasks. As detailed in Table 3.1, there are 15 metrics for each task categorized into (1) resource usage, (2) microarchitectural, and (3) scheduling. The trace contains time stamped data, which we arrange in time-order to form the time-series data set for our simulation framework.

Alibaba Trace. The Alibaba trace includes two data sets [4, 130]: (1) a 2017 trace consisting of 1.3K machines over 12 hours; (2) a 2018 trace consisting of 4K machines over 8 days. Both have the same format. We use the batch processing data set from these traces, which consists of jobs with various interdependent tasks. Each task is run as a set of instances, consisting of identical processing of different input data. Thus, for Alibaba, we treat tasks as the computation and instances as the sub-computation. Table 3.2 shows the metrics available

Table 3.1: **Task metrics in the Google Trace.**

Category	Task Metric	Descriptions
Resource Usage	MCU	Mean CPU usage
	MAXCPU	Maximum CPU usage
	SCPU	Sampled CPU usage
	CMU	Canonical memory usage
	AMU	Assigned memory usage
	MAXMU	Maximum memory usage
	UPC	Unmapped page cache memory usage
	TPC	Total page cache memory usage
	MIO	Mean disk I/O time
	MAXIO	Maximum disk I/O time
	MDK	Mean local disk space used
μarch	CPI	Cycles per instruction
	MAI	Memory accesses per instruction
SKD	EV	Number times task is evicted
	FL	Number times task fails

for each instance. We arrange the tasks as a time-series of instances based on the time order, removing non-terminated instances. We filter the tasks to those with at least 100 instances reducing the data set size to 1M tasks.

3.4.3 Points of Comparison

We compare Sherlock to the following:

1. *Correlation* (Corr): train a correlation-based model on observed tasks and predict stragglers on unseen tasks.
2. *Correlation-Oracle* (Corr-O): train a correlation-based model with prior knowledge of stragglers.
3. *LogReg* (Log): train a logistic regression model on observed tasks and predict stragglers on unseen tasks.

Table 3.2: **Instance metrics in the Alibaba Trace.**

Inst. Metric	Descriptions
ACPU	Average CPU numbers of actual instance running
MCPU	Maximum CPU numbers of actual instance running
AMEM	Average normalized memory of actual instance running
MMEM	Maximum normalized memory of actual instance running

4. *Wrangler*: a complete solution for straggler prediction using linear support vector machines and oversampling [149].
5. *Causal-Oracle* (Causal-O): apply proposed causal prediction using prior knowledge of stragglers.

We use Gradient Boosting (GB) as the default correlation learner due to its high accuracy. We update the GB model online as tasks resolve. Unlike the other methods, Wrangler is an offline method and assumes it has examples of straggling at training (which it then oversamples), while Sherlock assumes no prior knowledge of straggling. To replicate the setting in the original Wrangler paper, we randomly sample 2/3 non-straggler and straggler tasks respectively for training, which corresponds to training 2 hours and testing on the next hour as proposed in the Wrangler paper.

To summarize, *Correlation*, *LogReg* and *Sherlock* are online, while Wrangler is offline. We include LogReg to demonstrate that logistic regression by itself is not useful, but only valuable when used as a weighting function within Sherlock. The two *Oracle* techniques are not practical, but we include them to see the best correlation and causal learning could possibly achieve if they had prior knowledge of all tasks at training. The oracles are not 100% accurate because not all stragglers can be correctly predicted from the input features.

3.4.4 Evaluation Metrics

We evaluate prediction accuracy and model interpretation:

Accuracy. We report true positive rate (TPR) and false positive rate (FPR) averaged over all computations for straggler prediction. TPR and FPR are defined as

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}},$$

where TP is true positive, FN is false negative, FP is false positive, and TN is true negative. Higher TPR and lower FPR values indicate the better model.

Model Interpretation. We measure the importance of features and rank their importance in descending order. To compare different models, we compare the relevance of each model's rankings to that of the *Causal-Oracle*. The relevance can be assessed by Average Precision at k (AP@k) [127], which indicates how many of the relevant items are concentrated in the highest ranked predictions:

$$\text{AP@k} = \frac{1}{\text{TG}} \sum_{i=1}^{k} \frac{\text{TP seen}}{i},$$

where TG refers to the total number of ground truth positives for the *Causal-Oracle* list and TP seen refers to the number of true positives seen till the position k for the predicted list.

3.5 Experimental Evaluation

We empirically evaluate the following questions:

1. How accurate are Sherlock's predictions?
2. Does Sherlock identify stragglers early?
3. Does Sherlock predict extreme stragglers?
4. How sensitive is Sherlock to its design decisions?
5. Does Sherlock generalize across learners?
6. Does Sherlock correctly interpret models?

7. What is Sherlock's overhead?

3.5.1 How accurate are Sherlock's predictions?

Figure 3.4(a) shows the accuracy results. The x-axis shows TPR (higher is better) and FPR (lower is better), while the y-axis shows the aggregated results across all jobs. Correlation achieves both the lowest true positive rate (TPR) and false positive rate (FPR), which is not surprising as it is vastly impacted by training imbalance: without many positive examples of stragglers, Correlation will be conservative and predict most tasks to be non-stragglers. LogReg performs similarly with Correlation since it also suffers from training imbalance. In contrast, Sherlock has the highest TPR among non-oracle methods, and lower FPR than Wrangler. Wrangler's TPR and FPR are consistent with prior results on different data sets [149]. Since they are built with prior knowledge, the oracle methods are better than their online counterparts, and Causal-Oracle is the best (as expected). Table 3.3(a) shows the average TPR and FPR over two data sets. From this table, Sherlock and Causal-Oracle have the best effect on TPR, which validates the effectiveness of causal prediction. The two oracles have improved FPR due to their prior knowledge of the data. Sherlock has higher FPR than Correlation because (1) Correlation is trained on data biased towards non-stragglers and (2) we design Sherlock to achieve high TPR at the cost of a slight FPR increase. Wrangler, however, has the worst FPR because it overfits the oversampled straggler examples (see Section 3.2.2). *Overall, these results demonstrate that Sherlock's reweighting predictions by propensity score has a dramatic positive effect on straggler prediction, even when the data shows extreme bias; i.e., no positive examples in the training set.*

Best and Worst Case Studies for Sherlock Predictions

The best case is job with ID 6308689702 for Google and the task with job ID 2 and task ID 2 for Alibaba. In both best cases, Sherlock achieves 100% TPR almost immediately when the jobs start running, while the rest of the methods could barely predicts stragglers until the end of the

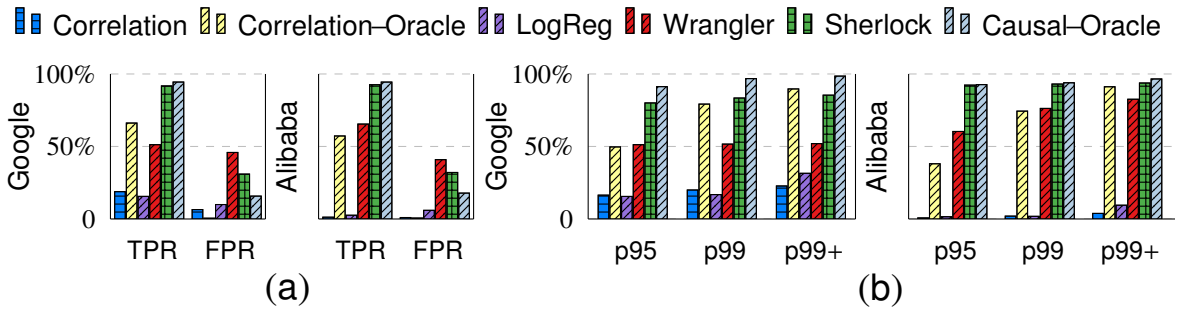


Figure 3.4: **(a) Prediction results by TPR and FPR. (b) Prediction results on long latency intervals p95, p99, and p99+.**

Table 3.3: **Average prediction results. Higher is better for TPR, p95, p99, and p99+. Lower is better for FPR.**

		Corr	Corr–O	Log	Wrangler	Sherlock	Causal–O
(a)	TPR	10%	61%	9%	58%	92%	94%
	FPR	3%	1%	8%	32%	30%	16%
(b)	p95	8 %	43%	8%	55%	86%	91%
	p99	10%	76%	9%	63%	88%	95%
	p99+	13%	90%	15%	67%	89%	97%

running time. Meanwhile, Sherlock also has as low an FPR (¡10%) as the other methods. The best case indicates that Sherlock is able to perform almost optimally by identifying stragglers early and efficiently.

In order to understand why Sherlock works almost perfectly in such cases, we inspect the normalized latency distribution of each job as shown in Figure 3.5. We can see that for both cases, non-stragglers dominate the job and are far apart from stragglers. As such, Sherlock benefits from propensity scores to separate stragglers from non-stragglers, while Correlation and Wrangler are suffer easily from such data skew.

The worst case is the job with ID 2902878525 for Google and the task with job ID 7 and task ID 11 for Alibaba. For Google, Sherlock's ability to predict stragglers is weak in the early stage and slowly increases to outperform Wrangler-1/6 and Wrangler-2/3 when the job almost completes. Although Wrangler-9/10 achieves the highest TPR, it requires a long waiting time. For Alibaba, all methods miss the true straggler prediction throughout the running time. After careful inspection of

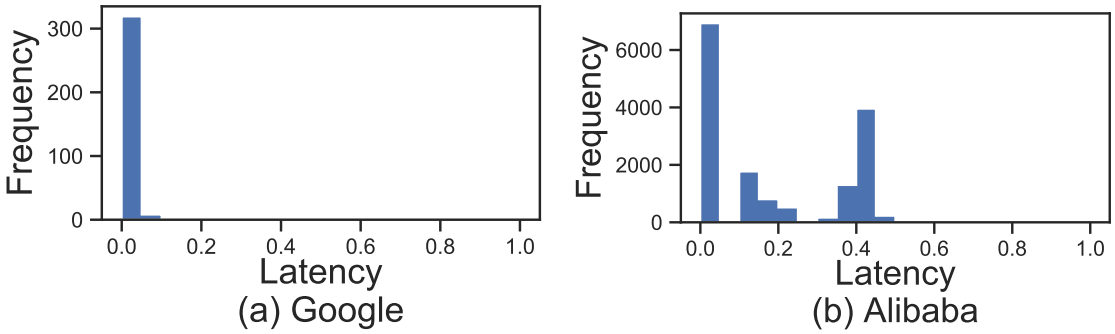


Figure 3.5: **Best cases for Google and Alibaba.**

the data, it is found that all instances within this task almost complete at same time, meaning that the variation is quite small and thus it is hard to distinguish stragglers and non-stragglers.

To understand why Sherlock works poorly in such cases, we again inspect the normalized latency distribution of each job as shown in Figure 3.6. It is obvious that for both cases, almost all tasks lie right on the border of the latency threshold and therefore it is hard for all the methods to distinguish non-stragglers and stragglers. Since Wrangler-9/10 observes data around the threshold (90%), it is no surprise that it works well.

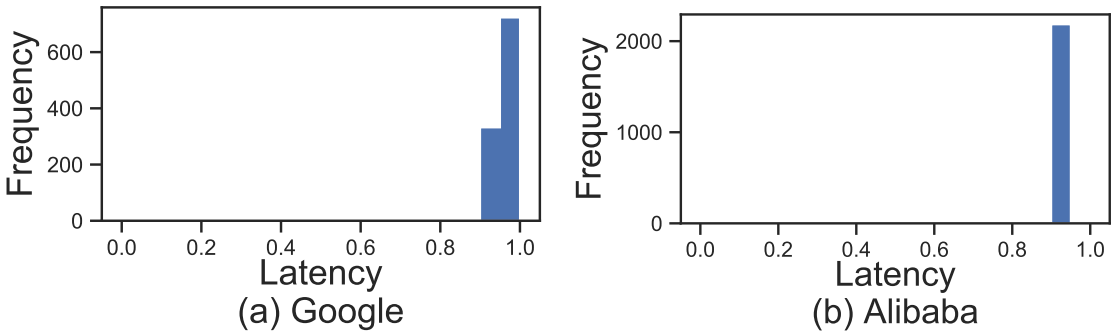


Figure 3.6: **Worst cases for Google and Alibaba.**

3.5.2 Does Sherlock identify stragglers early?

Since Sherlock works on live data, it should identify stragglers early in execution. To test this, we compute the time when all stragglers are correctly identified, measured from the start of job

execution. Identification can occur when a true positive is registered or when a false negative exceeds the latency threshold. For example, if the latency threshold is .9, the task takes one second and Sherlock predicts it to straggle at .2 seconds, we count the time of identification as .2. If that task was predicted not to straggle, its time of identification is .9, when the latency threshold is exceeded. We compare the time to identify stragglers for Correlation, LogReg, Wrangler, and Sherlock.

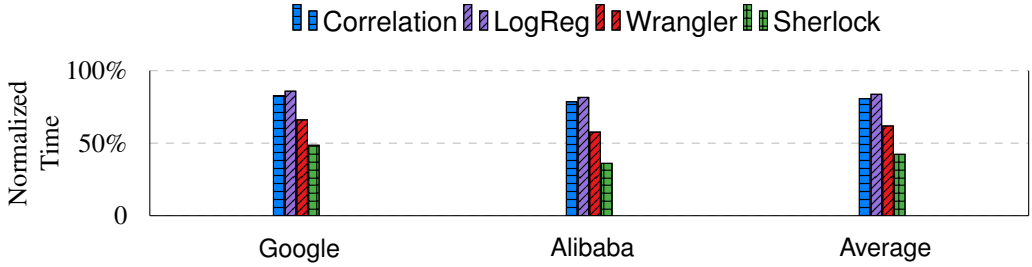


Figure 3.7: **Normalized time for straggler identification.**

Figure 3.7 shows the results, with the learner on the x-axis and the normalized time (100% means all tasks were only identified after exceeding the threshold) to identification on the y-axis. The results show that Sherlock is, by far, the fastest approach: identifying stragglers 1.91× faster than Correlation, 1.97× faster than LogReg, and 1.46× faster than Wrangler. These results include Sherlock's overhead, measured in detail below. Sherlock achieves this speed because weighting by propensity scores separates latency predictions for stragglers from non-stragglers very early in task execution. *These results show that Sherlock's causal approach identifies stragglers much earlier than other approaches, even though Sherlock has no examples of stragglers in its training set when it makes these predictions. Sherlock's speed makes it possible to intervene with the stragglers much earlier than prior work.*

3.5.3 Does Sherlock predict extreme stragglers?

Since extreme-tail tasks are major bottlenecks, we examine Sherlock's prediction performance for tail tasks. Here, we show TPR on three latency percentile intervals respectively: $[90, 95)\%$, $[95, 99)\%$, and $[99, 100]\%$, which are denoted as p95, p99, and p99+, in Figure 3.4(b) (all tail

tasks are stragglers so FPR is not available here). From this figure, we see that Sherlock achieves higher TPR than Correlation, LogReg, and Wrangler in all three intervals, and Causal-Oracle is the best. These trends are quantitatively visible in Table 3.3(b), where causal prediction methods—Sherlock and Causal-Oracle—outperform their correlation-based counterparts. *These results show that Sherlock's causal methods are, by far, the most effective at identifying extreme stragglers.*

3.5.4 How sensitive is Sherlock?

How sensitive is Sherlock to the latency threshold? We explore how Sherlock's accuracy changes as we change the threshold for straggling. Our default latency threshold is 90% and now we vary it from 70% to 95%, and re-evaluate accuracy. Figure 3.8(a) shows TPR and FPR as a function of latency threshold. With increasing latency threshold, TPR barely changes and FPR decreases around 10%, which indicates improvement of FPR. *These results show that Sherlock is robust to the definition of stragglers.*

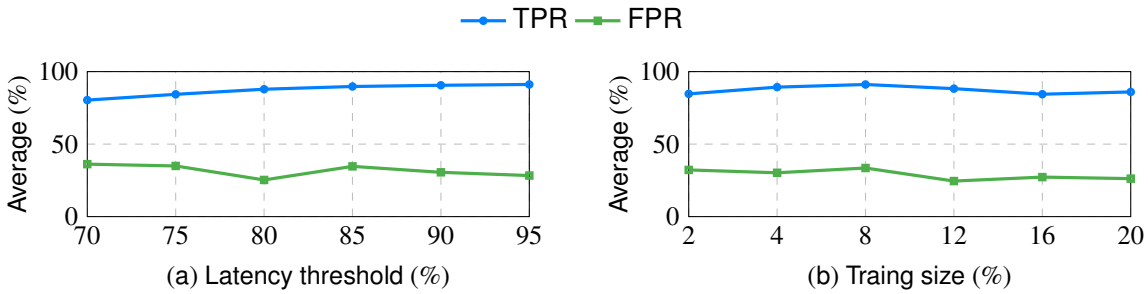


Figure 3.8: **Sensitivity to (a) threshold and (b) training set size.**

How sensitive is Sherlock to the training set size? Sherlock's default training size is 4%. Here we evaluate a range 2% to 20%. Figure 3.8(b) shows TPR and FPR as a function of the number of tasks observed in the initial training. With more training data available, TPR barely changes and FPR gets slightly better. *These results show that Sherlock is robust to the changes of training set size.*

Do online updates benefit Sherlock? Sherlock updates its model as true labels are revealed on live data. We explore how this affects Sherlock's performance. We compare TPR and FPR of

Sherlock both with and without online updates as a function of training set size in Figure 3.9. With updates Sherlock yields accurate results even when the training set is small. In contrast, without online updates, Sherlock performs poorly; i.e., barely identifying stragglers when training size is small and increasing FPR with more training data. *These results verify that online updates have a significant effect on Sherlock's performance; making this a good design choice for addressing Challenge (3) from Section 3.3.1.*

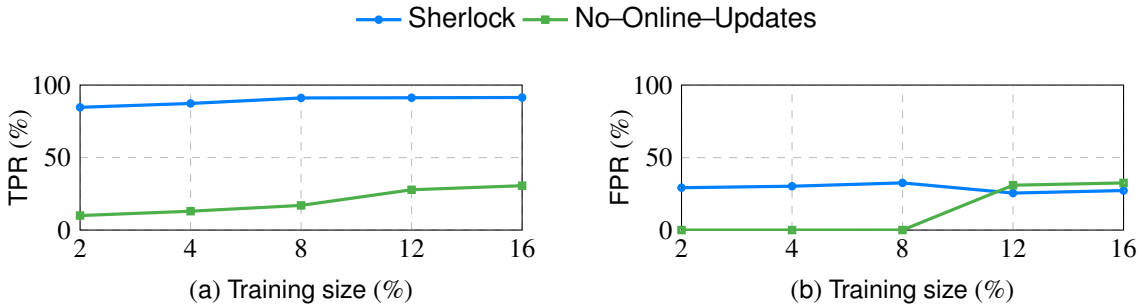


Figure 3.9: **Sherlock with and without online updates.**

How does Sherlock compare to oversampling stragglers? Sherlock uses propensity scoring to overcome training imbalance rather than oversampling; i.e., simply counting each straggler multiple times in training as is done by Wrangler [149, 150]. We justify this design choice by comparing Sherlock to Correlation, and Correlation with oversampling, i.e., Oversample-1/6 and Oversample-2/3, which randomly sample one sixth and two thirds of non-stragglers and stragglers for training respectively, and then count each straggler multiple times to balance training set. Figure 3.10 shows that oversampling improves TPR but worsens FPR compared to Correlation. However, Sherlock achieves both better TPR and FPR than oversampling. *These results justify that propensity scoring is a better approach than oversampling and it helps explain Sherlock's advantage over Wrangler.*

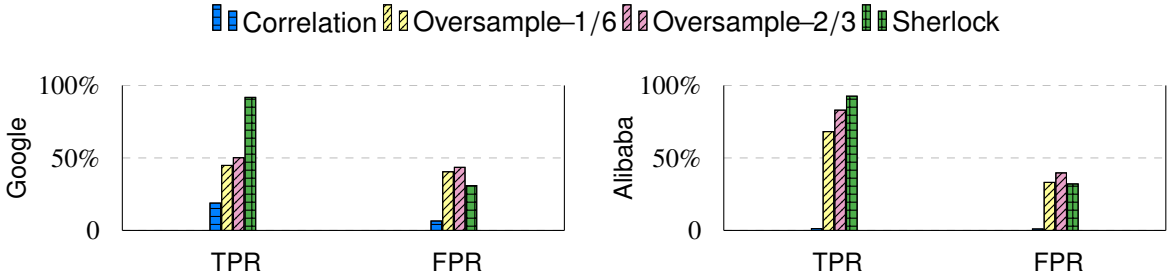


Figure 3.10: **Comparison with oversampling techniques.**

3.5.5 *Does Sherlock generalize across learners?*

Sherlock's results thus far are with the best learner (Gradient Boosting). Here, we examine its ability to generalize to different learners with *online* updates. In other words, we allow each to update its model after every task completes. This approach isolates the advantages of Sherlock's weighting procedure as all models are updated with all available information. For each learner we provide a reference to the technique itself, as well as to a prior use in a systems context. To promote reproducibility, we use the implementations from Scikit–learn for all learning models [116].

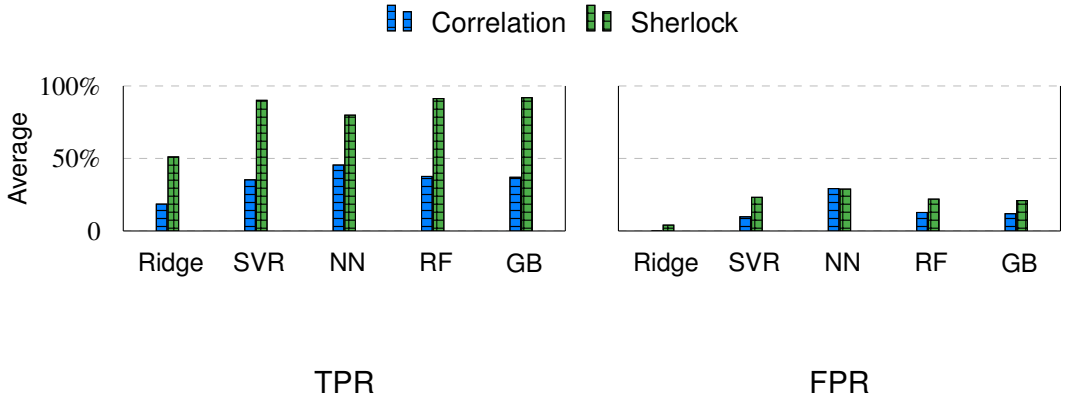


Figure 3.11: **Prediction results for all learners by TPR and FPR.**

1. *Ridge*: ℓ_2 regularized (or *ridge*) linear regression [76, 106]. The regularization parameter is 0.1.
2. *SVR*: support vector regression using nonlinear RBF kernel [34, 121]. The ℓ_2 regularization parameter is 1.
3. *NN*: multi-layer perceptron, a neural network model [62, 133]. We use three hidden layers with size 100, 50, and 10. The learning rate is 0.0001.
4. *RF*: random forest for regression [137, 33], an ensemble method. The number of trees is 100.
5. *GB*: gradient boosting for regression [55, 44], an ensemble method. The number of boosting stages is 100. This is also the default learner for all other evaluation.

Figure 3.11 shows how Sherlock improves all these correlation-based learners with online updates. Sherlock achieves overwhelmingly better TPR, especially for nonlinear and ensemble learn-

ers including SVR, RF, and GB. Sherlock NN is not as good as expected likely because it is quite difficult to optimally tune a neural network architecture for different jobs. *These results demonstrate Sherlock's generality and they are significant because they show that Sherlock's results are not tied to a particular online learning algorithm, but are a general benefit of augmenting existing correlation-based approaches with causal analysis.*

3.5.6 Does Sherlock correctly interpret models?

We validate whether Sherlock interprets models accurately with both aggregated interpretation results and case studies.

Aggregated Interpretation Results To evaluate Sherlock's interpretability we apply permutation feature importance to Correlation and both Oracles. We also compare to the features identified by Wrangler, corresponding to the highest magnitude weights in its linear model. We compare the relevance of features from each approach to that of Causal-Oracle—the best predictor—in terms of average precision at k (AP@k). For the Google trace, we group the total 15 features into 6 classes; e.g., "canonical memory usage" and "assigned memory usage" are grouped in the memory category. The Alibaba trace has 4 features and thus 4 classes. Figure 3.12 reports the relevance of top-3 and top-5 for Google and top-1 and top-3 for Alibaba. It shows that relevance with smaller k is lower, suggesting that the higher ranked features are harder to align. In particular, Correlation-Oracle is the best due to its oracle property. For non-oracle methods, Sherlock is the best—slightly better than Wrangler—meaning that Sherlock finds the most relevant features. *These results demonstrate that Sherlock's combination of propensity scoring with PFI makes it possible to gain insights into the straggling behavior in real-time.*

Case Studies We compare Sherlock's model interpretations to human expert analyses with case studies. Prior work categorized the causes for Google trace into six classes: limited memory, limited processor, I/O, data skew, cache bottleneck, and scheduling [161]. We do not have case studies for Alibaba as it has only four features [68].

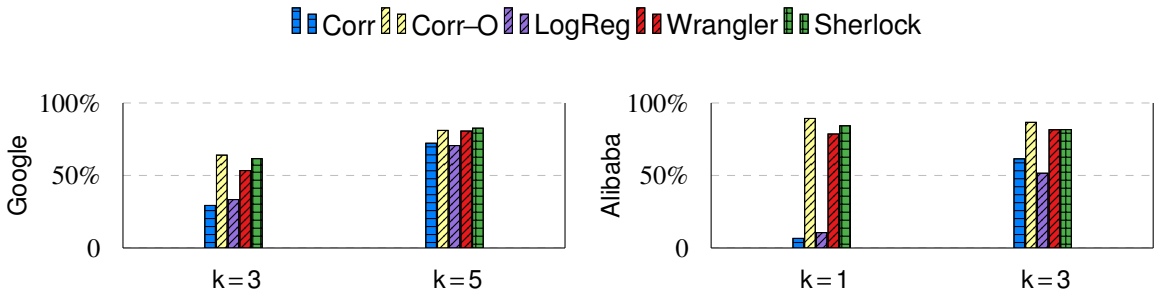
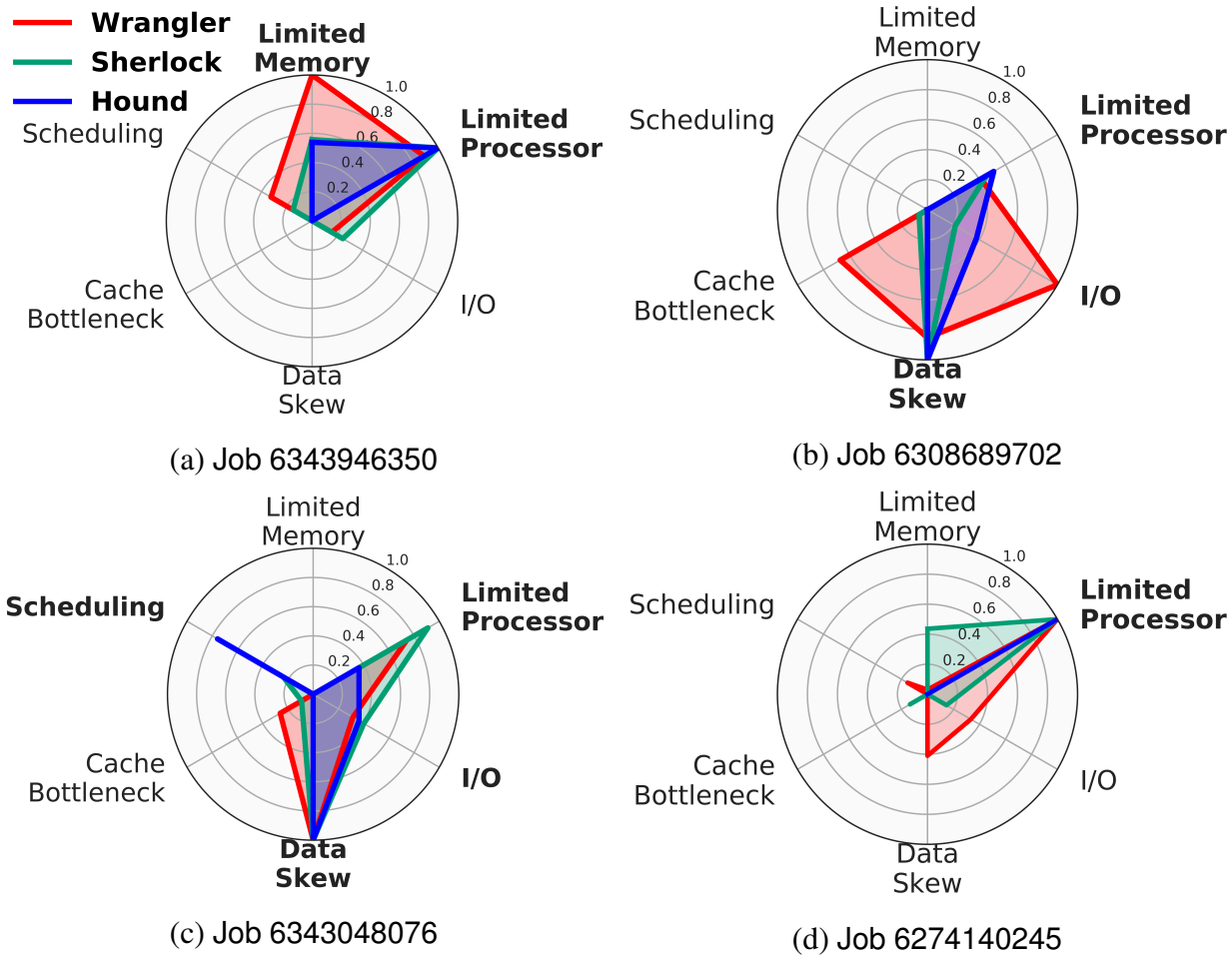


Figure 3.12: **Relevance results by AP@k. (Higher is better.)**

Figure 3.13 shows four studies where a job in the Google trace was analyzed to identify causes [161]. We compare Wrangler and Sherlock to Hound [161], which diagnoses causes of straggling (although it operates after the job has completed and does not make predictions). We use radar charts to display multi-dimensional causes and their relative strength, where bold labels are causes identified by human experts. We can see that the area covered by Sherlock and Wrangler heavily overlaps the area covered by Hound, and their strengths are consistent as well. In practice, causes depend on each other, and therefore it is likely for Sherlock and Wrangler to cover wider areas than the human experts, which indicates more possible causes than human labels. *These case studies further demonstrate Sherlock's ability to diagnose causes by discovering important features. We stress that Sherlock achieves roughly equal causal analysis as Hound and Wrangler, but arrives at this diagnosis much earlier (see Figure 3.7).*

3.5.7 What is Sherlock's overhead?

We collect overheads for all practical frameworks in Table 3.4 and 3.5, which show the number of computations per second predicted and interpreted on a single server with both a single core (S) and all available cores (P) on our testbed (see Section 3.4.1), respectively. In Table 3.4, Wrangler is the computationally cheapest compared to others due to linearity, while Correlation and Sherlock are built on Gradient Boosting—a nonlinear ensemble with better accuracy at the cost of higher overhead. The extra overhead from Sherlock compared to Correlation is due to logistic regression. We believe the overhead is tolerable for the benefit of more accurate straggler prediction in real



(a) Job 6343946350

(b) Job 6308689702

(c) Job 6343048076

(d) Job 6274140245

Figure 3.13: **Case studies of mixture of causes on example jobs. Causes by human labels from Hound [161] are in bold.**

time.

Table 3.4: **Number of tasks predicted per second on a server. (S: Serial. P: Parallel.)**

	Corr	LogReg	Wrangler	Sherlock
S	39,959	89,204	227,495	27,597
P	1,280,002	2,480,032	8,209,14	844,260

Table 3.5 shows Sherlock's performance for model interpretation, illustrating the number of tasks diagnosed per second with both a single core (S) and all available cores (P). The performance heavily depends on the number of features, so Alibaba is much faster than Google. Interpretation is obviously expensive compared to prediction, but it enables Sherlock to diagnose causes in real-

Table 3.5: **Number of tasks interpreted per second on a server. (S: Serial. P: Parallel.)**

	Google	Alibaba
S	2	9
P	63	211

time. In addition, the number of predicted stragglers is low, so the overhead of PFI will be paid much less frequently than that of the causal prediction.

3.6 Conclusion

This chapter introduces Sherlock, a straggler prediction framework that makes accurate live predictions without assuming prior knowledge or carefully curated training data by using techniques from causal analysis. Sherlock's key insight is that the measurable behavior (features) of non-stragglers and stragglers must be different even before the long latency reveals itself. As such, Sherlock employs propensity scores (PS) to quantify such difference. Although PS has been applied in post-facto causal inference in other sciences, Sherlock is the first to illustrate that it can be used for live predictions. Sherlock also proposes permutation feature importance to interpret black-box models to gain insights into the straggling behavior. We find that Sherlock predicts stragglers accurately and offers reliable model interpretability on live data before it sees any positive straggling examples in training. It is the only approach we are aware that accurately predicts stragglers with no prior examples of straggling behavior. This work is strong evidence that causal modeling has advantages over conventional correlation-based modeling when training data is not carefully curated. We hope this work inspires systems researchers to consider the nature of training data before applying learning, and incorporate causal analysis to solve systems problems.

CHAPTER 4

FUTURE WORK

ML has influenced much recent systems research including providing systems support for ML [131, 134] and applying ML to solve systems problems [40, 41, 107]. To further produce generalizable, robust, and interpretable results, it is crucial to understand the structure of the system problems and incorporate causal analysis. This dissertation constitutes a new understanding of ML for systems that opens the way to a novel class of methodology and application by incorporating causal inference techniques.

Looking forward, our goal is to extend the straggling prediction work to performance diagnosis and system intervention. By understanding both software- and hardware-level effects on straggling behavior, we will intervene in systems proactively and further deliver robust systems. The details are listed below:

- We will develop a causal model to configure software systems by understanding the latent structure. Software infrastructure has a huge number of configuration parameters that affect performance and misconfigurations are a leading cause of down time. Different from prior work by analyzing source code [148] or control theory [144], we will learn a more effective way to configure software systems with causal models—causal directed acyclic graphs (DAGs) [115]—so as to intervene in the software systems.
- We will further generalize software configurations to hardware characteristics to understand how software and hardware interact. Traditionally, software and hardware are studied separately. With causal models, we intend to configure software and hardware features simultaneously to develop cross-stack characterization and scheduling tools to eliminate stragglers in essence.
- We will take lessons from applying causal techniques to systems and further bridge the gap between empirical and theoretical research in causal inference by explaining why it works.

The ultimate goal is to allow ML-based system to tune application, system software, and hard-

ware parameters to meet system optimization goals for increasingly complicated system designs. Broadly, as access to computing becomes ever more valuable in science and society, the improved system efficiency and usability that our work offers will make it possible to put computing in the hands of more people.

REFERENCES

[1] Atul Adya, Daniel Myers, Jon Howell, Jeremy Elson, Colin Meek, Vishesh Khemani, Stefan Fulger, Pan Gu, Lakshminath Bhuvanagiri, Jason Hunter, Roberto Peon, Larry Kai, Alexander Shraer, Arif Merchant, and Kfir Lev-Ari. Slicer: Auto-sharding for datacenter applications. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 739–753. USENIX Association, 2016.

[2] Akshay Agrawal, Akshay Naresh Modi, Alexandre Passos, Allen Lavoie, Ashish Agarwal, Asim Shankar, Igor Ganichev, Josh Levenberg, Mingsheng Hong, Rajat Monga, et al. Tensorflow eager: A multi-stage, python-embedded dsl for machine learning. *arXiv preprint arXiv:1903.01855*, 2019.

[3] Mehmet Fatih Aktas and Emina Soljanin. Straggler mitigation at scale. *IEEE/ACM Transactions on Networking*, 27(6):2266–2279, 2019.

[4] Alibaba. Alibaba production cluster trace data. *https://github.com/alibaba/clusterdata*.

[5] Ganesh Ananthanarayanan, Sameer Agarwal, Srikanth Kandula, Albert Greenberg, Ion Stoica, Duke Harlan, and Ed Harris. Scarlett: Coping with skewed content popularity in mapreduce clusters. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys'11, pages 287–300, New York, NY, USA, 2011.

[6] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, NSDI'13, pages 185–198, USA, 2013.

[7] Ganesh Ananthanarayanan, Srikanth Kandula, Albert Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. Reining in the outliers in map-reduce clusters using mantri. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 265–278, USA, 2010.

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

[9] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[10] Peter C. Austin. An introduction to propensity score methods for reducing the effects of confounding in observational studies. *Multivariate Behavioral Research*, 46(3):399–424, 2011.

[11] Grant Ayers, Nayana Prasad Nagendra, David I August, Hyoun Kyu Cho, Svilen Kanev, Christos Kozyrakis, Trivikram Krishnamurthy, Heiner Litz, Tipp Moseley, and Parthasarathy Ranganathan. Asmdb: understanding and mitigating front-end stalls in warehouse-scale computers. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 462–473, 2019.

[12] L.A Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, Dec 2007.

[13] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. Ix: A protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 49–65, 2014.

[14] R. M. Bell, Y. Koren, and C. Volinsky. The bellkor 2008 solution to the netflix prize. Technical report, ATandT Labs, 2008.

[15] Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V Gratz, and Daniel A Jiménez. Perceptron-based prefetch filtering. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–13. IEEE, 2019.

[16] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, 2008.

[17] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO'41, pages 318–329, USA, 2008.

[18] Léon Bottou. Large-scale machine learning with stochastic gradient descent. Springer, 2010.

[19] J. Cai, E. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

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

[21] Yinzhi Cao, Alexander Fangxiao Yu, Andrew Aday, Eric Stahl, Jon Merwine, and Junfeng Yang. Efficient repair of polluted machine learning systems via causal unlearning. In *Proceedings of the 13th ACM ASIA Conference on Information, Computer and Communications Security*, 2018.

[22] Aaron Carroll and Gernot Heiser. Mobile multicores: Use them or waste them. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '13, pages 12:1–12:5, New York, NY, USA, 2013. ACM.

[23] Marcus Carvalho, Walfredo Cirne, Francisco Vilar Brasileiro, and John Wilkes. Long-term slos for reclaimed cloud computing resources. In Ed Lazowska, Doug Terry, Remzi H. Arpaci-Dusseau, and Johannes Gehrke, editors, *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 3-5, 2014*, pages 20:1–20:13. ACM, 2014.

[24] Adrian M Caulfield, Eric S Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, et al. A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016.

[25] M Soledad Cepeda, Ray Boston, John T Farrar, and Brian L Strom. Comparison of logistic regression versus propensity score when the number of events is low and there are multiple confounders. *American journal of epidemiology*, 158(3):280–287, 2003.

[26] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[27] 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 *IISWC*, 2009.

[28] Chi-Ou Chen, Ye-Qi Zhuo, Chao-Chun Yeh, Che-Min Lin, and Shih-Wei Liao. Machine learning-based configuration parameter tuning on hadoop system. In *BigData Congress*, 2015.

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

[30] Jian Chen, Lizy Kurian John, and Dimitris Kaseridis. Modeling program resource demand using inherent program characteristics. *SIGMETRICS Perform. Eval. Rev.*, 39(1):1–12, June 2011.

[31] Seungryul Choi and Donald Yeung. Learning-based smt processor resource distribution via hill-climbing. In *ISCA*, 2006.

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

[33] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, Boston, MA, March 2017. USENIX Association.

[34] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[35] Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. 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*, pages 86–99, 2017.

[36] Charlie Curtsinger and Emery D Berger. Coz: Finding code that counts with causal profiling. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 184–197, 2015.

[37] Jeffrey Dean. Achieving rapid response times in large online services. 2012.

[38] Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013.

[39] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation - Volume 6*, OSDI'04, page 10, USA, 2004.

[40] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'13, pages 77–88, New York, NY, USA, 2013.

[41] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'14, pages 127–144, New York, NY, USA, 2014.

[42] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.

[43] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Fred Chong. Memory cocktail therapy: A general learning-based framework to optimize dynamic tradeoffs in nvm. In *MICRO*, 2017.

[44] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Frederic T. Chong. Memory cocktail therapy: A general learning-based framework to optimize dynamic tradeoffs in nvms. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-50'17, pages 232–244, New York, NY, USA, 2017.

[45] 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.

[46] Thanh Do, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, and Haryadi S. Gunawi. Limplock: Understanding the impact of limpware on scale-out cloud

systems. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC'13, New York, NY, USA, 2013.

[47] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, 2017.

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

[49] Christophe Dubach, Timothy M. Jones, Edwin V. Bonilla, and Michael F. P. O'Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 43rd Annual International Symposium on Microarchitecture*, MICRO'43, pages 485–496, 2010.

[50] Andrew Estabrooks, Taeho Jo, and Nathalie Japkowicz. A multiple resampling method for learning from imbalanced data sets. *Computational intelligence*, 20(1):18–36, 2004.

[51] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 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, pages 226–231. AAAI Press, 1996.

[52] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[53] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously. *Journal of Machine Learning Research*, 20(177):1–81, 2019.

[54] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. A configurable cloud-scale dnn processor for real-time ai. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–14. IEEE, 2018.

[55] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

[56] Andrei Frumusanu. Improving the exynos 9810 galaxy s9: Part 2 - catching up with the snapdragon. *AnandTech*, April 2018.

[57] Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson. A case for machine learning to optimize multicore performance. In *First USENIX Workshop on Hot Topics in Parallelism (HotPar'09)*, 2009.

[58] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *22nd International*

Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Apr 2017.

[59] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators. In *24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Apr 2019.

[60] Elba Garza, Samira Mirbagher-Ajorpaz, Tahsin Ahmad Khan, and Daniel A Jiménez. Bit-level perceptron prediction for indirect branches. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 27–38. IEEE, 2019.

[61] EP George, J Stuart Hunter, William G Hunter, Roma Bins, Kay Kirlin IV, and Destiny Carroll. Statistics for experimenters: design, innovation, and discovery. 2005.

[62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.

[63] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018.

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

[65] Shuhang Gu, Qi Xie, Deyu Meng, Wangmeng Zuo, Xiangchu Feng, and Lei Zhang. Weighted nuclear norm minimization and its applications to low level vision. *Int. J. Comput. Vision*, 121(2):183–208, January 2017.

[66] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), August 2018.

[67] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S. McKinley, and Björn B. Brandenburg. Swayam: distributed autoscaling to meet slas of machine learning inference services with resource efficiency. In K. R. Jayaram, Anshul Gandhi, Bettina Kemme, and Peter R. Pietzuch, editors, *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Las Vegas, NV, USA, December 11 - 15, 2017*, pages 109–120. ACM, 2017.

[68] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: An analysis of alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[69] Maya R. Gupta and Yihua Chen. Theory and use of the em algorithm. *Found. Trends Signal Process.*, 4(3):223–296, March 2011.

[70] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, WWC '01, pages 3–14, Washington, DC, USA, 2001. IEEE Computer Society.

[71] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[72] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*, pages 29–42. ACM, 2017.

[73] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O. Suminto, Cesar A. Stuardo, Andrew A. Chien, and Haryadi S. Gunawi. Mittos: Supporting millisecond tail tolerance with fast rejecting slo-aware os interface. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP'17', pages 168–183, New York, NY, USA, 2017.

[74] Md. E. Haque, Yong Hun Eom, Yuxiong He, Sameh Elnikety, Ricardo Bianchini, and Kathryn S. McKinley. Few-to-many: Incremental parallelism for reducing tail latency in interactive services. In Özcan Özturk, Kemal Ebcioglu, and Sandhya Dwarkadas, editors, *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS'15, Istanbul, Turkey, March 14-18, 2015*, pages 161–175. ACM, 2015.

[75] Md Enamul Haque, Yuxiong He, Sameh Elnikety, Thu D. Nguyen, Ricardo Bianchini, and Kathryn S. McKinley. Exploiting heterogeneity for tail latency and energy efficiency. In Hillery C. Hunter, Jaime Moreno, Joel S. Emer, and Daniel Sánchez, editors, *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18, 2017*, pages 625–638. ACM, 2017.

[76] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., 2001.

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

[78] Henry Hoffmann. Jouleguard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP'15, pages 198–214, New York, NY, USA, 2015.

[79] Guido W Imbens and Donald B Rubin. *Causal inference in statistics, social, and biomedical sciences*. Cambridge University Press, 2015.

[80] Connor Imes, David H. K. Kim, Martina Maggio, and Henry Hoffmann. Poet: A portable approach to minimizing energy under soft real-time constraints. In *RTAS*, 2015.

[81] Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana. Self-optimizing memory controllers: A reinforcement learning approach. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA'08, pages 39–50, USA, 2008.

[82] Syed Muhammad Zeeshan Iqbal, Yuchen Liang, and Hakan Grahn. Parmibench - an open-source benchmark for embedded multiprocessor systems. *IEEE Comput. Archit. Lett.*, 9(2), July 2010.

[83] Anand Jayarajan, Jinliang Wei, Garth Gibson, Alexandra Fedorova, and Gennady Pekhimenko. Priority-based parameter propagation for distributed dnn training. *arXiv preprint arXiv:1905.03960*, 2019.

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

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

[86] David H. K. Kim, Connor Imes, and Henry Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *CPSNA*, 2015.

[87] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Selecta: Heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 759–773, 2018.

[88] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009.

[89] Felix Last, Georgios Douzas, and Fernando Bacao. Oversampling for imbalanced learning based on k-means and smote. *arXiv preprint arXiv:1711.00837*, 2017.

[90] Mathias Lécuyer, Riley Spahn, Kiran Vodrahalli, Roxana Geambasu, and Daniel Hsu. Privacy accounting and quality control in the sage differentially private ml platform. In Tim Brecht and Carey Williamson, editors, *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019*, pages 181–195. ACM, 2019.

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

[92] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS*, 2006.

[93] Benjamin C. Lee and David M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGPLAN Not.*, 41(11):185–194, October 2006.

[94] Benjamin C. Lee and David M. Brooks. Applied inference: Case studies in microarchitectural design. *TACO*, 7(2):8:1–8:37, 2010.

[95] 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, PPOPP 2007, San Jose, California, USA, March 14-17, 2007*, pages 249–258, 2007.

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

[97] Jing Li, Kunal Agrawal, Sameh Elnikety, Yuxiong He, I-Ting Angelina Lee, Chenyang Lu, and Kathryn S. McKinley. Work stealing for interactive services to meet target latency. In Rafael Asenjo and Tim Harris, editors, *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2016, Barcelona, Spain, March 12-16, 2016*, pages 14:1–14:13. ACM, 2016.

[98] Zhenhong Liu, Amir Yazdanbakhsh, Taejoon Park, Hadi Esmaeilzadeh, and Nam Sung Kim. Simul: An algorithm-driven approximate multiplier design for machine learning. *IEEE Micro*, 38(4):50–59, 2018.

[99] Haonan Lu, Christopher Hodsdon, Khiem Ngo, Shuai Mu, and Wyatt Lloyd. The snow theorem and latency-optimal read-only transactions. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 135–150, Savannah, GA, November 2016. USENIX Association.

[100] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. Tabla: A unified template-based framework for accelerating statistical machine learning. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 14–26. IEEE, 2016.

[101] Mostafa Mahmoud, Kevin Siu, and Andreas Moshovos. Diffy: A déjà vu-free differential deep neural network accelerator. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 134–147. IEEE, 2018.

[102] Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Semantics-preserving parallelization of stochastic gradient descent. In *2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 21-25, 2018*, pages 224–233. IEEE Computer Society, 2018.

[103] J. F. Martinez and E. Ipek. Dynamic multicore resource management: A machine learning approach. *IEEE Micro*, 29(5):8–17, Sept 2009.

[104] 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.

[105] 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 2017, Columbus, OH, USA, July 17-21, 2017*, pages 125–134, 2017.

[106] 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 2017, Columbus, OH, USA, July 17-21, 2017*, pages 125–134, 2017.

[107] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'15, pages 267–281, New York, NY, USA, 2015.

[108] Jeffrey C. Mogul and John Wilkes. Nines are not enough: Meaningful metrics for clouds. In *Proceedings of the Workshop on Hot Topics in Operating Systems, HotOS 2019, Bertinoro, Italy, May 13-15, 2019*, pages 136–141. ACM, 2019.

[109] Lev Mukhanov, Konstantinos Tovletoglou, Hans Vandierendonck, Dimitrios S Nikolopoulos, and Georgios Karakonstantis. Workload-aware dram error prediction using machine learning. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, pages 106–118. IEEE, 2019.

[110] R. Narayanan, B. Ozisikyilmaz, J. Zambreno, G. Memik, and A. Choudhary. Minebench: A benchmark suite for data mining workloads. In *IISWC*, 2006.

[111] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-tolerant software distributed shared memory. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 291–305, 2015.

[112] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

[113] Adam J. Oliner, Anand P. Iyer, Ion Stoica, Eemil Lagerspetz, and Sasu Tarkoma. Carat: Collaborative energy diagnosis for mobile devices. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys'13, pages 10:1–10:14, New York, NY, USA, 2013. ACM.

[114] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[115] Judea Pearl. *Causality*. Cambridge university press, 2009.

[116] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit–learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[117] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 1–18. ACM, 2017.

[118] Drew D Penney and Lizhong Chen. A survey of machine learning applied to computer architecture design. *arXiv preprint arXiv:1909.12373*, 2019.

[119] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, and Christine A. Shoemaker. Flicker: A dynamically adaptive architecture for power limited multicore systems. In *ISCA*, 2013.

[120] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, and Christine A. Shoemaker. Flicker: a dynamically adaptive architecture for power limited multicore systems. In *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*, pages 13–23. ACM, 2013.

[121] Paula Petrica, Adam M. Izraelevitz, David H. Albonesi, and Christine A. Shoemaker. Flicker: A dynamically adaptive architecture for power limited multicore systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA'13, page 1323, New York, NY, USA, 2013.

[122] Dmitry Ponomarev, Gurhan Kucuk, and Kanad Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *MICRO*, 2001.

[123] K. V. Rashmi, Mosharaf Chowdhury, Jack Kosaian, Ion Stoica, and Kannan Ramchandran. Ec-cache: Load-balanced, low-latency cluster caching with online erasure coding. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*, pages 401–417. USENIX Association, 2016.

[124] Charles Reiss, John Wilkes, and Joseph L Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, pages 1–14, 2011.

[125] Xiaoqi Ren, Ganesh Ananthanarayanan, Adam Wierman, and Minlan Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM'15, pages 379–392, New York, NY, USA, 2015.

[126] Daniel Richins, Dharmisha Doshi, Matthew Blackmore, Aswathy Thulaseedharan Nair, Neha Pathapati, Ankit Patel, Brainard Daguman, Daniel Dobrijalowski, Ramesh Illikkal, Kevin Long, et al. Missing the forest for the trees: End-to-end ai application performance in edge data centers. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 515–528. IEEE, 2020.

[127] Stephen Robertson. A new interpretation of average precision. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 689–690, 2008.

[128] Paul R Rosenbaum and Donald B Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

[129] Malte Schwarzkopf and Peter Bailis. Research for practice: cluster scheduling for datacenters. *Commun. ACM*, 61(5):50–53, 2018.

[130] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. Legoos: A disseminated, distributed os for hardware resource disaggregation. In Andrea C. Arpaci-Dusseau and Geoff Voelker, editors, *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 69–87. USENIX Association, 2018.

[131] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–27, 2019.

[132] Sayeh Sharify, Alberto Delmas Lascorz, Mostafa Mahmoud, Milos Nikolic, Kevin Siu, Dylan Malone Stuart, Zissis Poulos, and Andreas Moshovos. Laconic deep learning inference acceleration. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 304–317. IEEE, 2019.

[133] Zhan Shi, Xiangru Huang, Akanksha Jain, and Calvin Lin. Applying deep learning to the cache replacement problem. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pages 413–425, New York, NY, USA, 2019.

[134] Muthian Sivathanu, Tapan Chugh, Sanjay S. Singapuram, and Lidong Zhou. Astra: Exploiting predictability to optimize deep learning. In Iris Bahar, Maurice Herlihy, Emmett Witchel, and Alvin R. Lebeck, editors, *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*, pages 909–923. ACM, 2019.

[135] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: A platform for os-level power management. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys'09, pages 289–302, New York, NY, USA, 2009.

[136] Srinath Sridharan, Gagan Gupta, and Gurindar S. Sohi. Holistic run-time parallelism management for time and energy efficiency. In *ICS*, 2013.

[137] Leo Breiman Statistics and Leo Breiman. Random forests. In *Machine Learning*, pages 5–32, 2001.

[138] G. Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11, 2007.

[139] Erik Tomusk, Christophe Dubach, and Michael F. P. O'Boyle. Four metrics to evaluate heterogeneous multicores. *TACO*, 12(4):37:1–37:25, 2016.

[140] Erik Tomusk, Christophe Dubach, and Michael F. P. O'Boyle. Selecting heterogeneous cores for diversity. *TACO*, 13(4):49:1–49:25, 2016.

[141] Emma Tosch, Eytan Bakshy, Emery D Berger, David D Jensen, and J Eliot B Moss. Planalyzer: assessing threats to the validity of online experiments. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–30, 2019.

[142] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *SIGMOD*, 2017.

[143] Kenzo Van Craeynest, Aamer Jaleel, Lieven Eeckhout, Paolo Narvaez, and Joel Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12, pages 213–224, Washington, DC, USA, 2012. IEEE Computer Society.

[144] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. Understanding and auto-adjusting performance-sensitive configurations. *ACM SIGPLAN Notices*, 53(2):154–168, 2018.

[145] Gary M Weiss and Foster Provost. The effect of class distribution on classifier learning: an empirical study. 2001.

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

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

[148] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early detection of configuration errors to reduce failure damage. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, Savannah, GA, November 2016.

[149] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. Wrangler: Predictable and faster jobs using fewer resources. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC'14, pages 1–14, New York, NY, USA, 2014.

[150] Neeraja Jayant Yadwadkar, Bharath Hariharan, Joseph Gonzalez, and Randy H. Katz. Faster jobs in distributed data processing using multi-task learning. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 532–540. SIAM, 2015.

[151] Xi Yang, Stephen M. Blackburn, and Kathryn S. McKinley. Elfen scheduling: Fine-grain principled borrowing from latency-critical workloads using simultaneous multithreading. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 309–322, Denver, CO, June 2016. USENIX Association.

[152] Amir Yazdanbakhsh, Kambiz Samadi, Nam Sung Kim, and Hadi Esmaeilzadeh. Ganax: A unified mimd-simd acceleration for generative adversarial networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 650–661. IEEE, 2018.

[153] Joshua J. Yi, David J. Lilja, and Douglas M. Hawkins. A statistically rigorous approach for improving simulation methodology. In *HPCA*, 2003.

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

[155] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 29–42, USA, 2008.

[156] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *ASPLOS*, 2016.

[157] Xiao Zhang, Rongrong Zhong, Sandhya Dwarkadas, and Kai Shen. A flexible framework for throttling-enabled multicore management (temm). In *ICPP*, 2012.

[158] Xu Zhang, Qingwei Lin, Yong Xu, Si Qin, Hongyu Zhang, Bo Qiao, Yingnong Dang, Xinsheng Yang, Qian Cheng, Murali Chintalapati, Youjiang Wu, Ken Hsieh, Kaixin Sui, Xin Meng, Yaohai Xu, Wenchi Zhang, Furao Shen, and Dongmei Zhang. Cross-dataset time series anomaly detection for cloud systems. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1063–1076, Renton, WA, July 2019. USENIX Association.

[159] Yiying Zhang and Yutong Huang. "learned": Operating systems. *Operating Systems Review*, 53(1):40–45, 2019.

[160] Qingyuan Zhao and Trevor Hastie. Causal interpretations of black-box models. *Journal of Business and Economic Statistics*, pages 1–10, 2019.

[161] Pengfei Zheng and Benjamin C. Lee. Hound: Causal learning for datacenter-scale straggler diagnosis. In *the 2018 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'18, pages 59–61, New York, NY, USA, 2018.

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

[163] Yuhao Zhu and Vijay Janapa Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013.

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