Building Efficient and Practical Machine Learning Systems

Qinghao Hu

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

20/07/2023

Date

Qinghao Hu

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

20/07/2023

Date

Prof. Tianwei Zhang

Authorship Attribution Statement

This thesis contains material from four papers published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

Chapter 2 is published as *Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang.* ***Characterization and prediction of deep learning workloads in large-scale gpu datacenters****. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '21, 2021.*

The contributions of the co-authors are as follows:

- Prof. Tianwei Zhang, Prof. Yonggang Wen, and Dr. Peng Sun played pivotal roles in shaping the initial project direction, offering invaluable insights, engaging in constructive discussions regarding the system design, and making significant revisions to the manuscript draft.
- I proposed the idea, implemented the system framework, conducted the experiments and wrote the manuscript.
- Dr. Peng Sun and Dr. Shengen Yan collected dataset used in experiments and further preprocess the data.

Chapter 3 is published as *Qinghao Hu, Zhisheng Ye, Meng Zhang, Qiaoling Chen, Peng Sun, Yonggang Wen, and Tianwei Zhang.* ***Hydro: Surrogate-based hyperparameter tuning service in datacenters****. In USENIX Symposium on Operating Systems Design and Implementation, OSDI '23, 2023.*

The contributions of the co-authors are as follows:

- Prof. Tianwei Zhang, Prof. Yonggang Wen, and Dr. Peng Sun played pivotal roles in shaping the initial project direction, offering invaluable insights, engaging in constructive discussions regarding the system design, and making significant revisions to the manuscript draft.
- I proposed the idea, implemented the system framework of Hydro Tuner, conducted the experiments and wrote the manuscript.
- Zhisheng Ye and Qiaoling Chen implemented and evaluated the prototype of Bubble Squeezer, and assisted me to write the partial manuscript.
- Meng Zhang assisted me in setting up the experimental environment, debugging code scripts, and plotting various figures.

Chapter 4 is published as *Qinghao Hu*, Meng Zhang*, Peng Sun, Yonggang Wen, and Tianwei Zhang.* ***Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs****. In Architectural Support for Programming Languages and Operating Systems, ASPLOS '23, 2023.*

The contributions of the co-authors are as follows:

- Prof. Tianwei Zhang, Prof. Yonggang Wen, and Dr. Peng Sun played pivotal roles in shaping the initial project direction, offering invaluable insights, engaging in constructive discussions regarding the system design, and making significant revisions to the manuscript draft.

- I proposed the idea, implemented the system framework, conducted the experiments and wrote the manuscript.
- Meng Zhang and I made equal contributions to this work. She played a crucial role in assisting me in conducting physical experiments and creating visual representations of the results.

Chapter 5 is published as *Qinghao Hu, Harsha Nori, Peng Sun, Yonggang Wen, and Tianwei Zhang.* ***Primo: Practical learningaugmented systems with interpretable models****. In USENIX Annual Technical Conference, USENIX ATC '22, 2022.*

The contributions of the co-authors are as follows:

- Prof. Tianwei Zhang, Prof. Yonggang Wen, and Dr. Peng Sun played pivotal roles in shaping the initial project direction, offering invaluable insights, engaging in constructive discussions regarding the system design, and making significant revisions to the manuscript draft.
- I proposed the idea, implemented the system framework, conducted the experiments and wrote the manuscript.
- Harsha Nori implemented the Monotonic Constraint module and assisted me in writing the manuscript.

20/07/2023

Date

Qinghao Hu

Acknowledgements

Upon reaching the culmination of my Ph.D. journey, I would like to convey my utmost gratitude and profound appreciation to all people who have lent their support. This thesis would not come to fruition without their unwavering assistance, invaluable guidance, and ceaseless encouragement.

I stand profoundly indebted to my advisor, Prof. Tianwei Zhang, whose influence on my academic journey is immeasurable. As I glance back at the outset of my doctoral study, I consider myself incredibly fortunate to have transitioned from another advisor at NTU to Tianwei, becoming part of his inaugural group of Ph.D. students. Although I had scarce research experience at that juncture, Tianwei consistently mentored me with patience and kindness that I forever cherish. He steered me towards a promising research area, that of deep learning job scheduling, urging me to begin with fundamental characterization studies and to delve into its implications at my own pace. This approach eased my foray into research, allowing me to navigate the complexities of datacenter scheduling and establish a solid foundation in the field. When it came to crafting my first research manuscript Helios, Tianwei's rigorous and meticulous comments were invaluable. He carefully reviewed each sentence, table, and figure, providing detailed feedback, thereby refining my work. Our collective efforts earned the unanimous endorsement from the reviewers, instilling immense confidence in my subsequent research. Besides, Tianwei has afforded me invaluable opportunities to collaborate with industry professionals and has generously sponsored my travels to conferences where I am able to present my work. Attempting to articulate my gratitude to Tianwei is an uphill task as words simply fall short. The experience in his group would undoubtedly mold my future in multifaceted ways and I aspire to be a professor like him in my own future career.

Moreover, I am privileged to be under the mentorship of my co-advisor, Prof. Yonggang Wen. Drawing upon his vast academic reservoir of knowledge and experience, he bestows upon me invaluable guidance which greatly aids in shaping my research direction, polishing my presentation skills, and strategizing my academic career path. Not stopping at that, he graciously avails me of myriad opportunities to interact with

a diverse cadre of professionals from other domains, which not only augments the richness of my research but also expands my intellectual horizons. The indelible imprint he has left on my Ph.D. journey will undeniably continue to illuminate my future pursuits. I harbor deep pride and gratitude for the invaluable experience gained during my time under his mentorship.

I would like to extend my sincere gratitude to my industrial mentor, Dr. Peng Sun. Despite the constraints imposed by the COVID-19 pandemic necessitating our discussions to perform predominantly online, the insights I have acquired from him have been profoundly illuminating. His hands-on experience at SenseTime and Shanghai AI Laboratory has provided me with a deep understanding of the real-world challenges and issues within the machine learning systems domain. The lessons I learned under his guidance significantly influenced my work on both Primo and Lucid. Additionally, his assistance in securing necessary resources from the company for research purposes was invaluable, for which I am immensely thankful.

In addition to my advisors, I am extremely grateful to the rest of my thesis advisory committee: Prof. Xueyan Tang and Prof. Lee Bu Sung. Their insightful feedback and remarks significantly elevated the quality of this thesis.

The research works in this thesis would not have been possible without the synergies of talented people in our group. I was fortunate to have worked alongside some truly remarkable co-authors: Wei Gao, Zhisheng Ye, Meng Zhang, Qiaoling Chen, Dr. Haozhao Wang and Haochong Lan. I genuinely cherish every invigorating discussion and creative brainstorming we engaged in. Moreover, the nights spent striving to meet conference deadlines hold a special place in my memory.

I joined the S-Lab in its infancy and became the third research staff member. Despite its swift expansion over the years, growing to include over a hundred researchers, the lab has well preserved its vibrant and friendly atmosphere, fostering an environment that encourages students to freely explore. I have been fortunate to work with Guanlin Li and Dr. Yuan Xu from our group, as well as Prof. Chen Change Loy, Prof. Shuai Yang, Prof. Yixin Cao, Dr. Liming Jiang, Dr. Wenwei Zhang, Dr. Yuhang Zang, Dr. Guangcong Wang, Dr. Wei Li, Dr. Shoukang Hu, Dr. Man Zhou, Yuming Jiang, Yushi Lan, Jingkang Yang, Chong Zhou and other members from MMLab (affiliated with S-Lab). The discussion with people in different fields has greatly broadened my horizons and inspired me to think outside of the box. Additionally, I am grateful to supportive administrative staffs of S-Lab: Dr. Yuen Fei Wong, Dr. Geong Sen Poh, Wei Tang, Chee Guan Koh and Eunice Yeo, for their assistance in various matters, such as travel reimbursement, server purchase and maintenance.

I would also like to convey my thanks to members in the Cyber Security Lab (CSL) and Cloud Application and Platform (CAP) group of NTU: Prof. Shangwei Guo, Dr. Guowen Xu, Dr. Wenhao Fu, Dr. Jianfei Sun, Dr. Hao Ren, Dr. Hangcheng Liu, Dr. Jie Zhang, Dr. Hangcheng Liu, Dr. Jie Zhang, Dr. Jianfei Sun, Dr. Shudong Zhang, Dr. Wenbo Jiang, Dr. Huaizheng Zhang, Dr. Weiming Zhuang, Ke Jiang, Xingshuo Han, Kangjie Chen, Xiaoxuan Lou, Gelei Deng, Terence Wen Zheng Ng, Hui En Pang, Kangqiao Zhao, Xiaobei Yan, Yutong Wu, Meng Hao, Hanxiao Chen, Yi Xie, Zhaoxuan Wang, Jianda Chen, Shiqian Zhao, Ruihang Wang, Zhiwei Cao, Erdong Chen, Jiabei Liu, Bo Zhang, Meng Sheng, Jie Li, Minghao Li, Tianshu Liu.

I had the extraordinary opportunity to intern at Shanghai AI Laboratory, an experience that was nothing short of exhilarating. My heartfelt thanks go to the remarkable team: Guoteng Wang, Zerui Wang, Diandian Gu, Yingtong Xiong, Yang Gao, Xun Chen, Ting Huang, Penglong Jiao, Jiaxing Li, Wengweng Qu, Teng Wan, Zhihao Xu, Lei Zhang, Xingxue Zhang, Dr. Qizhen Weng, and Li Ma. Their support and assistance played an invaluable role in our research work.

I firmly believe that the most impactful research emerges when individuals from diverse fields collaborate. I have been fortunate to engage in discussions or collaborations with some truly amazing talents with various research backgrounds, including Harsha Nori and Greg Yang from Microsoft, Richard Liaw and Antoni Baum from Anyscale, Shang Wang and Xin Li from UofT, Ziji Shi, Shenggan Cheng, and Shenggui Li from NUS. Their insights have been instrumental, significantly enriching the depth and breadth of my research projects.

I am profoundly grateful to my friends, both within the NTU and beyond, for their continued presence and support throughout the years. They have imbued my life with joy, offering a tranquil refuge from the unyielding pursuit of work.

Last, but definitely the most: my deepest gratitude goes to my family, including my parents, sister and my beloved girlfriend. Despite the geographical distance, I have always felt their steadfast support, relentless encouragement and profound love in every step of my Ph.D. journey. It is not hyperbolic to say that this thesis would not be possible without them. My parents recognized and fostered my nascent passion in science and engineering. When it was time to determine my career, they gently steered me towards the realm of research, consistently motivating me to relentlessly pursue my utmost potential. I dedicate this thesis to them.

To my dear family.

Abstract

With the widespread adoption of deep learning (DL) applications in recent years, training DL models has become increasingly prevalent. Nevertheless, training these models is typically time-consuming and computation-intensive, relying heavily on expensive heterogeneous infrastructure. To facilitate model development, numerous research institutes, technology companies and cloud providers have invested substantially in establishing their large-scale GPU clusters. Regrettably, these clusters are frequently underutilized for various reasons, primarily due to (1) *inefficiency*: agnostic to the unique features of DL training jobs; (2) *impracticality*: arduous to deploy preemptive scheduling mechanisms in practice. This thesis presents a suite of techniques to tackle the challenges associated with resource management and job scheduling at the datacenter level. Moreover, we expand our research to encompass broader scenarios of machine learning systems, aiming to develop highly efficient and practical systems in real-world applications.

In the first part of this thesis, we specialize in developing tailored systems to enhance the efficiency of deep learning job execution in datacenters. To accomplish this objective, a thorough grasp of job features and user behaviors is essential. Unfortunately, prior research has only offered limited analysis of DL workloads. Therefore, we perform a comprehensive investigation into the characteristics of DL jobs and resource management within a SenseTime datacenter, containing over 6,000 GPUs. We uncover some interesting conclusions from the perspectives of clusters, jobs and users, which inspire us to manage resources based on historical data to minimize job queuing and energy consumption. Furthermore, aside from optimizing the scheduling of general training jobs, we observe a widespread occurrence of hyperparameter tuning jobs, which consume substantial resources within GPU clusters. Consequently, we build a holistic system that automatically applies the novel hyperparameter transfer theory together with multiple system techniques to jointly improve the tuning efficiency. At the job level, the system automatically scales models and optimizes tuning efficiency through inter-trial fusion, which combines multiple smaller models into a unified entity. At the cluster level, it interacts with the scheduler to dynamically allocate resources and execute trials. Specifically, it extends tuning resources

by interleaving training with pipeline-enabled large model training tasks, effectively utilizing idle time intervals on each node, referred to as bubbles. Our experiments on the GPT-3 XL model demonstrate a significant acceleration in the hyperparameter tuning process, achieving a makespan reduction of 78.5×.

In the second part of this thesis, we explore and address the challenges that hinder the practical deployment of research prototypes for datacenter schedulers and other machine learning systems. Although recent research works on DL-tailored schedulers have showcased their impressive ability to enhance job efficiency, deploying them in practice poses significant challenges due to several substantial defects, including inflexible intrusive manner, exorbitant integration cost and limited scalability. To bridge these gaps, we design a non-intrusive and transparent scheduler that can provide better performance than preemptive and intrusive schedulers. It utilizes a predication-based packing strategy to circumvent interference and orchestrate resources based on estimated job priority values to achieve efficient scheduling. Additionally, in more general system-related research domains, machine learning techniques have been widely adopted to enhance system performance. However, we identify similar gaps in practical deployment, such as opaque decision processes, poor generalization and robustness, as well as exorbitant training and inference overhead. We develop a unified framework to resolve the above challenges and facilitate transparent, accurate and lightweight system with interpretable models. It optimizes both the training and post-processing stages involved in constructing learning-augmented systems. Through evaluations conducted on cutting-edge systems in storage and networking domains, our framework can provide clear model interpretations, lower deployment costs and better system performance.

Contents

List of Publications

[1] Qinghao Hu, Zhisheng Ye, Meng Zhang, Qiaoling Chen, Peng Sun, Yonggang Wen, and Tianwei Zhang. **Hydro: Surrogate-Based Hyperparameter Tuning Service in Datacenters**. In *USENIX Symposium on Operating Systems Design and Implementation*, OSDI '23, 2023.

[2] Qinghao Hu*, Meng Zhang*, Peng Sun, Yonggang Wen, and Tianwei Zhang. **Lucid: A Non-Intrusive, Scalable and Interpretable Scheduler for Deep Learning Training Jobs**. In *Architectural Support for Programming Languages and Operating Systems*, ASPLOS '23, 2023.

[3] Qinghao Hu, Harsha Nori, Peng Sun, Yonggang Wen, and Tianwei Zhang. **Primo: Practical Learning-Augmented Systems with Interpretable Models**. In *USENIX Annual Technical Conference*, USENIX ATC '22, 2022.

[4] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. **Characterization and Prediction of Deep Learning Workloads in Large-Scale GPU Datacenters**. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, 2021.

[5] Wei Gao*, Qinghao Hu*, Zhisheng Ye*, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. **Deep Learning Workload Scheduling in GPU Datacenters: Taxonomy, Challenges and Vision**. arXiv '22, 2022.

[6] Meng Zhang, Qinghao Hu, Peng Sun, Yonggang Wen, and Tianwei Zhang. **Boosting Distributed Full-graph GNN Training with Asynchronous One-bit Communication**. arXiv '23, 2023.

* Equal Contribution.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Emerging machine learning (ML) technologies have empowered a lot of new applications to change our life. Examples of these groundbreaking applications include ChatGPT [11], which revolutionizes conversational interactions, AI painting with Stable Diffusion [12], which produces astonishing artwork, and reinforcement learning [13] for self-driving vehicles. These remarkable advancements heavily rely on advanced system support, such as deep learning frameworks like Tensorflow [14] and PyTorch [15] for efficient model training and inference, as well as schedulers like Kubernetes [16] and Slurm [17] to effectively manage user workloads. However, the rapid growth in the quantity of ML workloads and the emergence of large-scale models with billions of parameters pose significant challenges in designing ML-tailored systems to achieve these demands.

On the other hand, ML techniques are also being adopted to optimize systems across diverse fields, such as storage, network management, security, and cluster scheduling. By leveraging ML's capabilities, these learning-augmented systems aim to enhance performance, efficiency, and adaptability. However, the practical deployment of these systems presents various challenges. For instance, the training cost for ML models can be exorbitant, hindering their widespread adoption. Additionally, the inference overhead, or the computational resources required for real-time ML processing, can be substantial. Furthermore, generalization issues arise when ML models trained on specific datasets struggle to perform well on unseen data, limiting their effectiveness in real-world scenarios.

Consequently, the pursuit of ***efficiency*** and ***practicality*** becomes a crucial objective when constructing ML systems. In this thesis, we aim to tackle these challenges through the synergy of machine learning and system techniques, thereby fostering

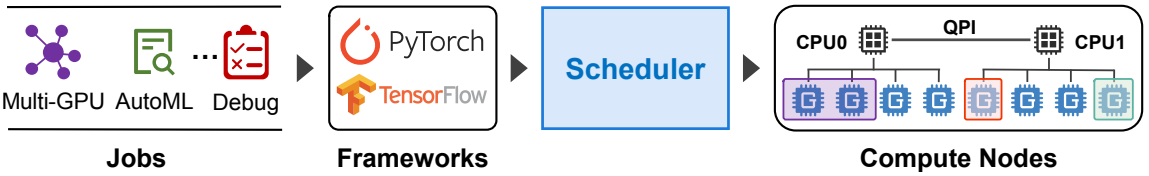


FIGURE 1.1: The workflow of DL training job in a datacenter.

the advancement of ML applications and learning-augmented systems. We not only focus on the domain of datacenter workload scheduling, encompassing both inter- and intra-job scheduling optimization, but also delve into broader optimization possibilities across various system scenarios, including storage, networking, and energy conservation.

1.2 Preliminaries

In this section, we provide an overview of the foundational concepts and background knowledge that form the basis for understanding the subsequent chapters in this thesis. These preliminaries serve to familiarize the reader with the key terminology and methodologies relevant to my research.

1.2.1 Deep Learning Training

A DL model acquires its parameters (i.e., weights) through an *iterative* process [18, 19]. In each iteration, it operates on a batch of labeled data to update the model weights using gradient descent. The entire training process typically involves multiple iterations with mini-batches and can span from hours to days. This process can be interrupted and resumed using checkpoints [20, 21]. DL jobs typically demand some GPUs as the primary resource, along with associated resources such as CPUs and memory. Besides, hardware energy consumption is also a significant factor needs to be considered. Thus, the cost of training a DL model can be exorbitant, especially for large-scale models with billions of parameters [22].

1.2.2 Scheduling Training Jobs in Datacenters

In order to amortize infrastructure cost, it is a common practice for tech companies and research institutes to build multi-tenant DL datacenters to facilitate DL model development. These datacenters are often divided into multiple Virtual Clusters (VCs) dedicated to different product groups within the company [2, 23, 24]. Users submit

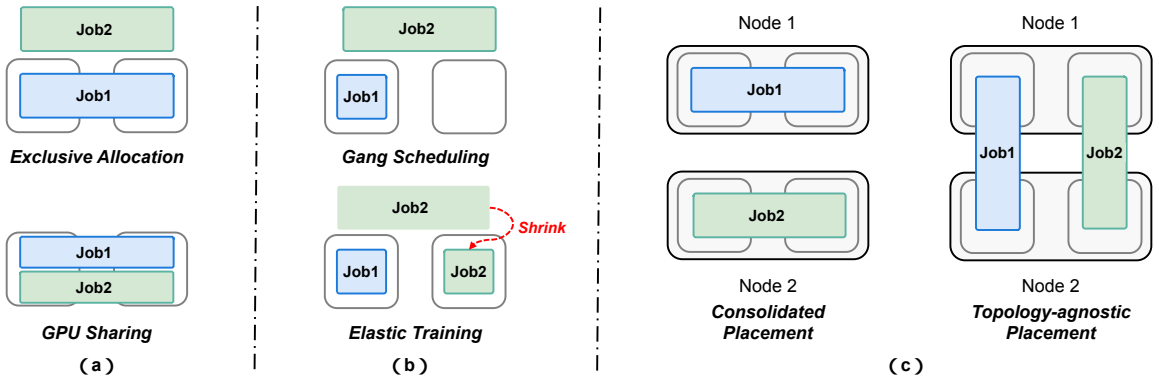


FIGURE 1.2: Illustration of DL training job scheduling mechanisms. (a) Exclusive Allocation versus GPU Sharing. (b) Gang Scheduling versus Elastic Training. (c) Consolidated Placement versus Topology-agnostic Placement.

DL training jobs to the cluster, specifying relevant configurations such as GPU and CPU requirements, as well as job names.

As shown in Figure 1.1, we depict the end-to-end workflow of a DL training job in a datacenter, showcasing how cluster schedulers allocate resources to these jobs. The workflow begins with the submission of a training job to the cluster. This entails supplying the requisite input data, specifying the preferred DL model and algorithms, and configuring any additional job parameters. These jobs can serve various purposes, such as training a complex model utilizing multiple GPUs, fine-tuning an existing model, or conducting hyperparameter tuning. The execution of these jobs relies on deep learning frameworks, such as PyTorch and TensorFlow. Subsequently, the cluster scheduler allocates computing resources to the job, determining the order of job allocation and selecting suitable server(s) for efficient execution of the training task.

Scheduling Mechanisms. Conventional schedulers often struggle to fully exploit cluster resources because they lack awareness of the advanced mechanisms inherent in DL training jobs. For instance, as shown in Figure 1.2 (a), exclusive allocation signifies that a DL job possesses exclusive ownership of the allocated resources. In contrast, GPU sharing allows multiple jobs to coexist on the same GPU device, utilizing resources in a time- or space-sharing manner. Unlike CPUs, GPUs lack built-in hardware-level support for fine-grained sharing among users, resulting in exclusive allocation to DL training jobs. To tackle this challenge, datacenters employ various technologies such as NVIDIA Multi-Instance GPU (MIG) [25], Multi-Process Service (MPS) [26], and GPU virtualization to enable GPU sharing.

Figure 1.2 (b) presents two scheduling mechanisms employed for data-parallel DL jobs. Firstly, gang scheduling entails allocating all GPUs simultaneously in an all-or-nothing fashion for DL training [27]. This requirement arises from the inherent characteristics of DL frameworks. On the other hand, elastic training eliminates the rigid GPU request constraint and allows for a dynamic number of GPUs to be utilized

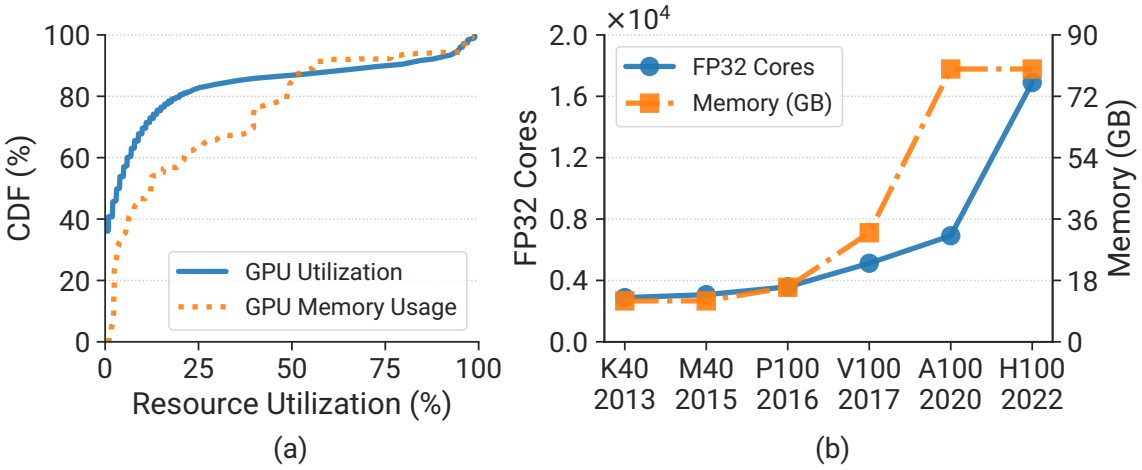


FIGURE 1.3: Background. (a) GPU utilization distribution in an Alibaba cluster [1]. (b) Exponential growth of NVIDIA datacenter GPU capability. x-axis: GPU name & release year.

during training jobs. Many scheduling systems incorporate elastic training to enhance GPU utilization and expedite the training process.

Distributed DL jobs are highly influenced by the locality of allocated GPU resources. Figure 1.2 (c) depicts two types of placement: consolidated placement and topology-agnostic placement. Consolidated placement efficiently reduces communication overhead compared to topology-agnostic placement since the intra-server bandwidth (i.e., PCIe or NVLink [28]) is much faster inter-server bandwidth (i.e., Ethernet or InfiniBand [29]). Besides, the communication sensitivity of training jobs is determined by the infrastructure. Advanced interconnect links, such as InfiniBand [29], provide higher bandwidth, which helps alleviate communication overhead.

1.2.3 Characteristics of DL Clusters

Low GPU Utilization. Recent works [1, 18, 30, 31] show a common phenomenon that most GPUs are underutilized in DL clusters. Figure 1.3 (a) shows the Cumulative Distribution Function (CDF) of one-week GPU usage statistics collected from an Alibaba datacenter [1]. The GPU memory consumption is normalized by the memory capacity of the GPU. It is evident that only 16% of the GPUs achieve higher than 50% GPU utilization. Additionally, with the rapid evolution of GPU computing capability as shown in Figure 1.3 (b), future GPUs can deal with more complex and larger-scale DL training jobs. However, they also become more prone to be underutilized for most small-scale or mid-scale jobs.

High-skewed Workload Distribution. Real-world production DL clusters [2, 23, 24] present similar workload distributions: (1) *Small-scale*. Over 95% jobs are single-node jobs (within 4/8 GPUs) in Microsoft [24] and SenseTime (Chapter 2). (2)

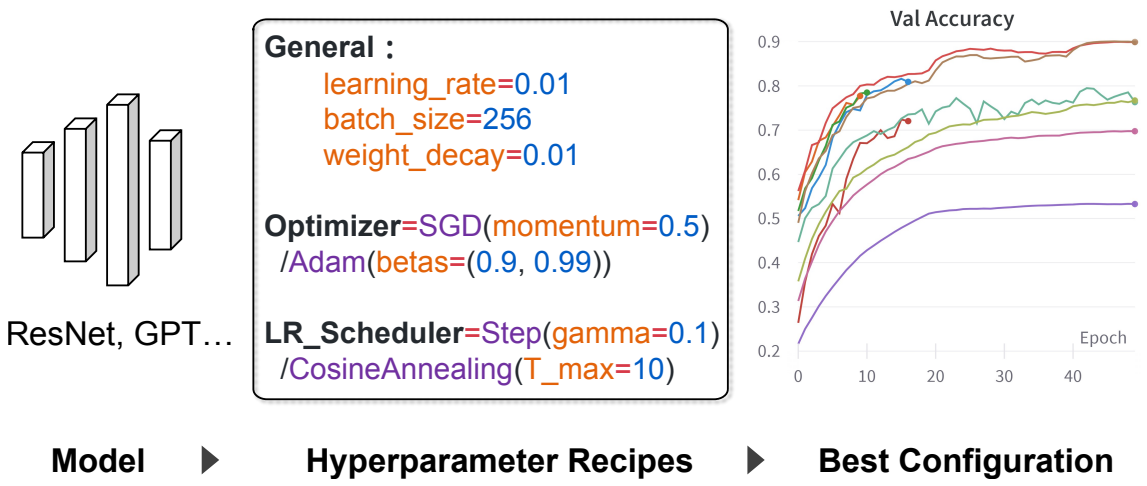


FIGURE 1.4: Illustration of hyperparameter tuning

Recurring. Most jobs (∼90%) are recurring hyperparameter searching jobs [2, 32]. (3) *Debugging.* The majority of jobs are short-term for debugging purposes, where nearly 70% of resources in Microsoft are occupied by failed or canceled jobs. Users desire to obtain debugging job feedback timely. However, the diversity of workloads is often ignored by existing works and it lacks specific design for debugging jobs.

1.2.4 Hyperparameter Tuning

Hyperparameter tuning (i.e., Hyperparameter Optimization, HPO) is a crucial step in the development of deep learning models as it improves model performance at the expense of significant resources. Acquiring a qualified deep learning model is challenging due to its high sensitivity to hyperparameters, which control the training process and must be set before training begins. Poorly chosen hyperparameters can lead to training instability and inferior model quality, while well-tuned hyperparameters can greatly enhance model performance. Figure 1.4 illustrates the process of hyperparameter tuning. To train a qualified model, users typically experiment with multiple configurations and use feedback from these jobs to determine which configuration to keep or discard. This process leverages the feedback-driven exploration feature of deep learning workloads.

In the general workflow of an HPO job, the user defines a search space of hyperparameters to explore. The tuning algorithm then generates a set of training trials, with each trial containing a unique hyperparameter configuration sampled from the search space. The HPO system coordinates the execution of these trials until the best hyperparameter configuration is discovered.

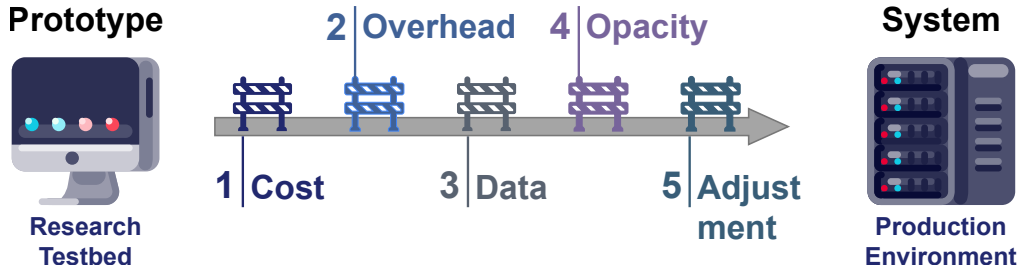


FIGURE 1.5: Summary of 5 challenges in learning-augmented system deployment.

1.2.5 System Practicality

Beyond efficiency, easy to deploy systems into practice is significant. However, most existing work targets excellent system performance while ignoring its complexity and usability. As a result, it is typically nontrivial to shift a research prototype into a production-level system in practice.

For example, although machine learning has demonstrated remarkable performance in various computer systems, some substantial flaws (shown in Figure 1.5) can prohibit its deployment in practice, including opaque decision processes, poor generalization and robustness, as well as exorbitant training and inference overhead. Specifically, the challenges are as follows: 1. *High training and tuning cost*, as reported by Microsoft [33], the operating cost often exceeds enterprise expectations due to frequent model fine-tuning or retraining to maintain the system performance; 2. *Exorbitant inference overhead*, which may cause some side-effect to the main production workloads, because ML models occupy too many resources; 3. *Susceptible to the quantity and quality of data*, where insufficient data may lead to inferior performance than heuristic algorithms; 4. *Opaque decision-making process*, where prediction processes are unintelligible to humans and operators have insufficient confidence to deploy the systems; 5. *Difficulty in troubleshooting and adjustment*, where improper modifications may cause severe performance degradation [34].

In Chapter 4 and Chapter 5, I will provide a comprehensive analysis of the practicality concerns in scheduling systems and other related systems. Furthermore, I will present effective solutions to tackle these issues.

1.3 Summary of Contributions and Results

> **Thesis Statement**: *The synergy between system software stack (e.g., cluster scheduler, execution framework) and machine learning technique (e.g., algorithm, theory) can significantly enhance the efficiency of model training in practice.*

The synergy of ML and system is bringing many exciting new opportunities to both fields, and is shaping future's promising applications. It is believed that this is the golden era for interdisciplinary research of machine learning and systems. This thesis presents a suite of techniques to tackle the challenges associated with resource management and job scheduling at the datacenter level. Moreover, we expand our research to encompass broader scenarios of machine learning systems, aiming to develop highly efficient and practical systems in real-world applications.

In the first part of this thesis, we specialize in developing tailored systems to enhance the efficiency of deep learning job execution in datacenters.

Helios [23]: Prediction-based Large-Scale Cluster Management. When operating a deep learning datacenter, optimization of resource scheduling and management can bring significant financial benefits. Achieving this goal requires a deep understanding of the job features and user behaviors. Based on the comprehensive study of the characteristics of SenseTime DL workloads, I introduce a general-purpose framework to optimize cluster resource management based on historical data. It adopts the "plug-and-play" fashion, where different resource management services can be integrated into this framework. A scheduling service and an energy saving service are implemented as case studies. Helios not only improves scheduling efficiency by up to 6.5× but also conserves over 1.65 million kWh of electricity.

Hydro [35]: Surrogate-based Hyperparameter Tuning Service. Hyperparameter tuning is an essential step in deep learning model development that provides better model performance at the cost of substantial resources. While existing systems can improve tuning efficiency, they still fail to handle large models with billions of parameters and efficiently leverage cluster resources. Motivated by these deficiencies, we introduce Hydro, a surrogate-based hyperparameter tuning service that optimizes tuning workloads in both the job-level and cluster- level granularities. Our experiments on six models show that Hydro Tuner can dramatically reduce tuning makespan by up to 78.5× and no reduction in tuning quality or sacrificing the training throughput of large models.

In the second part of this thesis, we explore and address the challenges that hinder the practical deployment of research prototypes for datacenter schedulers and other machine learning systems.

Lucid [36]: Non-Intrusive Deep Learning Job Scheduling. Based on the aforementioned scheduler study, I find recent deep learning workload schedulers are arduous to deploy in practice due to some substantial defects, including inflexible intrusive manner, exorbitant integration and maintenance cost. Motivated by these issues, I design and implement Lucid, a non-intrusive deep learning workload scheduler based

on interpretable models. It contains a two-dimensional optimized profiler and indolent packing strategy. Lucid orchestrates resources based on estimated job priority values and sharing scores to achieve efficient scheduling. Our extensive experiments demonstrate Lucid significantly improves the average job completion time (JCT) by 5.2∼8.2×.

Primo [37]: Practical Learning-Augmented Systems. Some substantial flaws prohibit ML-based systems deployment in practice, including opaque decision processes, poor generalization and robustness, as well as exorbitant training and inference overhead. To this end, I propose Primo, a unified framework for developers to design and provide inherent interpretability for practical learning-augmented systems. Primo provides not only precise and comprehensive interpretations for developers to understand and adjust models, but also better prediction accuracy and smaller overhead. To extensively evaluate Primo in real scenarios, we apply Primo to three state-of-the-art (SOTA) learning-augmented systems, including two online systems (flash storage and video streaming) and an offline system (SmartNIC offloading). Compared with previous work, Primo can provide up to 2.8× system performance improvement, 79× inference latency reduction and 100× training cost conservation.

1.4 Roadmap

This thesis is organized into two parts.

Part I: *Building Efficient Systems: Optimization on Training and Tuning Workloads in Datacenters*. It focuses on ML workload scheduling system research at both the cluster-level and job-level to enhance system efficiency.

- Chapter 2 introduces a general-purpose framework for optimizing cluster resource management in DL datacenters, improving scheduling efficiency and conserving electricity.
- Chapter 3 proposes a surrogate-based hyperparameter tuning service that dramatically reduces tuning makespan and maintains tuning quality for large models.

Part II: *Building Practical Systems: Optimization on Machine Learning Systems in Various Domains*. It analyses and overcomes the obstacles that impede the real-world implementation of datacenter schedulers and other machine learning systems.

- Chapter 4 describes a non-intrusive deep learning workload scheduler based on interpretable models that can outperform preemptive schedulers.

- Chapter 5 presents a unified framework for designing practical learning-augmented systems with inherent interpretability, providing improved system performance, inference latency reduction, and training cost conservation.

Last, the conclusion (Chapter 6) summarizes the thesis and discusses future work.

Part I

Building Efficient Systems: Optimization on Training and Tuning Workloads in Datacenters

Chapter 2

Helios: Understanding and Optimizing DL Workload Scheduling

2.1 Introduction

Over the years, we have witnessed the remarkable impact of Deep Learning (DL) technology and applications on every aspect of our daily life, e.g., face recognition [38], language translation [39], advertisement recommendation [40], etc. The outstanding performance of DL models comes from the complex neural network structures and may contain trillions of parameters [41]. Training a production model may require large amounts of GPU resources to support thousands of petaflops operations [22]. Hence, it is a common practice for research institutes, AI companies and cloud providers to build large-scale GPU clusters to facilitate DL model development. These clusters are managed in a multi-tenancy fashion, offering services to different groups and users based on their demands, with resource regulation and access controls.

A job scheduler is necessary to manage resources and schedule jobs. It determines the resource utilization of the entire cluster, and job performance, which further affects the operation cost and user experience. Understanding the characteristics of DL workloads is indispensable for managing and operating GPU clusters. DL jobs share some similar features as conventional HPC workloads, which are generally different from big data jobs in cloud computing (e.g., MapReduce). (1) *Iterative process* [42, 43]. Similar to some numerical simulation and analysis jobs in HPC, typical DL training jobs are also iterative computation. The target model is obtained through the gradient descent update iteratively and the long-term training process can be suspended and resumed via checkpoints. (2) *Gang scheduling* [27]. DL training jobs require all the GPUs to be allocated simultaneously in an all-or-nothing manner [1, 44]. This may

cause resource fragmentation in GPU clusters. (3) *Exclusive allocation* [24, 45]. The GPU resources are allocated exclusively in DL datacenters. Although the advanced NVIDIA Multi-Instance GPU (MIG) technology [25] provides intrinsic hardware-level support for fine-grained sharing on NVIDIA A100 GPUs, existing GPU datacenters built with previous-generation GPUs typically only support coarse-grained GPU allocation [31].

Furthermore, DL training jobs also exhibit some unique features different from most HPC or big data workloads. (1) *Inherent heterogeneity* [18]. DL training jobs typically require a number of GPUs as the dominant resources, as well as associated resources such as CPUs and memory. (2) *Placement sensitivity* [42]. Multi-GPU jobs usually perform gradient synchronization to update model parameters at the end of each iteration. Better interconnect topology can achieve lower communication overhead and faster training speed for multi-GPU jobs. Meanwhile, colocating multiple jobs in one server could cause performance degradation due to the interference of system resources like PCIe bandwidth [24]. (3) *Feedback-driven exploration* [18]. To train a model, users typically try several configurations, and use early feedback from these jobs to decide which one should be kept or killed. This gives higher cancellation rates in GPU clusters [24].

A variety of works proposed new schedulers specifically for GPU clusters to achieve higher performance, resource utilization and fairness [1, 42, 45–47]. To adapt to the characteristics of DL workloads, these works adopt the Philly trace [24] from Microsoft GPU cluster for design and evaluation. To the best of our knowledge, the Philly trace is the only publicly available DL workload dataset so far. However, analysis of only this trace may be inadequate to prove the generality of the designs and can cause the overfitting issue, as pointed in [48, 49]. In addition, this trace was collected in 2017. Due to the rapid development of DL technology and demands, this trace may not be able to reflect the latest characteristics of DL jobs anymore.

We present an in-depth study about the characterization of DL workloads and scheduler designs. Our study is based on a new set of DL job traces collected from a datacenter in SenseTime, named Helios. These traces have the following benefits. (1) They were collected over six months in 2020, which can represent the emerging DL algorithms and models. (2) They incorporate more than 3 million jobs among four independent GPU clusters. These jobs exhibit high variety, covering different types (training, inference, data preprocessing, etc.), applications (computer vision, natural language processing, etc.), and purposes (product development, research etc.). (3) These traces include rich information, which enables thorough analysis from different aspects, and diverse evaluations of scheduling systems. It is worth noting that we do not aim to compare our traces with Philly and show the advantages. Instead, we release our traces and expect they can be an alternative to Philly. Our traces together

with Philly can increase the diversity of job workloads to enable more general design and evaluations (as we will do for our prediction framework in this work). We also hope this can inspire other organizations to release their job traces to benefit the community of deep learning systems.

Based on the traces, this work makes two key contributions. First, we perform a large-scale analysis of the DL jobs in the four clusters, from the perspectives of jobs, clusters and users. Prior works [24, 50] only focused on the job-level characterization. Our study provides more extensive analysis about the behaviors of GPU clusters and users. Through our analysis, we identify seven new implications, which can shed light on the design of GPU clusters.

Second, we introduce a novel prediction-based framework to manage compute resources and schedule DL jobs. This framework is inspired by some implications from our trace analysis: the characteristics of clusters and jobs exhibit obvious predictable patterns, which gives us the opportunities to forecast those behaviors in advance and then optimize the management of jobs and resources. This framework can be integrated with different services. Each service builds a machine learning model from the historical data, and predicts the future states of the cluster, or upcoming job information. Based on the prediction results, the service can make optimized actions for resource and job management. To maintain prominent performance, the prediction model is also kept updated with new data. We present two services as case studies: (1) A *Quasi-Shortest-Service-First* scheduling service can assign each new job a priority score based on the historical information. It then schedules the jobs based on these priority scores, which can reduce the queuing delay by up to 20.2× and improve the overall job completion time (JCT) by up to 6.5×. (2) A *Cluster Energy Saving* service can proactively predict the demanded compute nodes in advance, and then leverages the Dynamic Resource Sleep (DRS) technique to efficiently power off the unnecessary nodes. It can improve up to 13% of node utilization rate and conserve millions of kilowatt hours of electricity annually across the four clusters.

2.2 Background

In this section, we first introduce our GPU datacenter, dubbed Helios (§2.2.1). Then we describe the DL workloads running in this datacenter (§2.2.2) and the corresponding job traces (§2.2.3).

TABLE 2.1: Configurations of four clusters in Helios (All data are collected on September 1st, 2020, except # of jobs and VCs, which cover the period of April-September, 2020).

	Venus	**Earth**	**Saturn**	**Uranus**	**Total**
CPU	Intel, 48 threads/node		Intel, 64 threads/node		-
RAM	376GB per node		256GB per node		-
Network	IB EDR		IB FDR		-
GPU model	Volta	Volta	Pascal & Volta	Pascal	-
# of VCs	27	25	28	25	105
# of Nodes	133	143	262	264	802
# of GPUs	1,064	1,144	2,096	2,112	6,416
# of Jobs	247k	873k	1,753k	490k	3,363k

2.2.1 Helios Datacenter

Helios is a private datacenter dedicated to developing DL models for research and production in SenseTime, containing multiple multi-tenant clusters. In this chapter, we select 4 representative clusters: `Venus`, `Earth`, `Saturn`, and `Uranus`. Table 2.1 shows the configurations of each cluster. Note that `Saturn` is a heterogeneous cluster with mixed NVIDIA Pascal and Volta GPUs, while the other three clusters are composed of identical Volta or Pascal GPUs. Our conclusions from the analysis of these 4 clusters are general for other clusters in Helios as well.

Each cluster in Helios serves multiple groups in SenseTime concurrently. To support resource isolation and management for multi-tenancy, a cluster is further divided into several Virtual Clusters (VCs), and each VC is dedicated to one group with its demanded resources. All GPUs within one VC are homogeneous. Each node is exclusively allocated to one VC, and over-subscription of GPU resource quota is not enabled in Helios. Configuration of a VC can be dynamically changed in three situations: (1) when the demand of a group is increased, new servers will be purchased and allocated to its VC, or form a new VC; (2) when a group is less busy, the size of its VC may be scaled down; (3) when groups are combined or split, their VCs are also merged or split correspondingly.

To reduce the communication overhead in distributed workloads, GPUs are interconnected to each other via a hierarchic network: (1) intra-node communication is achieved via the high-bandwidth and low-latency PCIe (for Pascal GPUs) or NVLink (for Volta GPUs) [28] ; (2) inter-node communication within the same Remote Direct Memory Access (RDMA) domain is achieved via the high-speed InfiniBand. To improve the workload performance and reduce network interference, cross-RDMA-domain distributed training (communication through TCP/IP) are not allowed in Helios.

The distributed storage system is also critical for workload performance. To support the massive data throughput in DL jobs, Helios adopts Lustre [51] as the file system, and the input data for the jobs are stored in Ceph [52] as the object storage. In addition, Memcached [53] is used to accelerate data access.

The *Slurm* workload manager [17] is adopted to regulate the resources and job execution. Specifically, it dynamically adjusts the configurations of VCs, including the resource quota in each VC, total number of VCs, job time limit, etc. Meanwhile, it is also responsible for the online scheduling of DL jobs, following three steps. (1) A user submits his or her job to a VC with specified job resource demands (e.g., numbers of GPUs and CPUs). If the CPU requirement is not specified, the scheduler will allocate CPU cores proportional to the requested GPU counts. (2) *Slurm* maintains a separate allocation queue for each VC (*VCQueue*), and selects jobs for scheduling. All jobs in one *VCQueue* have the same priority setting so the scheduling order is only determined by the job's submission time. (3) *Slurm* allocates jobs in a consolidated paradigm by packing jobs into as few nodes as possible. A user can also select specific nodes if he or she has special topology requirements. For instance, some exploratory jobs are placed across specific numbers of nodes for testing the performance impact of GPU affinity. Only 0.15% of the jobs are placed in a way customized by the users. After the job is scheduled, it keeps running until completion or being terminated by the user. Preemption is not supported in Helios.

2.2.2 Workloads in Helios

Helios supports various types of jobs in the DL development pipeline, e.g., data preprocessing, model training, inference, quantization, etc. These workloads are submitted by product groups for developing commercial products, as well as research groups for exploring new technologies. They range over different DL domains, including computer vision, natural language processing, reinforcement learning, etc.

A majority of the GPU jobs are DL training, which mainly follow an iterative fashion [45]: the training task consists of many iterations, where gradient descent is applied to update model parameters based on the mini-batch data in each iteration. To scale with complex models and large datasets, many jobs adopt the distributed data-parallel training scheme across multiple GPUs. In each iteration, every GPU processes a subset of data in parallel and then performs gradient synchronization to update model parameters. This synchronization typically adopts the parameter sever [54] or all-reduce strategy for high-speed multi-node/multi-GPU communication (e.g., NCCL [55] as backend). Users mainly adopt the built-in libraries (e.g., DistributedDataParallel in Pytorch, MultiWorkerMirroredStrategy in Tensorflow) to implement their jobs.

TABLE 2.2: Comparisons between Helios and Philly traces.

	Helios	**Philly**		**Helios**	**Philly**
# of clusters	4	1	Duration	6 months	83 days
# of VCs	105	14	Average # of GPUs	3.72	1.75
# of Jobs	3.36M	103k	Average Duration	6,652s	28,329s
# of GPU Jobs	1.58M	103k	Maximum # of GPUs	2,048	128
# of CPU Jobs	1.78M	0	Maximum Duration	50 days	60 days

In addition, there are also a quantity of jobs for data/model preprocessing and post-processing. For instance, some CPU jobs generate large-scale training datasets by extracting frames from videos; some jobs rescale the images according to the model's requirements. To speed up model inference, it is common to perform post-training quantization to reduce model size before deployment.

2.2.3 DL Job Traces from Helios

We collect jobs from each of the 4 clusters in Helios, which serves as the basis of our analysis and system design in this work. These four traces span 6 months from April 2020 to September 2020, covering a total of 3.36 million jobs over 802 compute nodes with 6416 GPUs. Each trace contains two parts: (1) We collect the job logs through the *Slurm* `sacct` command, which provides the rich information for each job. (2) The daily VC configurations of each cluster from *Slurm*. Besides, we leverage node allocation details from the job logs to infer the timing information for each cluster.

To the best of our knowledge, this is the largest set of DL job traces, and also the first one with comprehensive types of jobs in addition to DL training. We release these traces to the research community, and expect they can benefit researchers for DL workload analysis and design of GPU datacenter systems.

Comparisons with the Philly Trace Microsoft released a trace of DL training jobs from its internal cluster Philly [24]. It is currently the most popular public trace containing rich information about production-level DL training jobs. A quantity of works on GPU resource management and optimization leveraged this Philly trace for analysis and evaluation [1, 42, 45–47, 56, 57].

DL has experienced rapid development over the years. New models and algorithms are emerging with increased resource demands [22, 41]. There is a 10.5× year-by-year increase in the number of DL jobs in Microsoft datacenters [45]. Hence, the Philly trace collected in 2017 may not be able to accurately reflect the characteristics of modern GPU clusters and DL jobs. We expect our trace can fill this gap with a larger number of jobs and the latest DL models and algorithms. We present detailed comparisons between our Helios trace and Microsoft Philly trace. Although the Philly

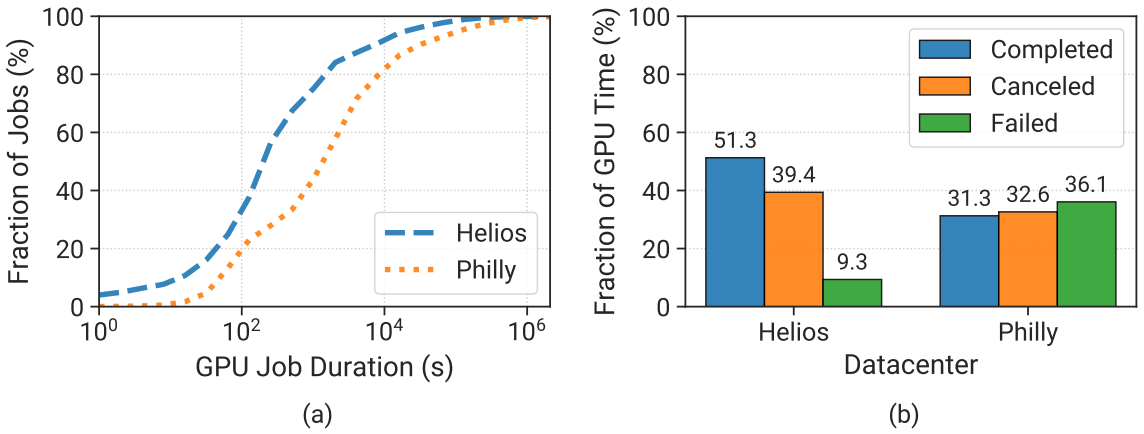


FIGURE 2.1: Comparisons of job characteristics between Helios and Philly. (a) The CDFs of the GPU job duration. (b) Distribution of GPU time by the final job status.

trace ranges from August 2017 to December 2017, jobs in the first two months exhibit abnormal behaviors (much less density and different features). So we select the period of October 2017 to December 2017 with 103,467 jobs, which was also adopted in the original paper [24].

Table 2.2 summarizes the comparisons between Helios and Philly traces. Our traces are collected from 4 independent clusters, while the Philly trace only describes one cluster. Helios contains 32.6 times more jobs than Philly and around half of jobs from Helios are CPU jobs. We will reveal some interesting insights through analyzing different characteristics of CPU & GPU jobs in §2.3, which cannot be learned from Philly. Moreover, the average number of GPUs required by our GPU jobs is over twice that of Philly. The maximum number of requested GPUs from our traces is 2048, which is an order of magnitude higher than Philly. These indicate that our traces provides new features of GPU clusters and jobs, which can increase the plurality and generality in DL system research.

We calculate the average duration of GPU jobs from Helios, which is much lower than Philly. There are three possible reasons: (1) Philly adopted *A YARN* [58] to schedule jobs, where failed jobs would be retried for a fixed number of times. Such retrials were counted into the entire job duration, statistically resulting in longer execution time. For instance, the longest job took about 60 days for a total of 3 attempts due to job failures, although the successful execution only took around 25 days. If we profile the Philly trace by regarding each attempt as an individual job, the average duration will drop to 17,398 seconds. (2) We keep all GPU jobs in Helios to reflect the realistic characteristics of a GPU datacenter. These include debugging and testing jobs which are much shorter than normal training jobs, causing a lower average duration. (3) Our datacenter provides more computing resources. Users usually request more resources

(two more GPUs on average) for training, which can also significantly accelerate the training process.

Figure 2.1(a) illustrates the Cumulative Distribution Function (CDF) of job duration. We observe that Philly jobs statistically took more time than Helios. This is consistent with our analysis from Table 2.2. Figure 2.1(b) shows the percentages of GPU time occupied by jobs with different statuses. We can see a significant fraction of GPU time contributed to the jobs which were finally ended up with the failure or canceled status. Particularly, over one-third of GPU time was wasted for the failed jobs in Philly and 9.3% in Helios.

We emphasize some terminologies in the job traces, which will be widely mentioned in our following analysis.

Job status: a job can end up with one of five statuses: (1) completed: it is finished successfully; (2) canceled: it is terminated by the user; (3) failed: it is terminated due to internal or external errors; (4) timeout: the execution time is out of limit; (5) node fail: it is terminated due to the node crash. Timeout and node fail are very rare in our traces, and will be regarded as failed in this study.

CPU job: this job is executed without any GPUs (e.g., image preprocessing, file decompression).

GPU job: the job needs to be executed on GPUs for acceleration (e.g., DL model training, model evaluation).

GPU time: this metric is used to quantify the amount of GPU resources required by the job. It is calculated as the product of total execution time and the number of GPUs.

CPU time: this is the product of total execution time and the number of CPUs. It is only considered for CPU job analysis.

Cluster utilization: this metric is used to characterize the resource utilization of a cluster. Since GPUs are the dominant resources in DL jobs, we calculate the cluster utilization as the ratio of active GPUs among the total GPUs in the cluster.

2.3 Characterization of DL Jobs

In this section, we perform a thorough analysis of our job traces. Some prior works analyzed the traditional big data traces from real-world datacenters [59–62]. In contrast, very few studies focused on the analysis of DL jobs. [24, 50] performed an empirical study of the characteristics of their clusters. We give more comprehensive analysis from the perspectives of clusters (§2.3.1), jobs (§2.3.2) and users (§2.3.3).

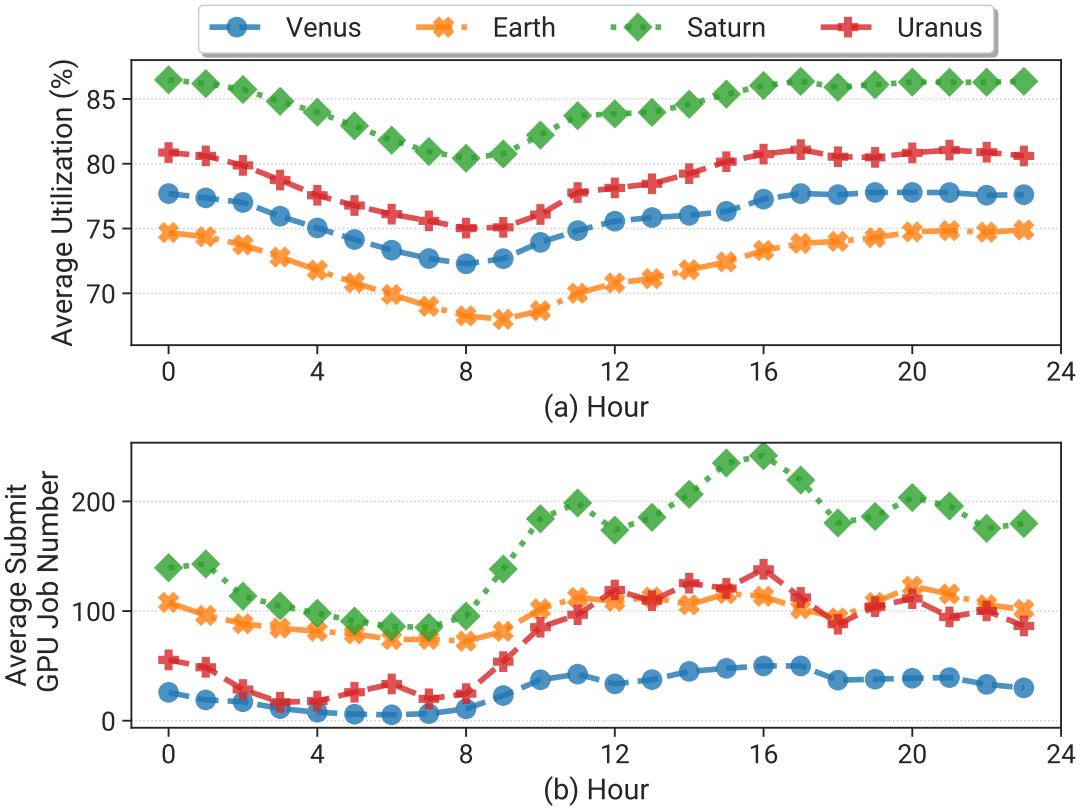


FIGURE 2.2: Daily pattern of the cluster usage in Helios. (a) Hourly average cluster utilization over six months. (b) Hourly average GPU job submission rates over six months.

Our traces cover different characteristics of DL jobs and behaviors of AI developers and researchers in SenseTime. For instance, users can submit long-term production jobs, as well as short-term exploratory jobs for debugging purposes. Users might early stop their jobs when they can (not) reach the expected performance. We perform our assessment statistically, and believe the conclusions are general for other organizations and clusters as well.

2.3.1 Cluster Characterization

Daily Trends of Cluster Usage Figure 2.2(a) shows the average cluster utilization for every hour in one day. All the clusters and users are in the same timezone, and hence exhibit similar patterns for the daily cluster usage. The utilization of all the clusters ranges from 65% to 90%. `Saturn` has the highest utilization, while `Venus` and `Earth` are relatively underutilized. The standard deviation of hourly utilization in `Saturn` is 7% and ranging from 10% to 12% in other clusters. Besides, we observe a 5$\sim$ 8% decrease at night (0 am – 8 am) for all the clusters, which is not very significant. This is because the workloads in Helios are mainly DL training jobs, which can take

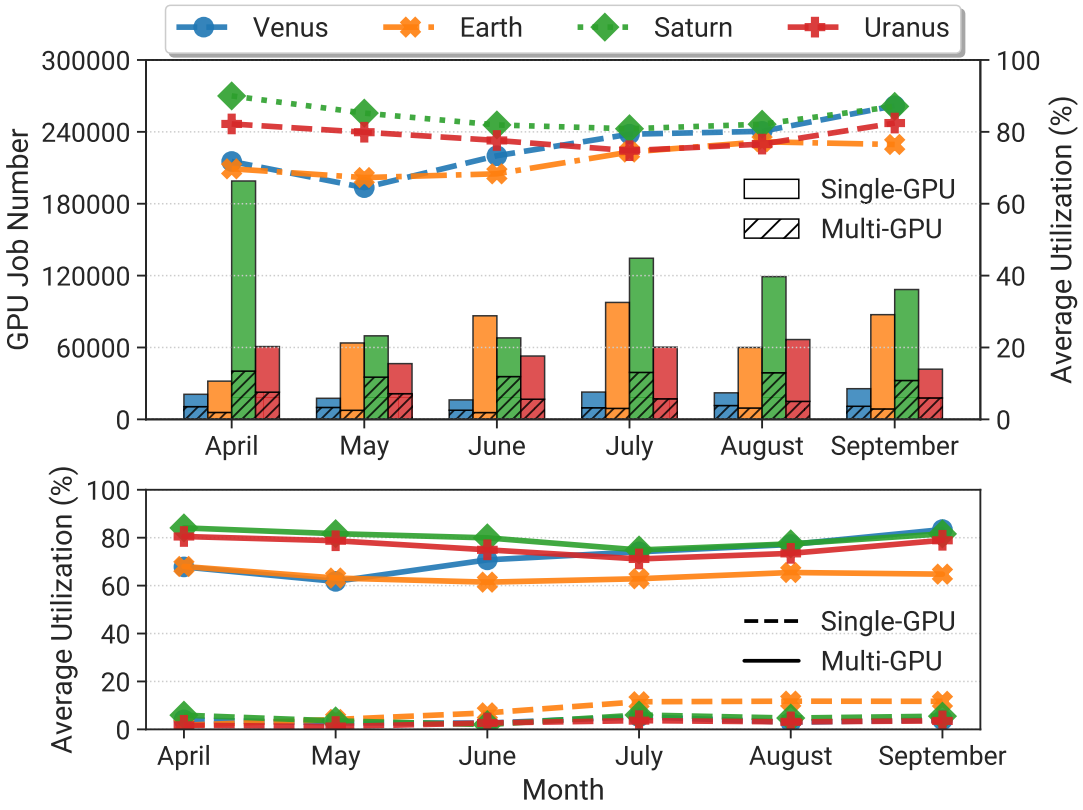


FIGURE 2.3: Monthly trends of cluster activities in Helios. Top: number of submitted (single- and multi-GPU) jobs (*bars*) and average cluster utilization (*dashed lines*). Bottom: average cluster utilization from multi-GPU jobs (*solid lines*) and single-GPU jobs (*dashed lines*).

hours or days to complete. It is common that some jobs are submitted in the daytime but still keep running overnight.

Figure 2.2(b) shows the average GPU job submission rate for each hour during the six months. All the clusters have similar patterns of the job submission in one day: the number drops to the lowest point at night (sleep), and experiences a slight drop around 12pm (lunch) and 6pm (dinner). It is also interesting to note that `Earth` has a stable and high submission rate ($\sim$ 100 jobs) per hour, but has the lowest utilization among the four clusters. This is because the GPU jobs in `Earth` are overall shorter than the other clusters. The cluster utilization depends on both the number of jobs as well as their running time. Further, the frequency of job submission is much lower than big data clusters [48, 59]. This implies some time-consuming scheduling optimization algorithms would apply for scheduling DL training jobs.

Implication #1: Both the cluster utilization and the job submission rate exhibit obvious daily patterns. This provides us opportunities to predict those behaviors in advance and then perform the optimal resource management and job scheduling.

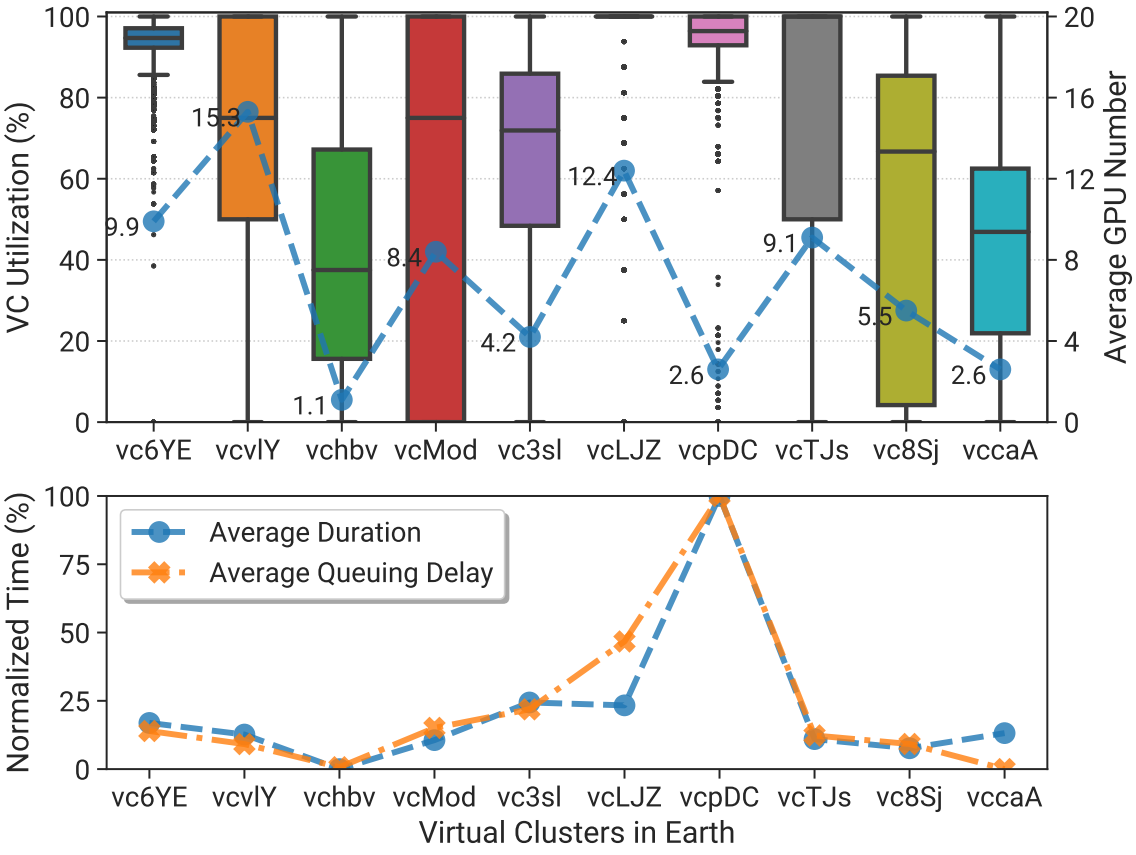


FIGURE 2.4: VC behaviors in `Earth`. Top: The boxplot of utilization distributions for the top 10 largest VCs and the job's average number of requested GPUs in each VC (*dashed line*). Bottom: Min-Max normalized average job duration (blue dashed line) and queuing delay (orange dashed line).

Monthly Trends of Cluster Usage We further analyze the monthly trends of GPU resources and job behaviors. Figure 2.3 (top) shows the monthly statistics of GPU job submission[1] (bars) and average cluster utilization (dashed lines). All the clusters have stable submissions of multi-GPU jobs each month, while the numbers of single-GPU jobs fluctuate dramatically. The utilization of `Saturn` and `Earth` remains stable under varied numbers of GPU jobs per month. Surprisingly, `Saturn` executed almost twice GPU jobs in July compared with May or June, whereas the utilization in May (85.21%) and June (81.92%) is even higher than July (80.87%). Furthermore, we find the distribution of multi-GPU jobs within the same cluster is similar each month. The average number of requested GPUs is very close with a standard deviation of 2.9. Therefore, we can accurately predict the monthly submissions of multi-GPU jobs from the previous months' data. Figure 2.3 (bottom) presents the cluster utilization contributed by single-GPU and multi-GPU jobs. It is evident that single-GPU jobs

[1]Our traces end on September 27th. So the reported numbers of September are around 10% lower than the actual one.

have little influence on the overall cluster utilization (less than 6% except for `Earth`). Conversely, multi-GPU jobs are dominant to cluster utilization.

Implication #2: For monthly trends, it is infeasible and unnecessary to predict the submissions of single-GPU jobs due to their weak impact on the cluster usage. In contrast, multi-GPU jobs exhibit more stable monthly patterns, and are critical to cluster utilization, which we can predict for better scheduling efficiency.

Virtual Cluster Behaviors In addition to the entire physical clusters, investigation of VCs is also indispensable. We select a period when the VC configuration remains stable (May in `Earth`). Figure 2.4 (top) shows the utilization distributions of the 10 largest VCs (in descending order) averaged per minute. Specifically, there are 208 GPUs in *vc6YE* and 32∼ 96 GPUs in other VCs. Each box is framed by the first and third quartiles, while the black line inside the box represents the median value. Both whiskers are defined at 1.5 times the InterQuartile Range (IQR). We plot the job's average number of requested GPUs for each VC above the corresponding box. Figure 2.4 (bottom) shows the average queuing delay and job duration for each VC.

We find the behaviors of each VC vary significantly, as they run different types of GPU jobs in terms of resource demands and duration. First, we observe that the VC utilization is positively correlated with the average GPU demands. *vc6YE* and *vcLJZ* keep over 90% utilization most of the time as they generally run large jobs. In contrast, the utilization of *vchbv* and *vccaA* is basically below 65% for hosting small jobs. One exception is *vcpDC*, which has high utilization but small average numbers of GPUs. Second, the job queuing delay is approximately proportional to the average job duration. Busy VCs (e.g., *vcLJZ* and *vcpDC*) typically have much longer queuing delay. These prove that job queuing and resource underutilization co-exist in our clusters due to imbalanced VCs.

The key reason is the adoption of static partitioning with VCs. This simple and mature solution is widely used in production GPU clusters (e.g., Microsoft [24, 47], Alibaba [1, 50]) for fairness among multi tenants. However, it also causes long queuing delay and resource underutilization. Researchers also designed advanced scheduling algorithms to address these issues with high fairness [42, 44, 47]. How to implement them into production clusters with better reliability and robustness will be an important future work.

Implication #3: Different groups submit DL jobs to their VCs with distinct GPU demands and duration. Hence, the imbalanced resource allocation across VCs can lead to low resource utilization and severe job queuing delay [63]. It is critical to consider fairness when designing schedulers for shared clusters [42, 44, 47].

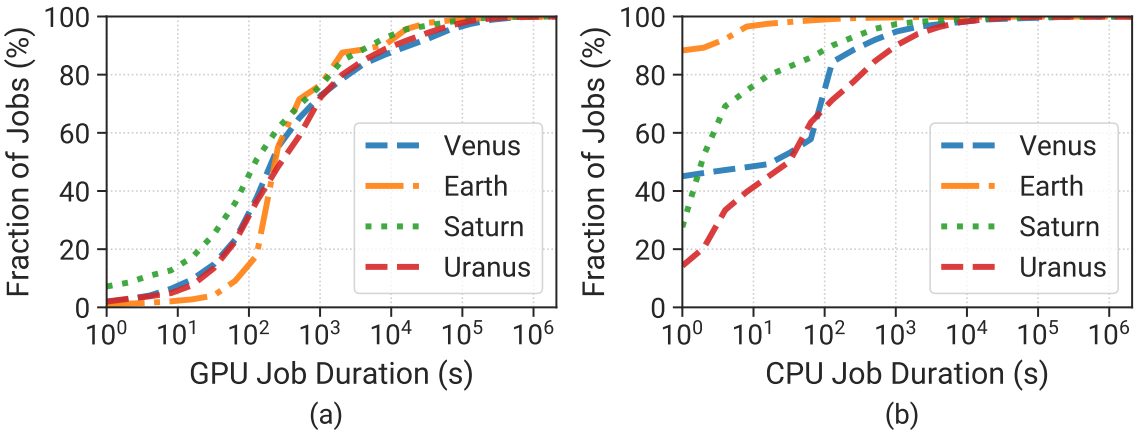


FIGURE 2.5: The CDFs of (a) GPU and (b) CPU job duration.

2.3.2 Job Characterization

Beyond the cluster statistics, our traces also contain rich information at the job level. We draw interesting conclusions from such information, as discussed below.

Job Execution Time As shown in Table 2.2, the number of GPU jobs is close to the number of CPU jobs in the Helios traces. However, the average execution time of GPU jobs (6,652s) is 10.6× longer than CPU jobs (629s). More than 50% of CPU jobs run for less than 2s. In contrast, the median execution time of GPU jobs is 206s. More specifically, Figure 2.5 compares the duration distributions of GPU and CPU jobs in each cluster. The duration of GPU jobs ranges from seconds to weeks, and is generally over an order of magnitude longer than CPU jobs. Interestingly, these four clusters have diverse duration distributions of CPU jobs, while similar distributions for GPU jobs. In `Earth`, short-term CPU jobs account for a larger portion compared with other clusters: nearly 90% of CPU jobs run for only one second in `Earth`. Most of them are related to training progress and node state queries. As for GPU jobs, roughly three-quarters of jobs last for less than 1000 seconds as they are mainly for model evaluation and program debugging.

To dive deeper into the characteristics of GPU jobs in each cluster, we investigate the relationship between the GPU demands, GPU time and number of GPU jobs. We show the CDFs of requested GPU demands with the number of jobs (Figure 2.6(a)) and GPU time (Figure 2.6(b)). We observe there are over 50% single-GPU jobs in each cluster, and the largest ratio is 90% in `Earth`. However, they only occupy 3∼12% of the total GPU time. In contrast, although the proportion of large-size jobs (≥8 GPUs) is smaller than 10%, they account for around 60% of computing resources.

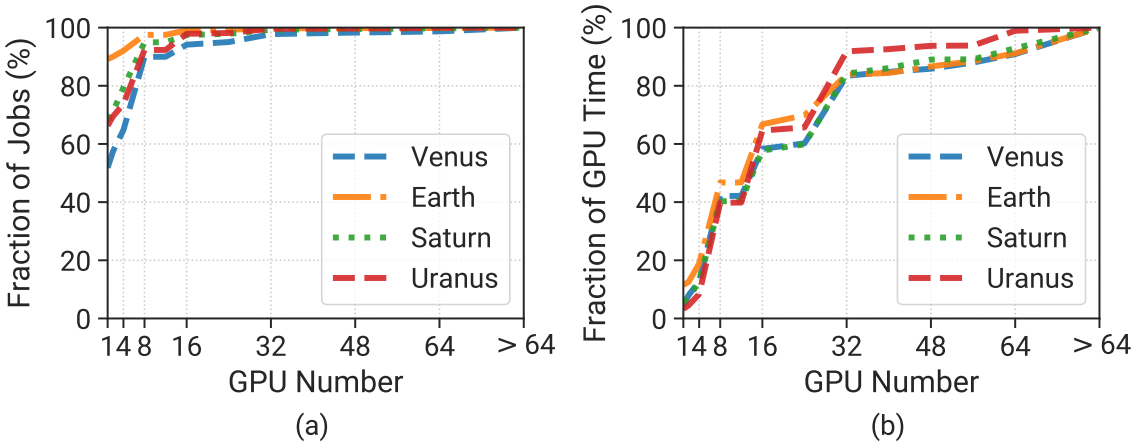


FIGURE 2.6: The CDFs of job sizes (in GPU number) with the number of (a) jobs and (b) GPU time.

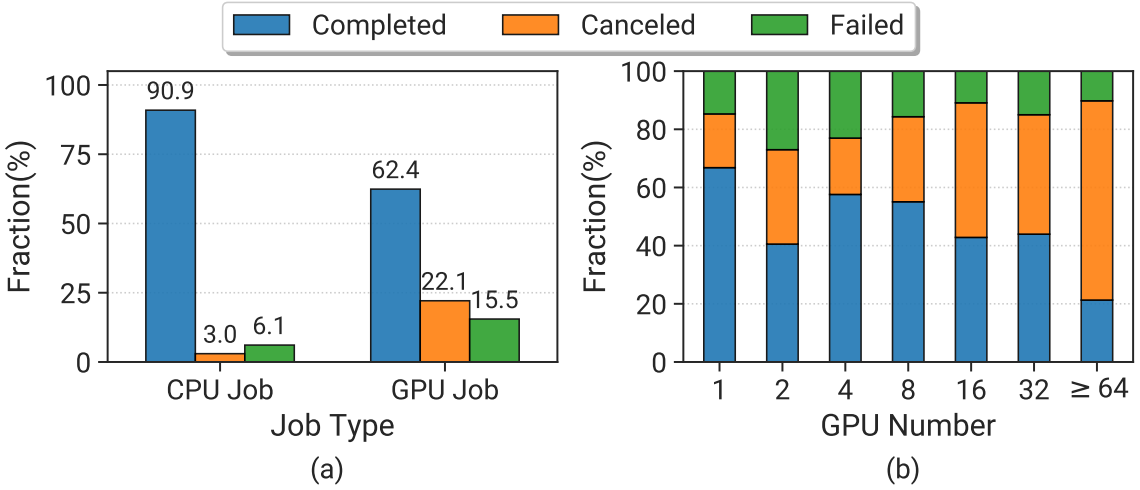


FIGURE 2.7: Distribution of jobs by their final statuses. (a) Comparisons of CPU and GPU jobs for their final statuses. (b) Percentages of final job statuses w.r.t. different GPU demands.

Implication #4: Despite the number of single-GPU jobs is predominant, GPU resources are mainly consumed by multi-GPU jobs. Hence, optimization of multi-GPU jobs is more important to improve cluster efficiency. This characteristic resembles traditional HPC workloads [48, 49, 64] and implies some optimization techniques in HPC can also be applied to GPU clusters.

Job Final Statuses Figure 2.7(a) summarizes the distributions of final statuses for CPU and GPU jobs of these 4 clusters. The ratio of unsuccessful GPU jobs (37.6%) is significantly higher than CPU jobs (9.1%). One reason is that some users prefer to terminate their DL training jobs in advance as the model accuracy already converges to the satisfactory value. These canceled jobs are still successful. Another reason is that users can inspect the training states and kill the poor-performing jobs earlier.

This reflects the *feedback-driven exploration* feature of DL training jobs. This early-stopping feature can be leveraged to optimize the scheduling efficiency. For instance, [18, 43] help users automatically make the early-stopping decision through recording training feedback and predicting future training performance. This can bring a huge benefit to datacenters.

There are also other causes for failed jobs, including timeout, node failure, incorrect inputs, runtime failure, etc. Microsoft [24, 65] presented a detailed analysis of the reasons for training job failures, so we do not conduct similar investigations in this chapter.

Implication #5: Since many DL training jobs can reach the convergence earlier than expected, the scheduler can automatically detect this condition and stop the jobs for resource efficiency [18, 43]. Users can use different metrics (e.g., loss, accuracy) and authorize the scheduler to monitor and manage their jobs.

In Figure 2.7(b), we quantify the ratios of different job final statuses in terms of GPU demands. We only consider the GPU numbers of $2^k (k \in \mathbb{N})$ as they are mostly requested in Helios. We observe that the ratio of job completion keeps decreasing as the number of GPUs increases, with an exception of 2-GPU jobs. For large jobs with 64 or more GPUs, only fewer than a quarter of jobs complete successfully while the canceled ratio even reaches roughly 70%. This is because these jobs typically run for very long time, and users have higher chances to early stop them to save time and resources.

Additionally, we find most failed jobs are terminated within a short time, which matches the conclusions in prior works [24, 65]. The majority of failures are incurred by user errors, such as script configuration, syntax/semantic errors in the program. However, plenty of short-term debugging jobs suffer from severe queuing delays. Users usually fail to get the code debugging feedback timely, which considerably affects their experience.

Implication #6: A lot of failed jobs are for debugging purposes, and last for a very short time. However, they are mixed with the long-term production jobs in the queue and possibly suffer from much longer waiting time than execution. A possible solution is to allocate a special VC for debugging jobs and enforce a short-term limit (e.g., 2 minutes). This can help users obtain error messages timely and filter most failed jobs for normal clusters.

2.3.3 User Characterization

We analyze the traces from the users' perspective, to discover methods for user experience enhancement. This has never been considered in prior works about DL job

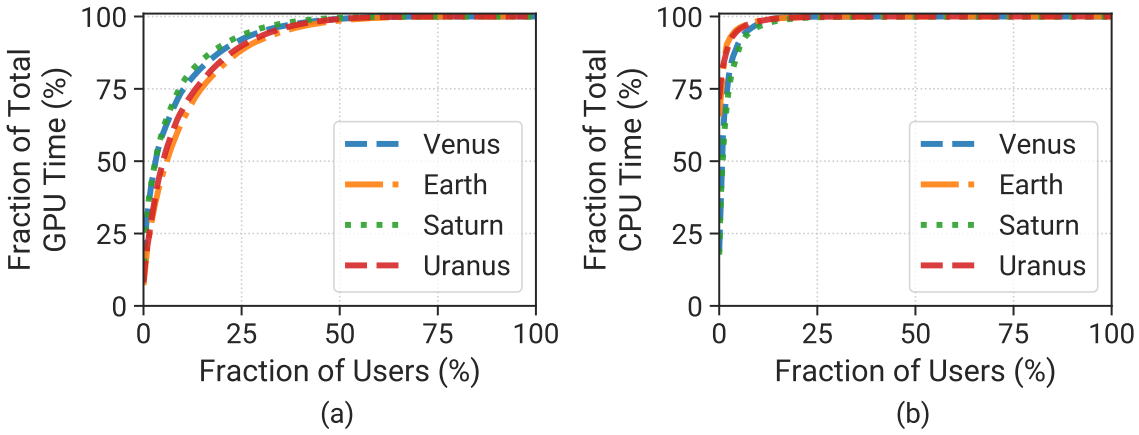


FIGURE 2.8: The CDFs of users that consume the cluster resources in terms of (a) GPU Time (b) CPU Time.

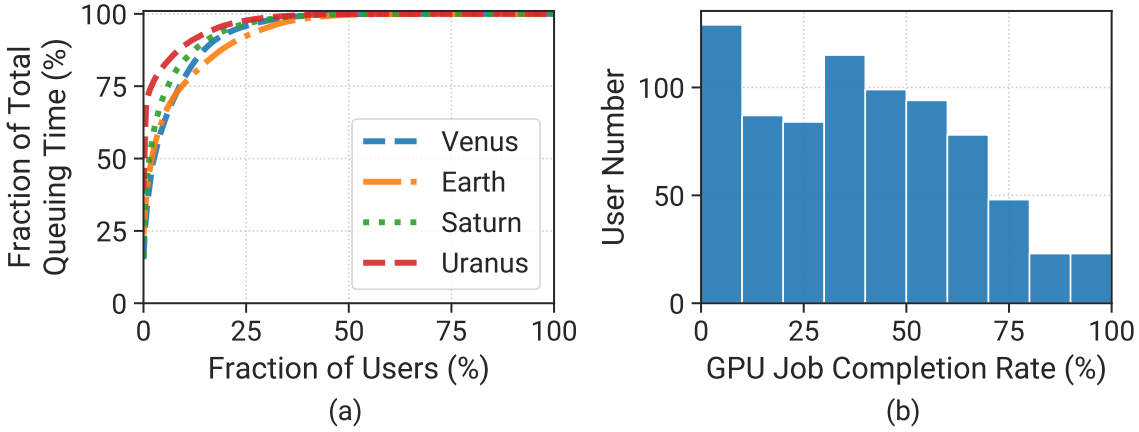


FIGURE 2.9: (a) The CDFs of users w.r.t. GPU job queuing delay. (b) The distribution of user GPU job completion ratios.

analysis. Each cluster has 200∼400 users. Some users can submit jobs to multiple clusters concurrently.

Similar to our analysis of job trends, we first investigate the consumption of CPU and GPU resources at the user level, as shown in Figure 2.8. We find the trends are similar across all the clusters. Compared with GPU time, the CDF curves of CPU time are much steeper, indicating CPU jobs are more concentrated within a small portion of users. This is because only 25% of users on average need to conduct CPU tasks (e.g., video frames extraction), and the top 5% of users occupy over 90% CPU time. In contrast, almost every user has GPU training jobs, and the top 5% of users consume 45∼60% GPU time.

Next, we study the distributions of GPU job queuing delay among users, as shown in Figure 2.9(a). We observe that most users do not suffer from severe job queuing, whereas a few users have jobs blocked for a long time. In `Uranus`, the top 1% of users

(only 3) bear over 70% queuing time, even they are not among the top 10 resource-consumption users. We name them "marquee users" [66], and their experiences need to be ameliorated.

Figure 2.9(b) shows the distributions of users for different GPU job completion rates. It is obvious that the users' GPU job completion rates are generally low, which proves that the high fraction of unsuccessful GPU jobs (shown in Figure 2.7) reflects the users' overall behaviors instead of some individual ones.

Implication #7: To alleviate the problem of unfair cluster queuing, it is recommended that the scheduler should consider our user-level analysis to perform the corresponding optimization. For instance, the scheduler can dynamically adjust temporary priorities to users, especially to the marquee ones, based on their current job queuing statuses. The VC configuration can also be regulated appropriately according to users' behaviors.

2.4 A Prediction-Based Framework

From §2.3, we find the feasibility of predicting clusters' behaviors (e.g., job duration, node states) from the history. Inspired by this observation, we design a novel prediction-based GPU resource management framework, which leverages the historical data to improve the resource usage efficiency and workload performance. Note that the source code for the framework is openly available at `https://github.com/S-Lab-System-Group/HeliosArtifact`. We have provided all ML model and system-related hyperparameters to facilitate the reproduction of our results. Feel free to modify these parameters to suit your specific cluster environment.

2.4.1 Framework Overview

Figure 2.10 illustrates the overview of our framework. It is designed as a centralized manager built atop each GPU cluster. It adopts the "plug-and-play" fashion, where different resource management services can be integrated into this framework. Each service is independent and targets a different perspective of optimization. They share common design philosophy, and follow the same workflow. The cluster operators can select services based on their demands.

The framework consists of two components: *Model Update Engine* and *Resource Orchestrator*. For each service, a machine learning model is trained to predict the job behaviors or cluster states. During the operation, the *Resource Orchestrator* uses the model to predict the upcoming *events* and determines the optimal *resource management* operation (❶), which will be executed in the cluster. Meanwhile, the *Model*

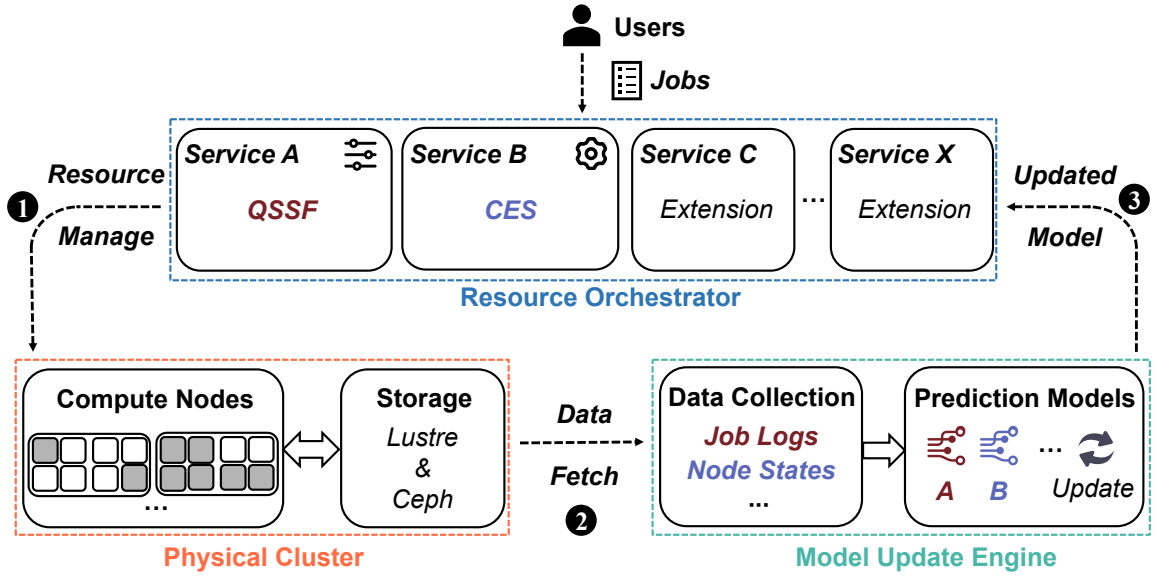


FIGURE 2.10: Overview of our prediction-based framework.

Update Engine fetches the run-time data (❷) regularly (e.g., every minute) or triggered by events, and fine-tunes the model periodically to adapt to the changes in the cluster (❸).

Our framework has several benefits. (1) *Extensibility*: we provide an abstract pipeline for resource management. As case studies, we design two novel services: *Quasi-Shortest-Service-First Scheduling* to minimize the cluster-wide average JCT (§2.4.2), and *Cluster Energy Saving* to improve the cluster energy efficiency (§2.4.3). Other services based on machine learning prediction can also be integrated into our framework, e.g., burstiness-aware resource manager [67, 68], network-aware job scheduler [69, 70], etc. (2) *High usability*: our framework can be deployed into arbitrary GPU clusters. The services work as plugins atop the current management systems (e.g. *Slurm*, *YARN*, *Kube-scheduler*) with minimal or no modifications to them. Users do not need to provide extra information or specifications. (3) *Low overhead*: the prediction and operation latency for each service typically takes milliseconds, which is negligible for DL workloads.

2.4.2 Quasi-Shortest-Service-First Scheduling

Motivation. All GPU clusters in Helios are managed by *Slurm*. Jobs are scheduled using the vanilla First-In-First-Out (FIFO) algorithm, similar to *YARN*'s Capacity Scheduler [58]. Due to such runtime-unaware scheduling style [71], users complain that even short-term jobs can suffer from long queuing delays.

Both Shortest-Job-First (SJF) and Shortest-Remaining-Time-First (SRTF) algorithms were proposed to reduce the average JCT [45, 72–74] with and without preemption

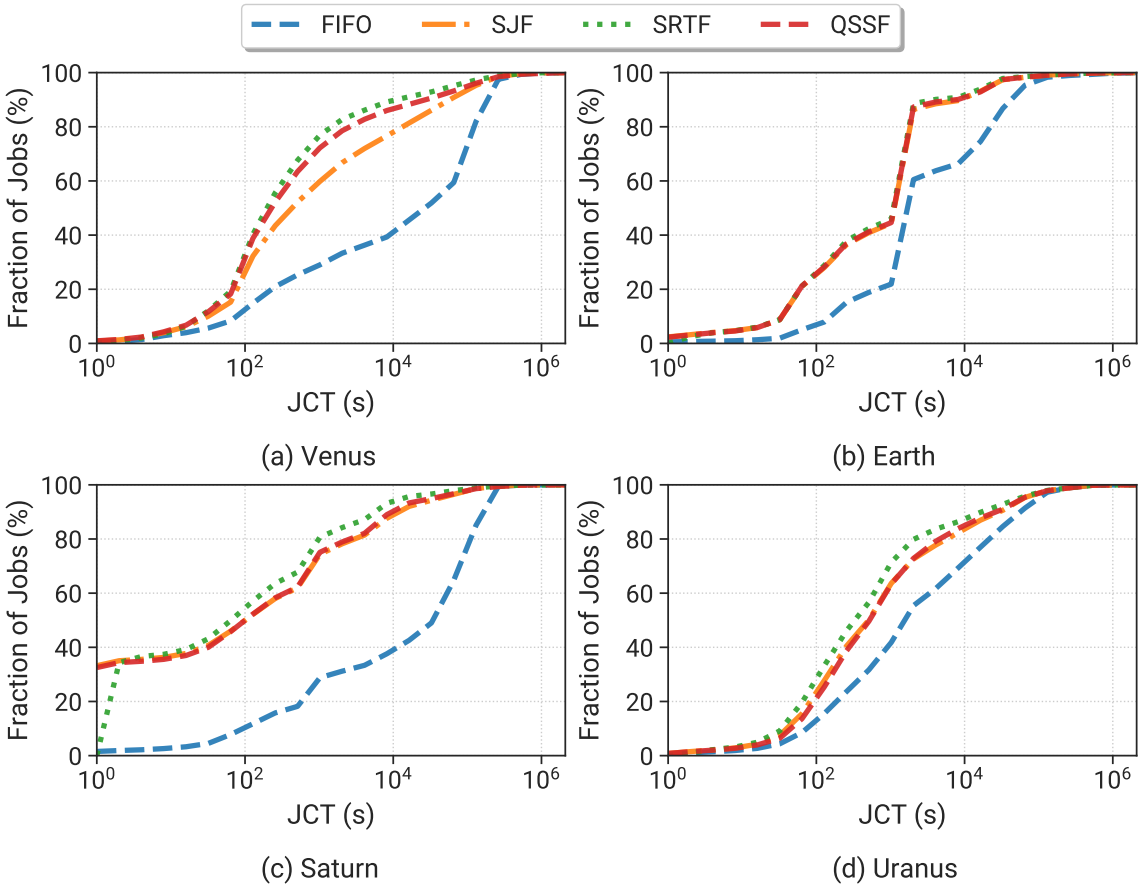


FIGURE 2.11: Comparisons of JCT distributions using different scheduling algorithms, based on the September job traces across 4 clusters in Helios. Note that SJF and SRTF are optimal baselines (with perfect job duration information).

respectively. However, they are too ideal and thus impractical in GPU clusters, due to the following two reasons: (1) some jobs in GPU clusters do not have iterative characteristics, and cannot be preempted and then restored from the checkpoints. Hence, the preemption-enabled scheduling algorithms (SRTF) are not applicable for these jobs. (2) These algorithms rely on the information of job duration or remaining execution time, which is uncertain and could be affected by many unexpected factors (e.g., termination in advance, training early-stopping [75], error and node crash, etc.). To address this issue, prior works try to obtain the job runtime information from the users [16, 58, 76], job profiling [77–81] or leveraging the job's periodic features [69, 82–84]. These approaches can either affect the usability or operation cost, or lack of generality for different types of workloads.

Driven by these limitations, we design a new Quasi-Shortest-Service-First (QSSF) scheduling service. This scheduler adopts the non-preemption mechanism, which can be applied to different types of jobs in our datacenter. Owing to the constraint of the gang scheduling for DL training jobs, large-size short-term jobs can occupy many GPUs, which can block multiple small-size jobs using the SJF scheduler [45]. We

Algorithm 1 Quasi-Shortest-Service-First Scheduler

Input: New job: $\mathcal{J}$, **VCQueue:** $\mathbb{Q}$, **Historical job trace:** $\mathbb{J}$

procedure QSSF SCHEDULE($\mathcal{J}$, $\mathbb{Q}$, $\mathbb{J}$)
  if $\mathcal{J}$ is not $\varnothing$ **then** ▷*Receive a new job*
    $\mathcal{J}.priority$ = Priority($\mathcal{J}$, $\mathbb{J}$) ▷*Assign priorty to the new job*
    Enqueue $\mathcal{J}$ to $\mathbb{Q}$
    SortJobPriority($\mathbb{Q}$) ▷*Sort by job priority*
  for all $Job \in \mathbb{Q}$ **do**
    if Consolidate(Job) is **True then**
      ConsolidateAllocate(Job) ▷*Job placement*
      Dequeue Job from $\mathbb{Q}$
    else
      break

function PRIORITY($\mathcal{J}$, $\mathbb{J}$)
  if UserMatch($\mathcal{J}.user, \mathbb{J}.users$) is $\varnothing$ **then**
    $\mathcal{P}_R$ = Distribution($\mathbb{J}.durations$) ▷*New user*
  else if SimilarName($\mathcal{J}.name$, $\mathbb{J}.user.names$) is $\varnothing$ **then**
    $\mathcal{P}_R$ = Distribution($\mathbb{J}.user.durations$) ▷*No matching historical job name*
  else
    $\mathcal{P}_R$ = RollingEstimator($\mathcal{J}$, $\mathbb{J}$) ▷*Exist matching historical job name*
  $\mathcal{P}_M$ = MLEstimator($\mathcal{J}$)
  $\mathcal{P} = \mathcal{N}(\lambda\mathcal{P}_R + (1-\lambda)\mathcal{P}_M)$ ▷$\mathcal{N}$: *GPU number of* $\mathcal{J}$, λ: *Merging coefficient*
  return $\mathcal{P}$

choose to rank jobs with GPU time instead of duration. Through the trace analysis, we find there exist strong correlations between the job duration and some attributes, such as job name, user, GPU demands, submission time, etc. Hence, our scheduler leverages these attributes to predict the jobs' priority orders (i.e., expected GPU time) for scheduling. Similar ideas were also applied for scheduling big data workloads [71, 85]. We consider more attributes with a machine learning model ensemble to enhance the scheduling performance for DL jobs.

Service Design Our QSSF scheduler builds a machine learning model to predict the expected GPU time of incoming jobs. When a user submits a DL job to the cluster, the scheduler immediately retrieves the relevant attributes (e.g., GPU & CPU demands, job name, user id, target VC, etc) and infers the job's expected GPU time from the model. Then the scheduler selects and allocates jobs based on the predicted GPU time and cluster resource states. After the job termination, the job's final information will be collected by the *Model Update Engine* for fine-tuning the prediction model.

We train a Gradient Boosting Decision Tree (GBDT) [86] model to capture the overall trend of the relevant jobs. Specifically, we extract all features and actual duration from the traces to construct a training and validation set. We encode all the category features (e.g., user name, VC name, job name). For the extremely sparse and high-dimensional features of job names, we utilize the Levenshtein distance [87] to cluster

the names and bucketize similar ones, which convert them into relatively dense numerical values. For the time-related features (e.g., job submission time), we parse them into several time attributes, such as month, day of the week, hour, minute. Finally, we train a GBDT model that can map these job attributes to the corresponding duration.

Algorithm 1 shows the pseudo-code of our QSSF algorithm. It operates on a scheduling frequency of one second, during which it evaluates the cluster's spare resources and the job queue, attempting resource allocation (line 6∼11). Besides, when a new job arrives, the scheduler assigns it a priority value, enqueues it, and subsequently sorts the job queue (lines 2 to 5). The core function of QSSF is PRIORITY (line 13), which returns a priority value ($\mathcal{P}$) for a given job ($\mathcal{J}$). $\mathcal{P}$ is calculated as the weighted sum of a rolling estimate $\mathcal{P}_R$ and machine learning estimate $\mathcal{P}_M$. The rolling estimate $\mathcal{P}_R$ is computed directly from the historical jobs with similar attributes. There are three cases for calculating $\mathcal{P}_R$: (1) If the job user cannot be found in the traces (new user), then $\mathcal{P}_R$ is the average duration of all the jobs with the same GPU demands in the traces (line 15). If the traces have the job submission records of the user (existing user), then we leverage the Levenshtein distance to find historical jobs from this user, which have similar names or formats as the incoming one. According to the result of SIMILARNAME function, there are two cases for existing users: (2) If no such historical jobs are found, then $\mathcal{P}_R$ is the average duration of all this user's jobs with the same GPU demands in the traces (line 17). (3) Otherwise, we compute $\mathcal{P}_R$ via exponentially weighted decay of duration of historical jobs with matched names (line 19). The machine learning estimate $\mathcal{P}_M$ is computed by the GBDT model (line 20), which considers the overall trend of the relevant jobs. Finally, we combine the two estimates $\mathcal{P}_R$ and $\mathcal{P}_M$, and multiply the requested GPU number ($\mathcal{N}$) to get the job's expected GPU time $\mathcal{P}$ as the priority value (line 21), which can reflect both spatial and temporal aspects of the job [45].

We select a job from the VC queue with the highest priority (lowest predicted GPU time) for scheduling. For job placement, unless the topology is specified by the user, the *ConsolidateAllocate* policy is adopted to allocate each job on as few nodes as possible to reduce the communication overhead. For instance, a 16-GPU job needs to wait for two compute nodes with 8 idle GPUs. Other placement solutions will violate the consolidation principle.

Evaluation We develop a trace-driven simulator to evaluate our QSSF algorithm. It emulates a datacenter with the same configuration as Helios in Table 2.1, which operates with the real-world job workflow: job arrival – queuing – running – completion/canceled/failed. Since the GPU resources are the bottleneck in our clusters, we mainly consider the GPU jobs in our simulation. We train the GBDT model using

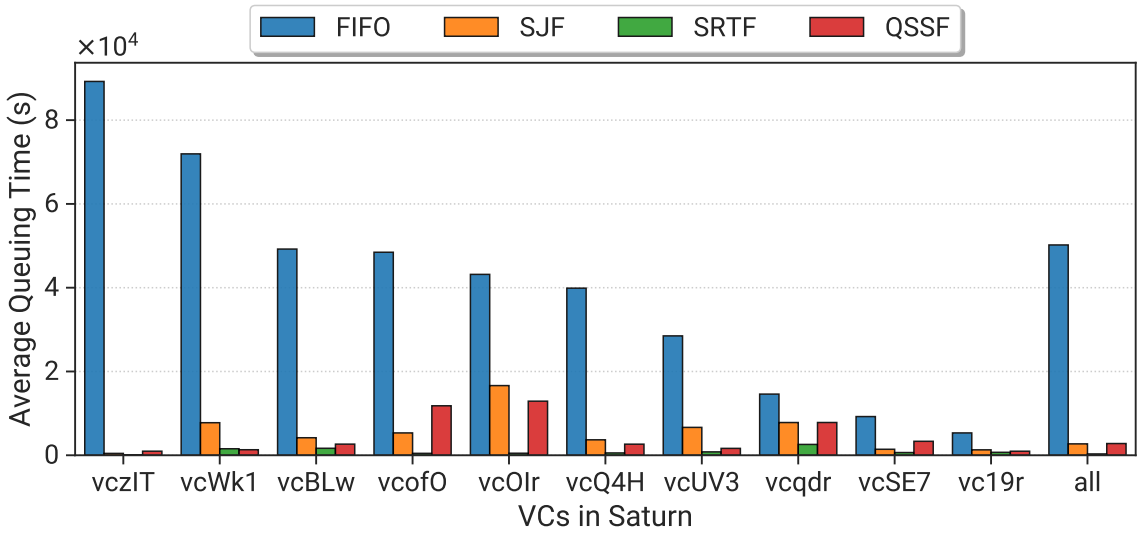


FIGURE 2.12: The average job queuing delay of the top 10 VCs in Saturn (September) with different scheduling algorithms. The column `all` represents the whole cluster.

the jobs from April to August in the traces, and evaluate the model with the jobs in September.

We consider three baseline algorithms for comparisons. (1) *FIFO*: the current scheduling policy in our clusters, which is simple but has poor performance. (2) *SJF*: the optimal policy to minimize the average JCT without preemption. (3) *SRTF*: the optimal preemption-enabled version of *SJF*, where we assume all the jobs can be preempted with negligible overhead. Both the SJF and SRTF algorithms are not practical, since they need perfect job duration information [45, 74], which cannot be obtained in reality. They serve as the upper bound of the scheduling performance. We assume the scheduler knows the exact job duration given in the trace. Also, we do not consider the backfill mechanism, as we want to explore how much benefit can be obtained solely from prediction. Integration of backfill with our QSSF service will be considered as future work. Furthermore, QSSF is not intended to be a state-of-the-art performance scheduler. For example, Tiresias [45] can achieve superior scheduling performance, albeit requiring preemption and code intrusion. In Chapter 4, QSSF serves as a baseline for our subsequent proposed scheduler, Lucid, which aims to surpass the performance of intrusive schedulers like Tiresias.

Figure 2.11 shows CDF curves of JCT in each cluster with different scheduling algorithms. We observe that our QSSF algorithm can significantly outperform the naive FIFO policy, and perform comparably with SRTF and SJF, without making unrealistic assumptions. The advantage of QSSF over FIFO is relatively smaller in `Uranus`, as the job queuing delay in this cluster is not as severe as the other three clusters:

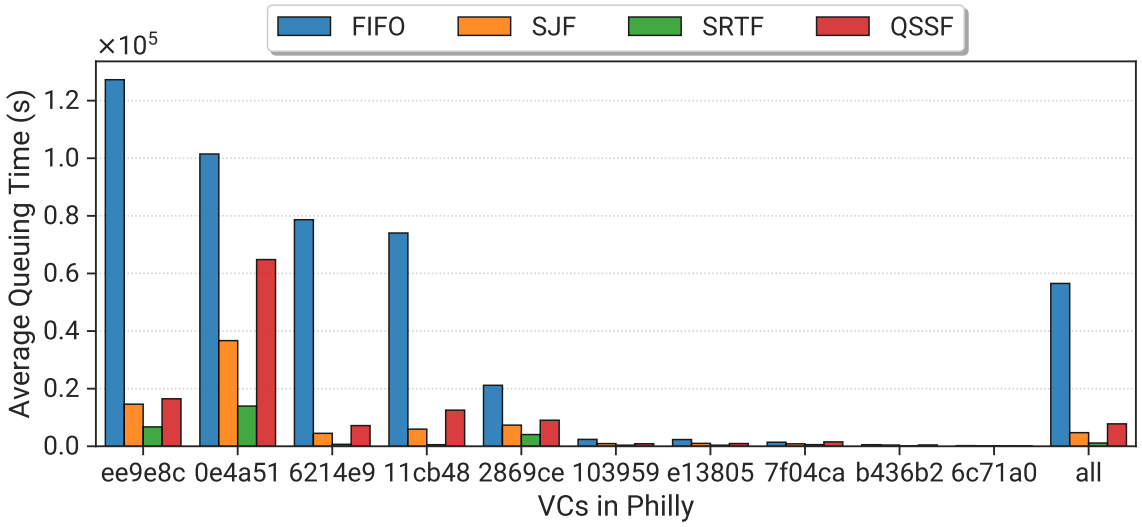


FIGURE 2.13: The average job queuing delay of the top 10 VCs in Philly (October and November) with different scheduling algorithms. The column `all` represents the whole cluster.

TABLE 2.3: Performance comparison of different schedulers.

		Venus	**Earth**	**Saturn**	**Uranus**	**Philly**
Average JCT (s)	FIFO	64,702	19,754	55,984	19,758	86,072
	SJF	21,095	6,892	8,501	13,226	34,272
	QSSF	18,349	6,732	8,581	13,123	37,324
Average Queuing Time (s)	FIFO	52,933	13,699	50,202	8,394	56,531
	SJF	9,325	837	2,719	1,861	4,731
	QSSF	6,580	677	2,798	1,759	7,783
# of Queuing Jobs	FIFO	15,336	30,030	65,991	16,917	30,282
	SJF	8,353	8,848	21,808	11,320	21,437
	QSSF	3,713	5,462	15,311	10,581	22,026

the queuing period takes 70∼90% of the total job completion time in other clusters while 42% in `Uranus` (Table 2.3).

From Table 2.3, QSSF reduces 37∼82% of queued jobs, which is even better than SJF. Interestingly, even though QSSF prioritizes short-term jobs to alleviate the head-blocking problem, the queuing delays of large jobs are also improved because of fewer queuing jobs. Table 2.4 shows the improvement of QSSF over SJF in different groups of jobs. Short-term jobs achieve at least 9.2× improvement while long-term jobs can also obtain 2.0∼4.8× improvement in Helios. This justifies that QSSF will not sacrifice the interest of long-term jobs and all kinds of jobs can benefit from our service.

To evaluate the performance differences caused by the scheduler in each VC, we depict the average job queuing delay of VCs in `Saturn`, as shown in Figure 2.12. We select

TABLE 2.4: The ratio of queuing delay between FIFO and QSSF in different job groups. A higher ratio indicates shorter delay and better efficiency in QSSF.

	Venus	**Earth**	**Saturn**	**Uranus**	**Philly**
short-term (<15 mins)	11.44	33.51	22.88	9.24	26.95
middle-term (15 mins∼6 hours)	4.13	13.39	7.39	2.49	5.54
long-term (>6 hours)	3.22	4.77	3.56	2.00	1.70

the top-10 VCs with the highest average queuing time, while the other VCs have little delay. We observe the average queuing time of QSSF is almost identical to SJF, and remain stable in each VC. Furthermore, QSSF even outperforms SRTF in *vcWk1*. This confirms the importance of GPU demands to enhance the scheduling efficiency, since large-size short-term jobs may block many small-size jobs. To summarize, compared with FIFO, QSSF achieves 1.5∼6.5× improvement in average JCT, and 4.8∼20.2× improvement in average queuing delay among 4 clusters in Helios.

In addition, we also evaluate the applicability of our QSSF service on the Philly trace. Since Microsoft did not release sufficient trace information (e.g., job names and VC configurations) which is necessary for our service, we make two reasonable assumptions. (1). The VC configurations are static during our evaluation period (from 1st October to 30th November, 2017) and the size of each VC is set corresponding to its workloads. (2). The priority values are generated randomly with a similar error distribution as Helios estimation. As shown in Table 2.3, QSSF obtains comparable performance with the optimal SJF in Philly, even without precise estimates. It achieves 2.3× improvement in average JCT, 7.3× improvement in average queuing delay, and reduces 27% of queued jobs. Figure 2.13 presents the VC-level analysis of QSSF performance. We observe QSSF brings large improvement in each VC with regard to the average queuing time.

2.4.3 Cluster Energy Saving

Motivation Electricity dominates the operation cost of modern GPU datacenters, even surpassing the manufacturing cost during the datacenters' lifetime [88]. How to reduce the energy consumption in GPU clusters becomes an important research direction. In reality, tremendous energy is wasted on idle compute nodes. Then the critical goal is to reduce energy consumption from those idle nodes while satisfying users' demands. There are generally two techniques to conserve energy in datacenters. (1) Dynamic Voltage and Frequency Scaling (DVFS) adjusts the CPU & GPU voltage and frequency to save power. (2) Dynamic Resource Sleep (DRS) puts idle servers into deep sleep states (or simply turning them off) [88, 89]. *Slurm* also provides an

Algorithm 2 Cluster Energy Saving Node Control

Input: Nodes #: Current Active $\mathbf{C_A}$, Current Running $\mathbf{C_R}$, Request $\mathbf{R}$, History Series $\mathcal{H}$, Prediction Series $\mathcal{P}$

procedure JobArrivalCheck($\mathbf{C_A}$, $\mathbf{R}$)
  if $\mathbf{C_A} < \mathbf{R}$ **then** ▷*Lack nodes*
    NodesWakeUp($\mathbf{R} - \mathbf{C_A} + \sigma$) ▷$\sigma$*: Buffer nodes*

procedure PeriodicCheck($\mathcal{H}$, $\mathbf{C_A}$, $\mathbf{C_R}$, $\mathcal{P}$)
  $\mathcal{T}_H$ = RecentNodesTrend($\mathcal{H}$)
  $\mathcal{T}_P$ = FutureNodesTrend($\mathcal{P}$)
  if $\mathcal{T}_H \geq \xi_H$ **and** $\mathcal{T}_P \geq \xi_P$ **then** ▷ξ_H, ξ_P*: Thresholds*
    $\mathbf{C_A} \leftarrow$ DynamicResourceSleep($\mathbf{C_R} + \sigma$)

integrated power saving mechanism [17] to switch the nodes between power saving and normal operation modes based on their workloads.

We introduce a novel Cluster Energy Saving (CES) service to achieve energy conservation in GPU datacenters. Existing DRS-based technique [88] simply turns off and on the nodes based on *recent* and *current* workloads. However, frequent server boot-up can introduce extra energy overhead and delay. In order to reduce the unnecessary mode switch operations, we build a prediction model to estimate the *future* utilization trend in our clusters based on the historical node state logs. With the prediction, we can select and power off the optimal number of servers. This saves energy consumption while maintaining the cluster usability. As described in §2.3.1, the average utilization rate of our clusters ranges from 65% to 90%, and partial VCs are underutilized all the time. Hence, this strategy can bring huge financial gains for the datacenter.

Service Design The core of our service is the prediction model, which performs a time-series forecasting task. The biggest challenge of building this model is the lack of accurate information about the future behaviors of the cluster. Therefore, we extract time-related data such as trend, seasonality, and any deterministic event as the features for prediction. Specifically, we encode repetitive patterns (e.g., hour, day of the week, date) of running nodes to explore the periodic variations. Node trends are calculated as the average values and standard deviations of active nodes under different rolling window sizes. Moreover, binary holiday indicators and various time scale lags are also crucial for prediction. We use these features to build a model which can predict the number of running nodes in the future. We try different machine learning algorithms, and find the GBDT [86] model performs the best over other classical or deep learning models, e.g., ARIMA [90], Prophet [91], and LSTM [92]. So we choose GBDT in our CES service, which can achieve around 3.6% error rate (measured in Symmetric Mean Absolute Percentage Error (SMAPE) [93]) in the `Earth` cluster. This can give reliable and accurate advice for powering off nodes.

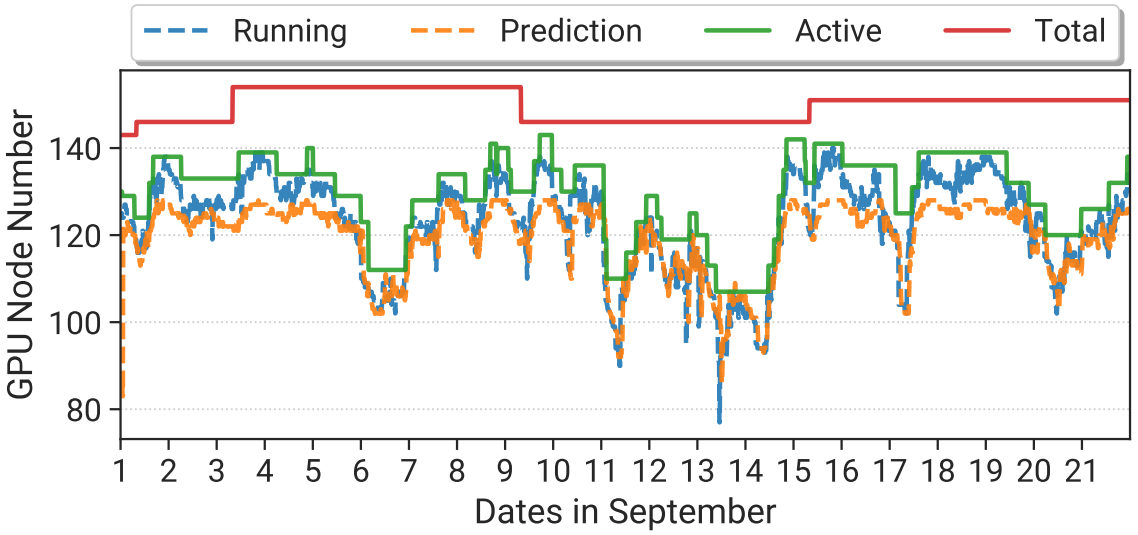


FIGURE 2.14: Numbers of compute nodes at different states in `Earth` from 1st September to 21st September (3 weeks).

With the prediction results, we can select the idle nodes and apply DRS to them. Algorithm 2 illustrates the procedure, where two steps are conducted to guarantee compute resources and conserve energy. (1) When a new job is submitted to the cluster, the service performs JOBARRIVALCHECK (line 1) to inspect whether the requested resources ($\mathbf{R}$) surpass the currently available ones ($\mathbf{C_A}$) in the cluster. If so, the service needs to wake up some nodes immediately via the Intelligent Platform Management Interface (IPMI). The quantity of nodes is determined by the resource gap ($\mathbf{R} - \mathbf{C_A}$) with an additional number σ to buffer unexpected future jobs. (2) Our service also calls PERIODICCHECK regularly (e.g., every 10 minutes) to check if extra nodes need to be powered off. The decision is made from both the historical and predicted future trends, to circumvent the incorrect DRS operations caused by the prediction error. Specifically, we call the function *RecentNodesTrend* to calculate the reduced number of active nodes during a fixed past period (e.g., one hour) from the historical data. We also call *FutureNodesTrend* to calculate the expected reduced number of active nodes (i.e., nodes can be powered off) in a fixed future period (typically 3 hours) based on the GBDT model prediction. If these two trends are larger than the thresholds (ξ_H, ξ_P), *DynamicResourceSleep* is called to reduce the active nodes ($\mathbf{C_A}$) to the current running number ($\mathbf{C_R}$) with a buffer σ.

In this work, we exploit the DRS technique on the selected nodes. Alternatively, we can also utilize *CPUfreq governor* and *nvidia-smi* [94] to adjust the frequency and voltage of CPUs & NVIDIA GPUs. According to [95], DVFS can not only improve the DL training performance by up to 33% but also save up to 23% energy consumption. Evaluations of these techniques will be our future work.

Evaluation We perform similar simulations based on the real-world traces from Helios. We select a period of 3 weeks (from 1st September to 21st September) in each cluster for evaluation, and the previous records are all used for training the prediction model. Figure 2.14 shows the GPU node behaviors in the `Earth` cluster. Two dashed lines denote the numbers of actual active nodes (blue) and our predictions (orange) respectively. We observe that our prediction can precisely reflect the actual trend with small estimate errors. The red solid line depicts the total number of GPU nodes. It is obvious to see a huge gap between the red and blue lines, representing a large number of energy-wasting idle nodes. With our CES service, a lot of idle nodes are powered off. The green solid line denotes the number of remaining active nodes. Qualitatively, we observe these nodes are just enough to meet users' demands while significantly reducing the energy waste.

Table 2.5 presents a quantitative analysis for each cluster. Our service can remarkably improve the node utilization, especially in `Earth` (13.0%) and `Uranus` (12.6%), as these clusters are relatively underutilized. Besides, our service only calls the function *NodesWakeUp* 1.1∼2.6 times a day in each cluster. In contrast, the vanilla DRS without considering nodes' future trends can incur an average of 34.1 *NodesWakeUp* operations a day, which causes much more turn-on energy overhead and job queue delay. Specifically, assuming each node takes 5 minutes to reboot, our service only affects 251 out of 198k jobs during 21 days. Nevertheless, vanilla DRS leads to nearly 6k jobs affected in the same traces.

Furthermore, we make a rough estimation about the reduction of energy consumption with CES using the data from Table 2.5. The power consumption of one single idle DGX-1 server is around 800 watts (obtained from NVIDIA BMC [96] by adding the input values of all PSUs). In addition, the cooling infrastructure typically consumes twice the energy as the servers in datacenter [97]. Hence, we can save over 1.65 million kilowatt hours of electricity annually across these 4 clusters. This can significantly reduce the operation cost.

We also evaluate our CSE service on the Philly trace. Microsoft provides per-minute statistics about GPU utilization on every node from the Ganglia monitoring system [24], from which we obtain a time sequence about the numbers of total and running GPU nodes. We select a period of 2 weeks (from 1st December to 14th December) for evaluation, and the previous data are all used for training the GBDT model. As shown in Figure 2.15, it is obvious that the change frequency of running nodes in Philly is lower than `Earth` and its cluster scale is over twice than `Earth`. Hence, the CES service only needs to take 0.5 times of nodes wake up action on average (Table 2.5), which has a negligible impact on the average JCT. Besides, more than 100 idle nodes can be powered off on average and the cluster node utilization rate is increased

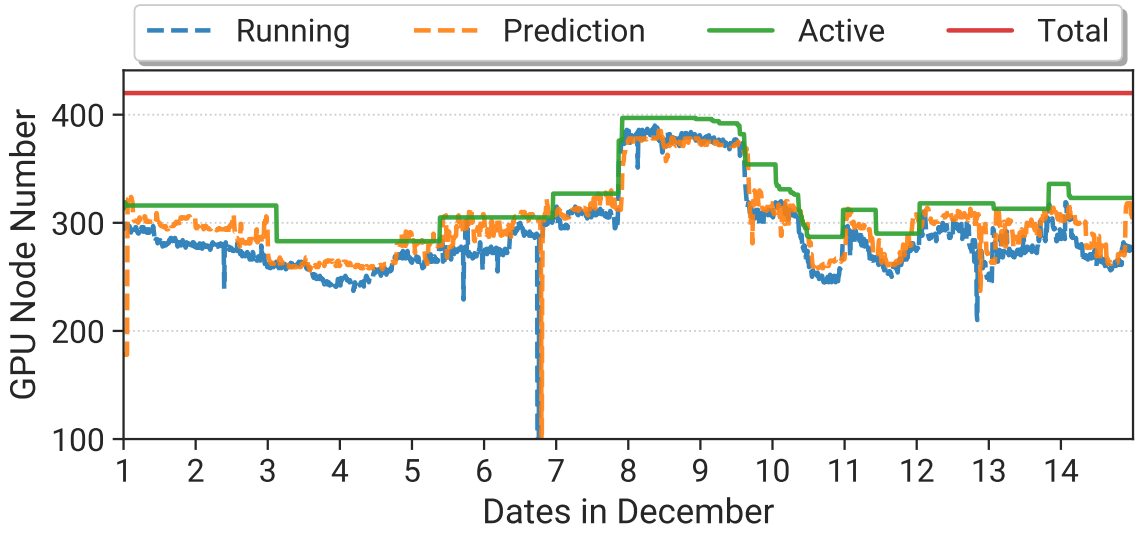


FIGURE 2.15: Numbers of compute nodes at different states in Philly from 1st December to 14th December (2 weeks).

TABLE 2.5: Performance of CES service in each cluster of Helios and Philly.

	Venus	**Earth**	**Saturn**	**Uranus**	**Philly**
Average # of DRS nodes	5.0	20.5	20.0	34.0	100.1
Average times of daily wake up	1.1	1.3	2.0	2.6	0.5
Average # of woken up nodes per time	6.7	6.7	7.1	9.7	27.1
Node utilization (Original)	92.7%	82.1%	90.2%	83.8%	69.0%
Node utilization (CES)	96.2%	95.1%	97.6%	96.4%	90.4%

from 69% to 90% (Table 2.5). This shows the CES service has strong applicability and generality for different clusters and workloads.

2.5 Related Work

Cluster characterization. A number of prior works conducted trace analysis for traditional CPU workloads, e.g., HPC systems [48, 49, 64, 66, 98–100], private clouds [59–61, 101]. Amvrosiadis et al. [48] presented an analysis of the private and HPC cluster traces about job characteristics, workload heterogeneity, and cluster utilization. They disclosed the correlations among job duration, behaviors, logical names and user behaviors. We further demonstrate the predictability of cluster resources and workloads information in GPU datacenters.

On the other hand, fewer studies focused on the analysis of characteristics of DL workloads in large-scale clusters. Jeon et al. [24] studied the DL job trace from the Microsoft cluster to identify the impact of the DL training job locality on the

GPU utilization as well as job failure reasons. Wang et al. [50] characterized various resource requirements and identified the performance bottlenecks of DL training jobs in Alibaba datacenter.

In comparison, our work provides a more comprehensive analysis about clusters, jobs, and user behaviors. Besides, wec provide more diversity to reflect the latest features of DL algorithms and technologies. These can deepen our understanding about the characteristics of DL clusters and jobs. We uncover seven interesting observations to inspire the new designs of efficient GPU schedulers.

Prediction-based scheduling. Prior knowledge of jobs can significantly facilitate the management of cluster resources. Modern cluster systems [16, 58, 76] expect users to provide estimates about their job duration, which may not be accurate as the job completion time is unexpected. Several methods were proposed to obtain more precise job information automatically for efficient scheduling. For instance, some schedulers predict the duration of new jobs based on the recurrent jobs [69, 82–84], or job structure knowledge [77–81]. These require the jobs to have explicit periodic patterns or known structures. For more general cases, some systems [71, 85, 101] predict the estimate from the history of relevant jobs.

For DL training job schedulers, Gandiva [18] leverages intra-job predictability to split GPU-time efficiently across multiple jobs to achieve lower latency. Optimus [43] adopts online fitting to predict model convergence during training. Tiresias [45] calculates the Gittins index as the job priority according to the duration distribution of previous jobs. AlloX [102] designs an estimator and profiles jobs in an online manner to determine the job duration. Different from these schedulers only for DL training, our QSSF scheduling service supports all types of DL jobs in the model developing pipeline. This better fits the industry requirements for production GPU clusters.

Energy efficiency for GPU clusters. Prior studies [89, 103] demonstrated that DRS does not affect the performance and efficiency of the cluster, which is consistent with our simulation results. Furthermore, a series of works [95, 104–109] indicated that GPU DVFS has huge benefits to save the energy consumption in GPU clusters. Some job scheduling algorithms [88, 110, 111] were then proposed based on these techniques to achieve energy efficiency in the clusters. However, they do not consider the impact of cluster future workloads, which can lead to more unnecessary server boot-up delays and extra energy overhead. In our CES service, we further enhance the benefits of the DRS by predicting the cluster usage in advance, which can get better efficiency.

2.6 Summary

In this chapter, we perform a large-scale analysis of the real-world DL job traces from four clusters in our datacenter. We present the characterizations of clusters, jobs and users, and identify seven implications, to guide us to design more efficient GPU cluster systems. Justified by the implication that the behaviors of jobs and clusters are predictable, we introduce a general-purpose GPU cluster management framework, which predicts the future behaviors of jobs and clusters to improve the resource utilization and job performance. As two case studies, we design a QSSF service to improve the average JCT by up to 6.5×, and a CES service to conserve annual power consumption of over 1.65 million kilowatt hours.

Helios traces are publicly available at https://github.com/S-Lab-System-Group/HeliosData. We expect they can benefit researchers in the design of GPU datacenter systems.

Chapter 3

Hydro: Surrogate-based Hyperparameter Tuning Service

3.1 Introduction

Over the years, we have witnessed the incredible performance and rapid popularity of Deep Learning (DL) across many domains, such as vision and speech. However, it is non-trivial to acquire a qualified DL model because its performance is highly sensitive to the *hyperparameters*, which control the training process and require to be set before training [112]. Poor hyperparameters result in training instability and inferior model quality. Conversely, well-tuned hyperparameters can significantly improve model performance. For instance, PyTorch [15] recently applies a new hyperparameter recipe on ResNet-50 [113] and achieves 80.9% ImageNet classification accuracy [114], which is 4.8% higher than the former version (76.1%). Besides, RoBERTa [115] also demonstrates the critical impact of hyperparameters on the performance of large language models. Accordingly, hyperparameter tuning becomes a common practice during DL model development.

Due to the high dimensionality of the search space, a hyperparameter tuning job typically contains a large group of *trials*, each with a unique configuration [32]. To accelerate the tuning process, tech companies and researchers build hyperparameter tuning systems as cloud services [3, 6, 116, 117] or standalone frameworks [7, 10, 32, 112, 118, 119] (Table 3.1). However, we argue that state-of-the-art tuning systems are still expensive and inefficient in practice, as they suffer from several fundamental problems:

- **Unacceptable cost of tuning large models**. The extraordinary performance of large foundation models (e.g., BERT [120], GPT-3 [22]) attracts wide downstream applications [11, 121, 122]. Meanwhile, the hyperparameter tuning demand for these

models increases rapidly. However, all of the existing tuning systems require training multiple trials using several times of resources, which is unaffordable for large models with billions of parameters. For example, training a SOTA language model PaLM [123] of Google takes over 6,000 TPU-v4 [124] for around 2 months. Performing a hyperparameter sweep on such model is intractable [125]. Consequently, hyperparameters of most large models are not well-tuned and can lead to subpar performance [115].

- **Inefficient hardware utilization**. Recent scheduling works [1, 18, 31, 36] report that GPUs are commonly underutilized in DL clusters due to massive training jobs involving mid- or small-scale models. Moreover, despite the growing trend of foundation models being employed in clusters, large-scale models often fail to fully utilize hardware resources due to the huge communication overhead and the presence of bubbles in the pipeline parallelism [126]. To improve resource utilization, some novel tuning systems incorporate features such as elastic training [112, 118, 119], GPU sharing [32], and inter-trial fusion [127]. However, these systems have certain limitations (§3.7) and often require substantial resources to explore trivial trials, which results in limited resources being contributed to the final model.
- **Agnostic to cluster-wide resources**. Hyperparameter tuning jobs are pervasive and occupy enormous resources in GPU clusters. As reported by Microsoft [24, 42], "approximately 90% of models require hyperparameter tuning, with each tuning job containing 75 trials in median." However, existing tuning systems only manage trials over the requested resources and lack interaction with cluster schedulers. Meanwhile, DL schedulers [18, 36, 45, 46, 56, 128, 129] also overlook the distinct characteristic of gradually diminishing hardware demand inherent in hyperparameter tuning jobs [112]. Consequently, the cluster encounters imbalanced resource problem: the active tuning jobs consistently occupy static resources, leaving some of them vacant, while the queued jobs are unable to request these idle resources from the scheduler. This leads to severe queuing delay, which is exacerbated when long-term large-scale model training jobs coexist and they occupy the majority of cluster resources.

To bridge these gaps, we design Hydro, a surrogate-based hyperparameter tuning service that optimizes tuning jobs in both the job-level and cluster-level granularities via automated model scaling, fusion and interleaving. The core design of Hydro derives from the following three insights. First, *it is feasible to search hyperparameters with a smaller model*. Instead of tuning hyperparameters directly on the target model, we find it is possible to tune a model with a much smaller surrogate model by applying a novel hyperparameter transfer theory [130, 131]. Second, *cross-model fusion can be used to improve resource utilization*. Since the scaled surrogate model is prone to incur GPU underutilization, we can utilize the model architecture consistency of different trials to fuse them into a single one, achieving much higher GPU utilization

and training throughput. Third, *ephemeral bubble resources in the datacenter can be leveraged for tuning.* Large model training jobs exist in the long term and occupy the majority of resources, which incurs the starvation of other jobs. We can leverage pipeline bubbles of large models to greatly extend the tuning job resources in an interleaving execution way, without hurting the training throughout of large models.

Incorporating the above insights, we build Hydro service to minimize the makespan of tuning workloads and improve the cluster-wide resource utilization. It consists of two key system components: (1) **Hydro Tuner** is the user interface that automatically generates surrogate models by *scaling* and *parametrization*. It optimizes tuning efficiency via *inter-trial* and *intra-trial fusion*, which involve combining multiple models into a single entity and subsequently performing compiler-based optimization. Besides, it efficiently orchestrates the tuning process with *adaptive fusion* and *eager transfer* mechanisms. (2) **Hydro Coordinator** is the datacenter interface that interacts with the scheduler to dynamically allocate resources and execute trials. It extends tuning resources by *interleaving training* with pipeline-enabled large model training tasks, effectively utilizing idle time intervals on each node known as *bubbles*, which are caused by the gaps between the forward and backward processing of micro-batches [126]. Besides, it improves resource utilization and cluster-wide performance by *heterogeneity-aware* allocation.

To extensively assess the performance of Hydro, we conduct evaluations across 6 models, such as GPT-3 XL [22] and ResNet [113]. Experiments on Hydro Tuner show that it substantially outperforms Ray by 8.7∼78.5× on makespan reduction with single-fidelity tuning algorithm, while obtaining better final model quality. Besides, our experiments on Hydro Coordinator demonstrate that interleaving with a large pipelined model can further extend the resource of tuning workload, without sacrificing the throughput of the large model.

Table 3.1 compares Hydro with existing tuning systems. To summarize, we make the following contributions:

★ We build a holistic system that automatically applies the novel hyperparameter transfer theory together with multiple system techniques to jointly improve the tuning efficiency.

★ We identify the opportunities for cluster-wide optimization in the datacenter, including squeezing bubble resources with interleaving and heterogeneity-aware trial allocation.

★ We demonstrate the excellent performance of surrogate-based hyperparameter tuning across general models.

TABLE 3.1: Comparison between Hydro and existing popular HPO systems: Google Vizier [3, 4], Amazon SageMaker [5, 6], Microsoft NNI [7, 8] and Anyscale Ray [9, 10]. ♦ denotes system cannot support the feature for many cases.

Features	Cloud Services		HPO Frameworks		**Hydro**
	Vizier	**SageMaker**	**NNI**	**Ray**	
Distributed Environment	✔	✔	✔	✔	✔
Elastic Training	♦	♦	♦	✔	✔
Auto Model Scaling	✘	✘	✘	✘	✔
Surrogate HP Transfer	✘	✘	✘	✘	✔
Inter-Trial Fusion	✘	✘	♦	✘	✔
Intra-Trial Fusion	✘	✘	✘	✘	✔
Heterogeneity Awareness	✘	✘	✘	✘	✔
Interleaving Training	✘	✘	✘	✘	✔

3.2 Background and Motivation

3.2.1 Hyperparameter Tuning

Hyperparameter Tuning (i.e., Hyperparameter Optimization, HPO) aims to identify the optimal hyperparameters via massive configuration exploration [112, 118]. In the general workflow of an HPO job: (1) the user designates a *search space* of hyperparameters to explore; (2) the tuning algorithm creates a set of training *trials* and each trial contains one unique hyperparameter configuration sampled from the search space; (3) the HPO system coordinates trials execution until the best hyperparameter configuration is found.

Existing research works typically optimize HPO efficiency from the tuning algorithm [132–140] or system [7, 32, 112, 118, 119, 127, 141, 142] aspects:

Algorithm taxonomy. Depending on whether to enable early stopping, tuning algorithms can be divided into two categories [143]: (1) *single-fidelity* (e.g, Random [144], Bayes [137]) algorithms require each trial to be fully trained, which is accurate but inefficient; (2) *multi-fidelity* (e.g., ASHA [132], BOHB [133]) algorithms stop unpromising trials via successive halving [145] or curve fitting [146] strategies. They are efficient but may miss the best hyperparameter configuration due to the use of "low-fidelity" evaluations. Hydro well supports both the single- and multi-fidelity algorithms.

System optimization. To further improve the tuning efficiency and resource utilization, there are two advanced techniques applied in state-of-the-art HPO systems: (1) *elastic training* dynamically allocates more GPU resources to the top performing

trials [112] and further adjusts the entire requested resources [56, 118, 119]. (2) *GPU sharing* (i.e., trial packing) allows multiple trials to share the GPU using the NVIDIA MPS [26] or MIG [25] technologies to achieve higher utilization [32]. Different from them, Hydro combines scaling, fusion and interleaving for ultimate efficiency.

3.2.2 Hyperparameter Transfer Theory

Recently, the remarkable success of foundation models has ignited a keen interest in exploring the relationship between model size and its optimal hyperparameter. *Scaling Laws* [147–149] empirically study the power-law functions of batch size and learning rate across varying model sizes. Nevertheless, the authors [147] candidly admit that only limited configurations are tested and the rule-of-thumb formulas break down when dealing with models that exceed one billion parameters.

Beyond heuristic exploration, some novel hyperparameter transfer strategies [130, 131, 150] are proposed by DL theorists. For simplicity, we call them ***parametrization***, the rule of how to adjust hyperparameters accordingly when models grow/shrink in both the width and depth. Different from existing HPO systems, Hydro enables automatic hyperparameter transfer based on parametrization. To make the obscure theory more accessible, we present a concise background overview of the underlying theory. [151] systematically builds a coherent theoretical framework for parameterization: the feature learning effect γ of a multilayer perceptron (MLP) model is proportional to

$$\gamma \equiv \frac{L}{w^{1-p}}, \ \ p \in [0, 1] \tag{3.1}$$

where w, L indicates the width and depth of the neural network respectively. For the purpose of simplicity, we assume that the numbers of the hidden-layer neurons are all of similar order, $w_1, w_2, ...w_{L-1} \sim w$. p is a metaparameter that interpolates different parametrization strategies into a unified framework, which is determined by inherent strategy. The objective is two-fold: first, to maintain a fixed γ that allows hyperparameters transfer across different model sizes, and second, to strive for a larger γ that facilitates better feature learning. To this end, there are two directions of parametrization:

(1) *Neural Tangent (NT) parametrization* ($p = 0$)[150]. It naturally arose from the study of infinite-wide neural network as Neural Tangent Kernel (NTK) [150, 152], which can keep γ fixed by scaling the depth along with the width as $L \sim w$. NTK is a kernel method to explain the evolution of neural networks during training, which is derived by applying the first-order Taylor expansion to linearized models. It belongs to the *lazy training* regime where the weights move very little [130], so that linearization approximately holds around the initial parameters and does not learn features, which

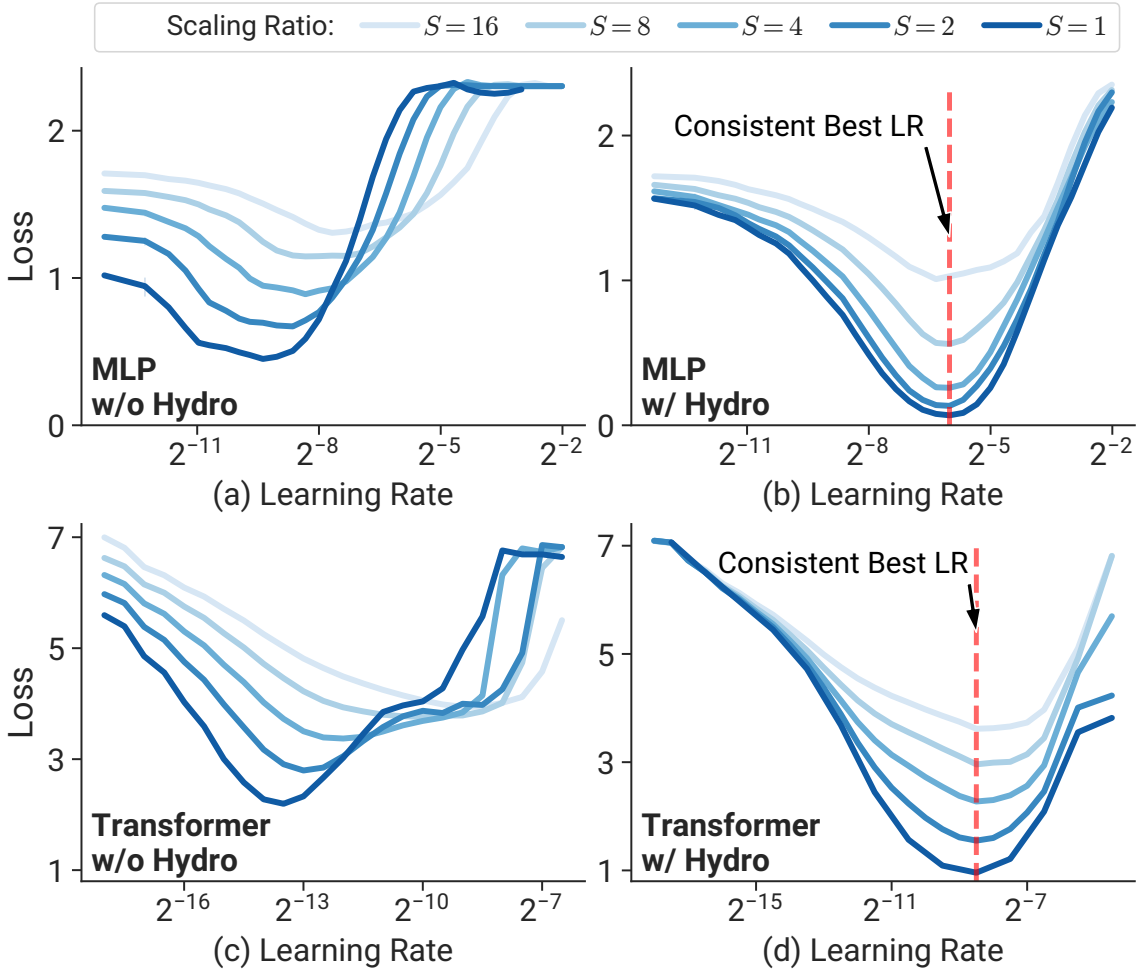


FIGURE 3.1: Effect of Hydro parametrization. The training loss against the learning rate on MLP (a, b) and Transformer (c, d) with different widths. S denotes the model scaling ratio.

is a fatal weakness of the NTK theory in practice. Moreover, NT parametrization does not make sense since the wider model does not always perform better in this context [131], which conflicts with common observations [147, 149].

(2) *Maximal Update (MU) parametrization* ($p = 1$)[130]. It generalizes the mean-field limit of the 1-hidden-layer case [153, 154] and should be the unique parametrization that retains the representation-learning capability (non-rigorously referred to *active training*, in contrast to lazy training of NT parametrization) for a large-scale neural network, which means training does not become trivial or stuck at the initialization in the large width limit. Colloquially, it is designed to solve the issue that the input layer is updated much more slowly than the output layer, and make all hidden activations update with the same speed in terms of width [131].

Hydro adopts the MU parametrization, which will be further elaborated in §3.4.1 and we refer readers to [130, 131, 155–159] for a comprehensive review of the theory.

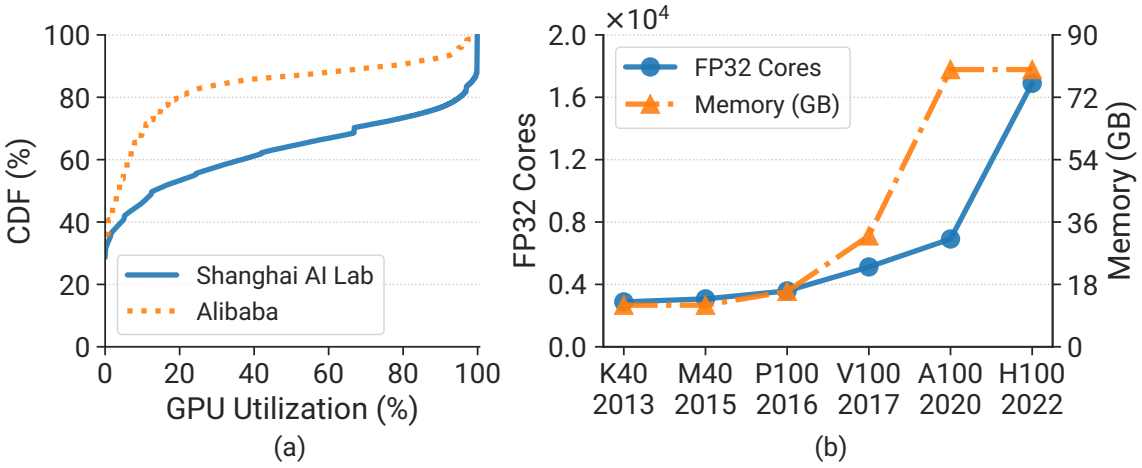


FIGURE 3.2: (a) GPU utilization distribution of one partition in our cluster and a GPU production cluster in Alibaba [1]. (b) Exponential growth of NVIDIA datacenter GPU capability. x-axis: GPU model name & release year.

3.2.3 Opportunities for Efficient Tuning

Lightweight surrogate-based tuning. Current HPO systems search hyperparameters directly on the *target model*, which is intuitive but inefficient. In contrast, Hydro makes it possible to tune a model with a much smaller *surrogate model* via applying a novel hyperparameter transfer technique (aforementioned in §3.2.2). For a clearer illustration of the surrogate-based tuning effect, we employ Hydro parametrization on two toy models and plot their converged training loss against a range of learning rates as shown in Figure 3.1. Specifically, the target MLP model contains two hidden layers (width=4096) and we train it with SGD on CIFAR-10. Similarly, the target Transformer model contains two TransformerEncoder layers (width=4096, i.e., d_{model}) and we train it with Adam on WikiText-2. Besides, we generate surrogate models with different scaling (shrinking) ratios S, and the smaller model is depicted by the lighter blue line. For instance, S=2 represents the model with width=2048. Obviously, the conventional training paradigm (Figure 3.1 (a, c)) cannot share the best hyperparameter across different sizes of models and there are orders of magnitude optimal learning rate shifts. However, Hydro parametrization (Figure 3.1 (b, d)) makes surrogate models stay approximately the same optimal learning rate as the target model, which implies the feasibility of surrogate-based tuning. Furthermore, Hydro parametrization can deliver better performance since both tuned MLP and Transformer achieve lower training loss than their counterparts. An intuitive explanation is that the learning rate of the conventional paradigm must tame logits' surge, but preceding layers do not learn appreciably. We perform a comprehensive evaluation of several models in §3.6.3 and demonstrate the superiority of Hydro.

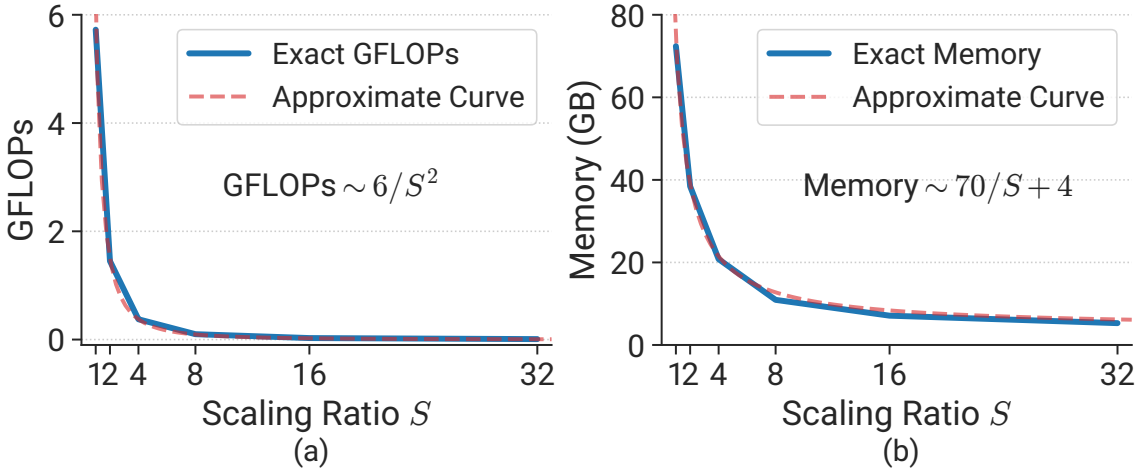


FIGURE 3.3: Model scaling effect of WideResNet-50. (a) Model GFLOPs (Giga Floating Point Operations). (b) GPU memory.

Fusion of numerous repetitive models. GPUs are commonly underutilized in DL clusters [1, 23, 31, 32, 36]. Figure 3.2 (a) plots the Cumulative Distribution Function (CDF) of one-week GPU utilization in one partition of our cluster, as well as an Alibaba trace [1] for reference. We find there are only 16% and 35% of GPUs achieving higher than 50% GPU utilization in Alibaba and Shanghai AI Laboratory respectively. This issue will be exacerbated if the Hydro surrogate-based tuning technique is applied. For instance, Figure 3.3 presents the model scaling effect of training WideResNet-50 on ImageNet, where GFLOPs follows approximately inverse-square (c_1/S^2) trend drop and memory footprint follows roughly $c_1/S + c_2$ trend decrease (c_i indicates constant). This implies model scaling can significantly reduce the computation overhead, but resources are more prone to be underutilized. To this end, inspired by JAX `vmap` function [160, 161], Hydro implements an *inter-trial fusion* mechanism to automatically combine multiple models into one. Operators of multiple trials can be fused owing to the property of HPO tasks: essential is a set of identical models (or with minor mutation). Compared with the conventional GPU sharing mechanism (e.g., MPS), Hydro can achieve higher training throughput, GPU utilization and lower memory footprint (Figure 3.8).

Cluster resource awareness. Although HPO jobs are pervasive in GPU datacenters, cluster schedulers typically regard them as general training workloads without any specific design. On the other hand, HPO systems [8–10] are cluster resource agnostic. This causes cluster-level inefficiency, such as long job queuing delay and low GPU utilization. However, the unique features of HPO jobs bring opportunities for more efficient tuning. (1) *Trial throughput insensitivity.* Unlike general DL jobs, HPO jobs are more tolerant to throughput slowdown of partial trials. Therefore, we can run more trials by leveraging ephemeral bubble resources of large language model training jobs, which are long-term existing in our datacenter (§3.5.1). (2) *Diminishing*

resource requirements. Multi-fidelity HPO jobs usually explore plenty of trials at the beginning and gradually decrease the search concurrency [112, 118, 119]. At the final stage, only a few trials are exploited. Therefore, we can not only reduce the total resource amount progressively, but also properly leverage the heterogeneous GPU resource (§3.5.2). With the rapid evolution of GPU computing capability as shown in Figure 3.2 (b), they become more prone to be underutilized for most small-scale trials [46]. Allocating trials to appropriate GPUs can significantly improve cluster-wide efficiency without hurting a single HPO job makespan.

3.3 Hydro Overview

Design principles & goals. For practical adoptions, Hydro follows three design principles: (a) *Automatic and simple*. Manually converting surrogate models is tedious and error-prone. Hence, the whole tuning workflow should be automated and easy to use, which requires minimum user code modification. (b) *Incentive and interference-free*. Although our system focuses on optimizing HPO jobs, it does not sacrifice other workload performance. Instead, it is altruistic and requires fewer resources than conventional systems, which benefits all cluster users. (c) *Modular and extensible*. Each component in Hydro can work independently to support more scenarios (e.g., cloud). Moreover, Hydro can be applied to general HPO tasks, and more tuning algorithms can be easily integrated. In addition, Hydro has two primary objectives: (1) minimizing the makespan of HPO workloads; (2) improving the cluster-wide resource utilization.

System architecture. Figure 3.4 depicts the architecture of the Hydro service. It consists of two key system components: ***Hydro Tuner*** (blue block) as a user interface to automatically generate surrogate models and optimize tuning trials, and ***Hydro Coordinator*** (purple block) for improving tuning efficiency and datacenter-level resource utilization. Each component contains several modules for different purposes. Specifically, there are three main modules in Hydro Tuner:

- *Model Shrinker*: to obtain surrogate models by automatically tracing, scaling and parametrization.
- *Trial Binder*: to better utilize accelerators by binding multiple trials and fusing internal operators.
- *Trial Planner*: to adaptively determine the tuning strategy based on the profiling information and intermediate results.

Additionally, Hydro Coordinator also includes three modules:

- *Bubble Squeezer*: to extend tuning workload resources by interleaving training with a pipeline-enabled large model.

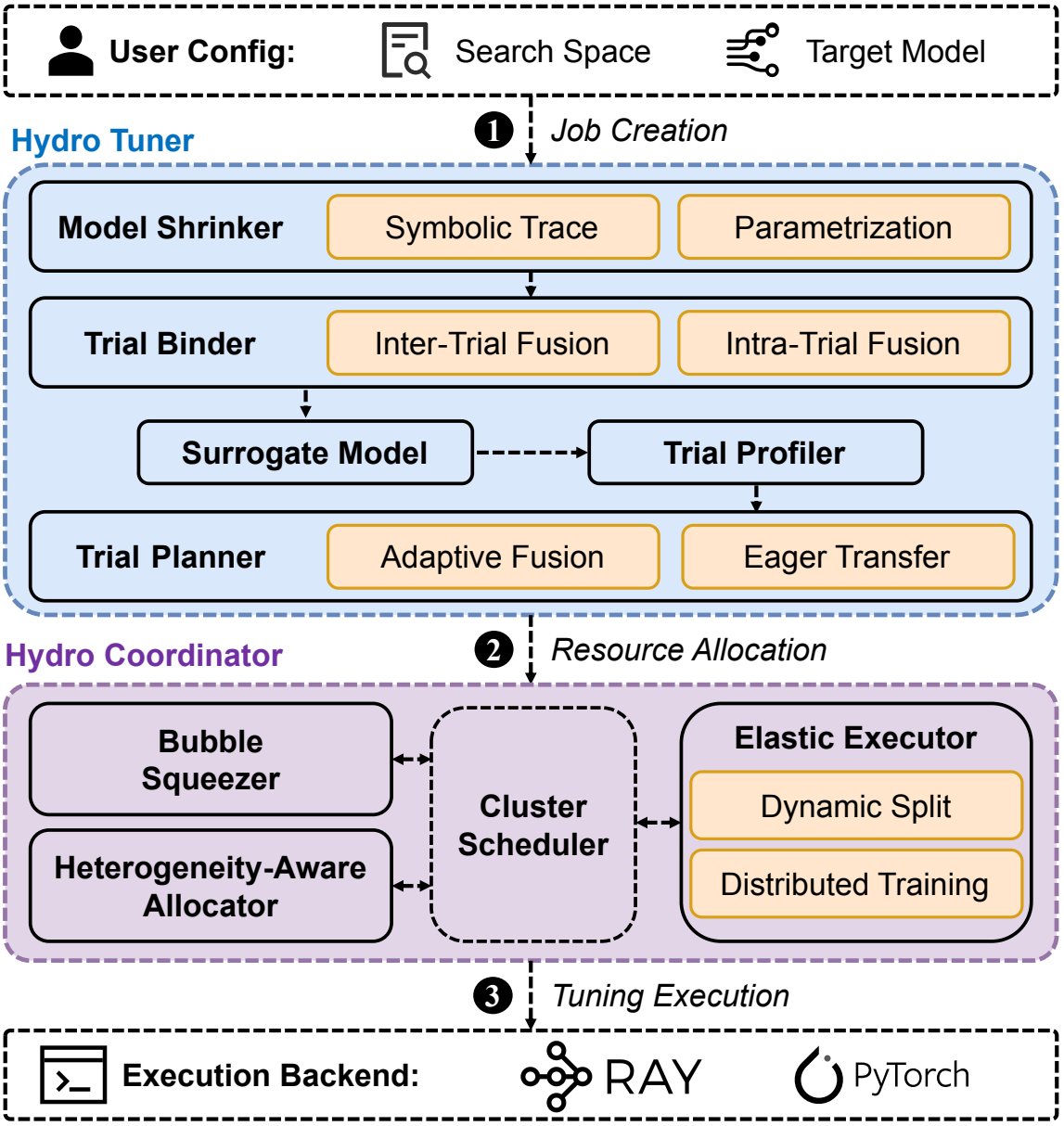


FIGURE 3.4: Overview of Hydro architecture and workflow.

- *Heterogeneity-Aware Allocator*: to improve resource utilization and cluster-wide performance by allocating proper accelerators on different tuning stages.
- *Elastic Executor*: to dynamically execute trials by splitting fused trials and enabling distributed training.

API Design. Hydro enables high-efficient surrogate-based hyperparameter tuning with a few lines in the developer's code, as shown in Figure 3.5. It follows the Ray Tune [10] API to define the search space and invoke the `fit()` function. To support Hydro functions, developers only require to wrap their model, dataloader and optimizer with the `prepare_xxx()` API (lines 6∼8). Hydro traces the whole function to control the trial execution, convert surrogate model, enable model fusion and elastic training.

```python
import ray, hydro
import hydro.train as ht

def train_func(config):
    # Wrap model, dataloader and optimizer
    model = ht.prepare_model(model)
    data_loader = ht.prepare_data_loader(data_loader)
    optimizer = ht.prepare_optimizer(SGD, lr=config["lr"])
    for _ in range(1):  # User defined training loop
        train_epoch(...)
        result = validate_epoch(...)
        ray.session.report(result)

search_space = {"lr": ray.qloguniform(1e-4, 1, 1e-4)}
trainer = hydro.Trainer(train_func)
tuner = hydro.Tuner(trainer, search_space, scaling_num=8)
results = tuner.fit()
```

FIGURE 3.5: A code example of how to use Hydro APIs to define the search space and perform hyperparameter tuning.

Tuning Workflow. The system workflow of Hydro is presented by black arrows in Figure 3.4. Specifically, when a developer wants to tune a model, she only needs to define the search space and invoke the Hydro APIs (❶). After job creation, Hydro Tuner automatically generates and optimizes surrogate models by scaling and fusion. Furthermore, it adopts *Trial Planner* to efficiently orchestrate the tuning process. Then Hydro Coordinator is responsible for contacting the cluster scheduler to dynamically allocate resources and execute trials (❷). It supports two novel mechanisms, which leverage ephemeral bubbles and heterogeneous resources to further improve datacenter efficiency. Finally, the tuning job is successfully scheduled and starts running, where Ray [9] and PyTorch [15] serve as the execution backend (❸). More details are introduced in the following sections (§3.4 & §3.5).

3.4 Hydro Tuner

Hydro Tuner is a core component of the Hydro service for job-level optimization. It consists of three modules: Model Shrinker, Trial Binder and Trial Planner.

3.4.1 Model Shrinker

Model Shrinker aims to obtain surrogate models by automatically tracing, scaling and parametrizing the target model. The upper part of Figure 3.6 depicts its workflow.

It first traces the target model and edits each layer's configuration to build a scaled model (①). To enable hyperparameter transfer, it then automatically parametrizes the scaled model by reinitializing the weight and adjusting the learning rate of each layer accordingly (②). Below we first summarize the MU parametrization theory that Hydro parametrization relies on, and then introduce how Hydro brings it into practice.

Maximal Update parametrization. As introduced in §3.2.2, Hydro employs the MU parametrization theory [130, 131] to search hyperparameters on a small surrogate model and transfer them to the large target model. The theory is built atop Tensor Programs [130, 131, 156–159], a unified theoretical framework that formulates the computation of common neural networks components as Gaussian Processes (GPs), including multilayer perceptrons (MLPs), recurrent neural networks (RNNs) (e.g., Long-Short Term Memory (LSTM) [92]), skip connections [113], convolutions [162] or graph convolutions [163, 164], pooling [162], batch normalization [165], layer normalization [166], and attention [167]. As a result, many practical models like ResNet [113] and Transformer [167] can be expressed as GPs and apply MU parametrization, since they inherently consist of these basic components.

➤ ***Theory assumption***. In contrast to prior works such as NTK [150] that necessitate unnatural conditions, MU parametrization only requires standard Gaussian initialization for the model, which is easily achievable in practice. In terms of data, MU parametrization requires i.i.d. samples, which is typically present in the same dataset. However, this requirement also limits its ability to support fine-tuning (§3.7).

➤ ***Key insight and mechanism***. The main idea of MU parametrization is: every activation vector has roughly i.i.d. coordinates at **any time** during training neural networks in the infinite-width limit. It aims to overcome the imbalanced per-layer learning speed issue in practice. To this end, MU parametrization performs layer-wise fine-grained adjustment, including per-layer initialization variance, learning rate and other optimizer-related hyperparameters (e.g., SGD momentum, Adam beta). Specifically, since the output layer is updated much faster than the input layer, MU parametrization suppresses the learning rate and initialization variance of output weights by w (width) times. In addition, for SGD-like optimizers (linear tensor update), the learning rate of input weights and all biases is multiplied by w. For Adam-like optimizers (non-linear tensor update, normalizes the gradient coordinate-wise), the learning rate of hidden weights is divided by w. Hence, MU parametrization ensures consistent magnitude updates for each layer during training regardless of its width so that hyperparameters can be transferred across models with different widths at any time (i.e., same converge speed across scaled models).

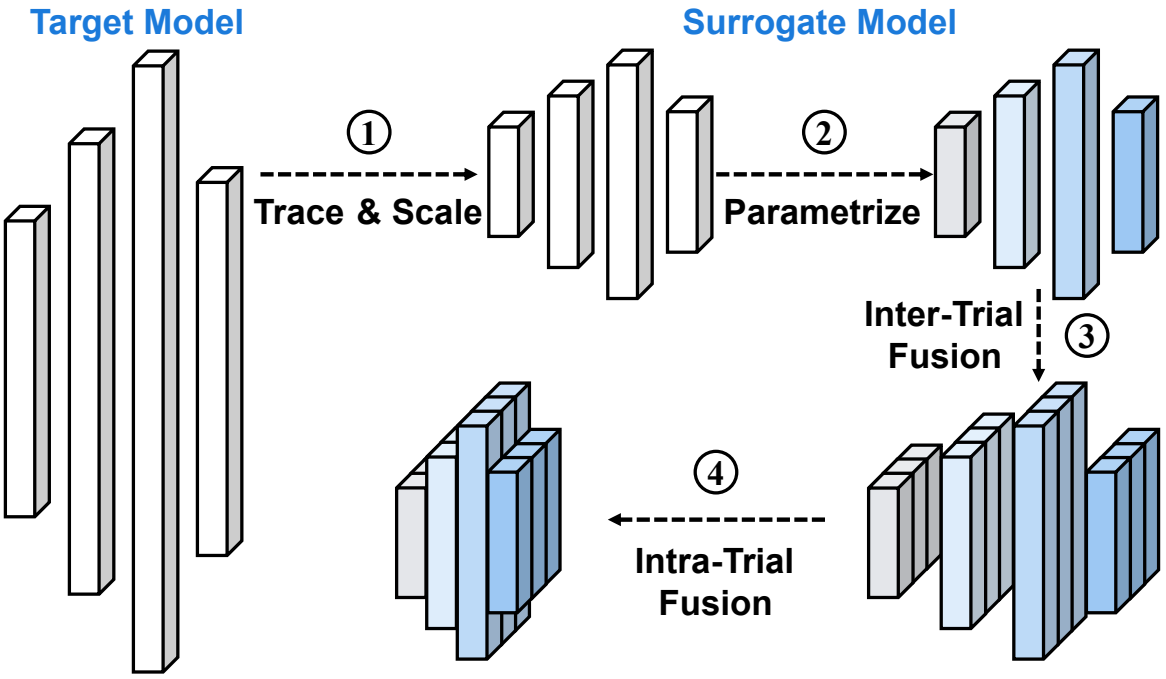


FIGURE 3.6: Illustration of Model Shrinker (①, ②) and Trial Binder (③, ④). The length of each bar represents layer width.

To summarize, in the large width limit, MU parametrization reveals that hyperparameters yielding lower training losses for narrower models also result in better performance for wider models through a specific transfer mechanism. Hydro leverages this effect to obtain better test accuracy efficiently via surrogate-based tuning, albeit without a rigorous theoretical guarantee for every model.

➤ ***Instructive example***. To provide a clearer explanation of why parametrization is necessary and how it operates, we recapitulate the key insights of [130] with an instructive example [131]. Consider a 1-hidden-layer linear model $f(x) = V^\top U x$ with scalar inputs and outputs, as well as w-width layer weights $V, U \in \mathbb{R}^{w\times 1}$. In common practice (e.g., Xavier initialization [168]), we initialize them with $V \sim \mathcal{N}(0, 1/w)$ and $U \sim \mathcal{N}(0, 1)$, which ensures $f(x) = \Theta(|x|)$ at initialization ($\Theta(\cdot)$ indicates asymptotically tight bound). After one step of SGD with learning rate 1, the new weights are $V' \leftarrow V + \theta U$ and $U' \leftarrow U + \theta V$, where θ is some scalar of size $\Theta(1)$ depending on the inputs, labels, and loss function. Then

$$\begin{aligned} f(x) &= V'^\top U' x \\ &= \left(V^\top U + \theta U^\top U + \theta V^\top V + \theta^2 U^\top V\right) x \end{aligned} \tag{3.2}$$

which will blow up with width w in the infinite limit because $U^\top U = \Theta(w)$ by Law of Large Numbers. In other word, it only allows $\mathcal{O}(1/w)$ learning rate so as to avoid float overflow, and yield kernel limits (§3.2.2). Consequently, it fails to perform feature learning (learning rate $\to 0$) to update weights after random initialization.

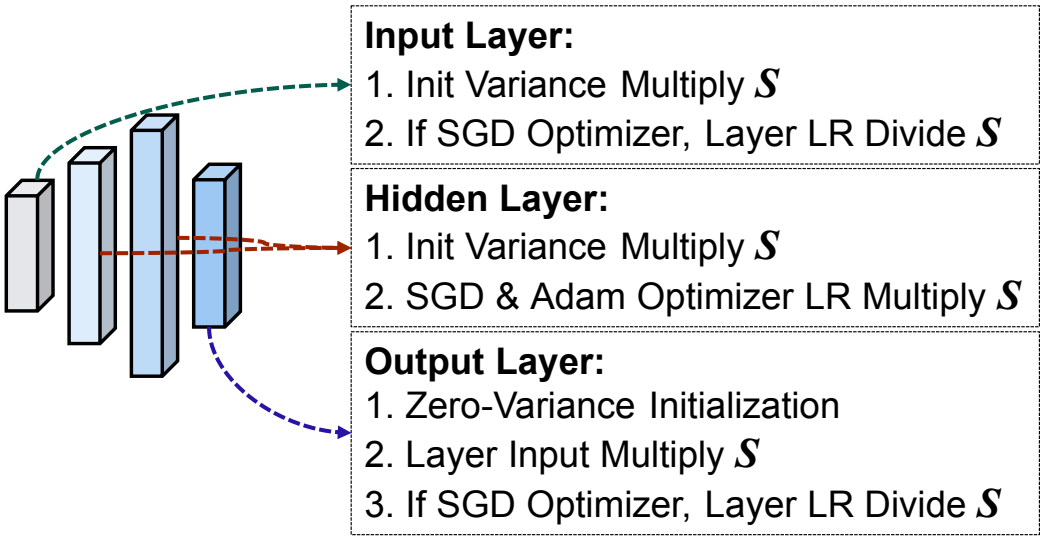


FIGURE 3.7: Hydro parametrization implementation. Illustration on a simple 4-layer model with SGD or Adam-like optimizer.

However, by applying maximal update parametrization, we have $V \sim \mathcal{N}(0, 1/w^2)$, $U \sim \mathcal{N}(0, 1)$, learning rates $\eta_V = 1/w$ and $\eta_U = w$. After one step of SGD, now we have

$$f(x) = \left(V^\top U + \theta w^{-1} U^\top U + \theta w V^\top V + \theta^2 U^\top V\right) x \tag{3.3}$$

and one can verify this is $\Theta(1)$ and remains bounded. In contrast to common practice, MU parametrization has $\Theta(1)$ learning rate and admits feature learning *maximally*, which allows every parameter to be updated maximally (in terms of scaling with width) without leading to float overflow.

➤ ***Heuristic adaptation.*** While Tensor Programs support more versatile model components (e.g., convolution), obtaining a closed-form solution for arbitrary models is infeasible. The efficacy of the MU parametrization has been rigorously demonstrated on a 2-hidden-layer MLP trained with SGD for multiple steps, and the proof can be readily extended to deeper MLPs [130]. For more general models in practice, some heuristic tricks are adopted to enhance their hyperparameter transferability. For example, Transformer [167] models require two additional operations in the self-attention: (1) scaling the attention logit by $1/d_k$ rather than $1/\sqrt{d_k}$, where d_k is the attention head size; (2) zero initialization on query layer q. We also empirically find that using a larger sequence length provides a better transfer effect for Transformer models. For models with some special components or architecture (e.g., MoE [169]), hyperparameters may not well transfer with MU parametrization alone. Hence, additional analysis and tailored adjustments may be required.

Hydro parametrization. It is arduous and error-prone to implement MU parametrization manually to generate a surrogate model. Developers are required to not only thoroughly understand the MU parametrization theory, but also manually adjust the model width, initialization function and learning rate layer by layer. Any incorrect adjustment may directly incur hyperparameter transfer failures. To this end, we implement Hydro parametrization, an automated and simplified parametrization strategy based on MU parametrization. We demonstrate the excellent effect of Hydro parametrization with visualized results in Figures 3.1 and 3.10.

For a clearer illustration, we present the Hydro parametrization process in Figure 3.7, which applies different strategies to the input, hidden and output layers. Developers only need to specify their desired *scaling ratio* S ($S = 8$ by default) and then Hydro will parametrize the model accordingly. Concretely, at the model initialization stage, we apply zero-variance initialization to the output layer instead of $1/w^2$, which will not be detrimental to performance and can remove this mismatch issue between the surrogate model and target model in the initial Gaussian process [131]. Moreover, we apply zero initialization to all biases, and weights as well as learning rate scaling strategies are annotated in the figure, which is invoked by the `prepare_optimizer` API to build a `hydro_optimizer`. Besides, we insert a `Multiply` layer in front of the output layer to scale its input by S.

Applicable Scope: Hydro parametrization works well for most ubiquitous hyperparameters that control model initialization and training, including learning rate, batchsize, lr_scheduler, momentum, etc. However, it has limited support on regularization-related hyperparameters, such as weight decay and dropout, because they naturally depend on both the model size and data size. Although parametrization cannot be applied to all hyperparameters, it is sufficient to achieve qualified performance in most cases. After most hyperparameters are tuned with the surrogate model, developers can further tune the regularization hyperparameters within a much smaller search space on the target model if needed. Moreover, we provide a comprehensive summary of additional limitations associated with Hydro parametrization in Section 3.7.

Trace and scale. Before performing the above parametrization, we need to first trace the target model and build a scaled model. Since there are various model definition styles in the PyTorch ecosystem, it is necessary to obtain a uniform and equivalent modality from disparate community model codes. We implement `HydroTracer` based on *torch.fx* [170], which allows developers to trace and edit the model. Specifically, we replace *call_function* nodes (e.g., *torch.nn.functional*) with the corresponding *call_module* nodes (e.g., *torch.nn*) for subsequent layer scaling and fusion. We apply different scaling rules to the input, output and hidden layers. For instance, we parse *nn.Linear* kwargs and modify both the *in_features* and *out_features* values by dividing S for hidden layers. In addition, we only scale the *out_features* of input and

in_features value of output layers. To handle the data-dependent control-flow, we use proxy nodes along with developer-provided concrete values to determine the execution flow [171]. According to our evaluation of notable models, including TorchVision [114] (e.g., ResNet [113], MobileNet [172], VGG [173]) and HuggingFace Transformers [174] (e.g., BERT [120], GPT [175], Swin [176]), developers can trace and scale these models with Hydro without modifying the code.

Correctness check. While Hydro has achieved automatic parameterization, there are still potential failures due to certain special model components that require heuristic adaptation as previously mentioned, as well as other corner cases that have not been considered. To this end, we further implement a safeguard mechanism to check the correctness of the parametrization and notify users whether they should use Hydro to prevent misleading hyperparameters and resource wastage. Firstly, Hydro performs a simple per-layer width check when scaling to avoid too narrow layers (e.g., only 1 neuron width for a Linear layer). Additionally, taking inspiration from gradient checking as a simple method for verifying the correctness of an autograd implementation, Hydro has a quick parameterization profiling stage that checks whether the average size (L1 value) of each activation vector is bounded to avoid possible parameterization failure based on [131]. It only lasts for very few steps at the beginning of the HPO job.

3.4.2 Trial Binder

Although Model Shrinker dramatically reduces the computation of each trial (Figure 3.3), it inevitably incurs the resource underutilization issue, which deteriorates small- or mid-size target models (e.g., deployed on edge devices). To address this problem, Trial Binder further optimizes surrogate models by binding multiple trials and fuses internal operators to better utilize accelerators. We illustrate its mechanism in the bottom part of Figure 3.6. It merges a set of fusible trials into a *HydroTrial* with grouped operators and optimizer (③). To further accelerate training, we automatically just-in-time (JIT) compile the fused (*inter*-) surrogate model to obtain fast and flexible fusion (*intra*-) kernels (④). Note that the last model with closer layer distance represents the reduced memory-bounded operations through intra-trial fusion.

Inter-trial fusion. There are plenty of trials with the same or similar model structure in a HPO job. Inspired by JAX `vmap` [160, 161], which returns a batched version of the target function by vectorizing each input along the axis specified, we can batch multiple trials into a single one by fusing their operators. Hydro implements an *inter-trial fusion* mechanism to automatically bind surrogate models. Specifically, Trial Binder traverses the traced surrogate model and replaces the *torch.nn* operators with grouped `hydro.nn` operators according to the predefined fusion rule and fusion count

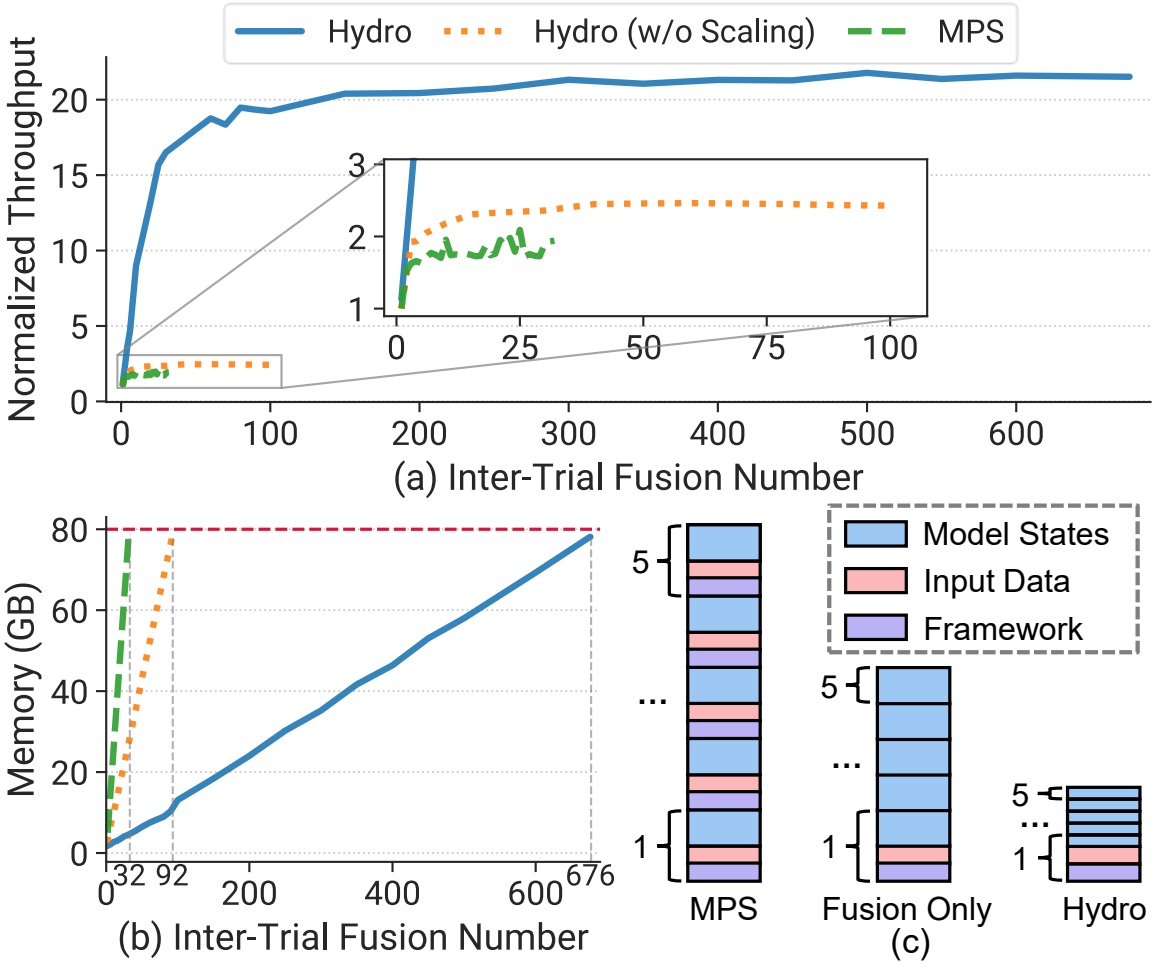


FIGURE 3.8: Inter-trial fusion effect on ResNet-18. (a) Accumulated throughout of fused surrogate model w.r.t the target model. (b) GPU memory footprint of different fusion counts. Red horizontal line denotes the A100 memory bound. (c) Schematic diagram of memory occupation detail of 5 models GPU sharing with MPS, Hydro and Fusion (w/o Scaling).

F determined by Trial Planner. `hydro.nn` provides mathematically equivalent implementations of batched original PyTorch operators based on HFTA [127]. For instance, `hydro.nn.Linear` is implemented atop *torch.baddbmm* (i.e., batch matrix-matrix product and add), which adds an additional dimension batch (i.e., F) compared with *torch.nn.Linear (addmm)*. Besides, for each `hydro.nn` operator, we reimplement the initialization function to support independent model-wise Hydro parametrization and realize the defusion mechanism to extract a specific sub-model. Additionally, `hydro_optimizer` and `hydro_lr_scheduler` are designed to support both the model fusion and parametrization simultaneously. These are performed automatically, and developers typically do not need to understand the rationale and modify codes.

Figure 3.8 plots the extraordinary effect of integrating model scaling with inter-trial fusion on ResNet-18 ($S = 8$), tested on CIFAR-10 with batchsize=256. It is evident that Hydro is capable of concurrently training impressive 676 models on a single A100

GPU. Compared with the conventional GPU sharing mechanism MPS [26] (MIG [25] has similar performance), Hydro achieves over 10× training throughput improvement and over 20× GPU memory conservation. If we directly apply inter-trial fusion to the target model (without scaling), the throughput improvement is relatively much limited. Furthermore, we provide an intuitive interpretation of how memory footprint reduction occurs in Figure 3.8 (c). The model states (blue blocks) encompass all aspects associated with model training such as model weights, gradients, activations, and optimizer states [177]. MPS has repetitive memory overheads incurred by CUDA context of DL framework (purple blocks) and independent data loading (pink blocks). In contrast, Hydro avoids such redundancy and further reduces model-related memory footprint. Note that here we only compare with vanilla training paradigm without considering more advanced memory optimization techniques like Salus [31]. Moreover, beyond the better GPU utilization and higher throughput, inter-trial fusion also alleviates the I/O pressure owing to the accompanied data-loading fusion.

Lazy intra-trial fusion. Hydro supports automatic model fusion to further accelerate training based on the `nvFuser` [178] compiler backend. Although plenty of previous works [179–181] demonstrate that operator fusion can improve training throughput via better memory locality, it does not always bring benefits to HPO workloads due to its high compiling overhead. For instance, `nvFuser` [178] takes approximately 2-epoch time to compile a ResNet-18 model to deliver around 10% speedup per epoch, which means a trial needs to run at least 20 epochs to avoid slowdown. However, most trials will end up in a few epochs for multi-fidelity tuning algorithms. To this end, Hydro apathetically adopts the intra-trial fusion. For simplicity, Hydro currently only applies to trials with *deterministic* training steps, such as all `HydroTrials` when applying single-fidelity tuning algorithms and the trial that trains the target model with transferred hyperparameters.

3.4.3 Trial Planner

Trial Planner is the key module that interacts with the tuning algorithm and trial executor. We introduce two mechanisms that improve the surrogate-based tuning efficiency.

Adaptive fusion. The trial count and resource amount vary significantly across different HPO jobs. Hence, the fusion count F of each `HydroTrial` should be adaptively determined to achieve the desired performance. Hydro contains the following steps to fuse trials and assign GPUs: (1) Trial Planner invokes the tuning algorithm to generate a set of hyperparameter configurations (trials). (2) Since inter-trial fusion requires trials with the same operator shapes, we split them into different *trial groups* according to their batchsizes. (3) Based on the linear growth of GPU memory shown

in Figure 3.8 (b), we can profile the trials with $F = 1$ and $F = 2$ for each *trial group* and estimate the upper bound of the fusion count F_{max}. (4) Hydro assigns all available GPUs to each *trial group* according to group's weight, which equals to $B \times N$ (denoted as the product of batchsize and trial count of the group). (5) Each *trial group* evenly distributes trials based on the group GPU amount and F_{max}, and Hydro fuses them as a `HydroTrial` on each GPU. In this way, Hydro can leverage as many GPUs as possible and achieve the optimal global throughput.

Eager transfer. As the HPO job progresses, more and more trials terminate and the degree of the parallelism gradually decreases, resulting in underutilized or idle resources. On the other hand, the best hyperparameter configuration sometimes appears in the early stage. Therefore, instead of training the target model after all the surrogate-based tuning trials are done, we can eagerly transfer the intermediate best hyperparameters and leverage vacated resources to validate the configuration on the target model. Hydro records all evaluated hyperparameters and schedules a *TargetTrial* for the target model training when 50% (customizable) of the surrogate-based tuning trials are done and there exist idle resources. If a better hyperparameter is searched, Hydro terminates the on-going *TargetTrial* or starts a new *TargetTrial* depending on the resource utilization. This mechanism efficiently shortens the job makespan and improves the resource utilization.

3.5 Hydro Coordinator

Hydro Coordinator focuses on cluster-level optimization. It consists of three modules: Bubble Squeezer, Heterogeneity-Aware Allocator and Elastic Executor. It is important to highlight that the first two modules are tailored for specific cluster scenarios. Specifically, Bubble Squeezer can only be activated when a pipeline-enabled foundation model pretraining job is running within the cluster. The Heterogeneity-Aware Allocator is meticulously designed to better leverage multiple generations of GPUs coexisting in the cluster.

3.5.1 Bubble Squeezer

In addition to HPO jobs, there are many kinds of workloads that coexist in the GPU datacenter, such as inference, debugging and large-scale distributed training jobs [2, 23, 24]. With the rapid popularity of foundation models (e.g., GPT-3 [22]) in recent years, some large model pretraining workloads exist in our datacenter in the long term. As complained by many users, the majority of machines are occupied by large model training jobs that usually last for days to weeks, which incurs the

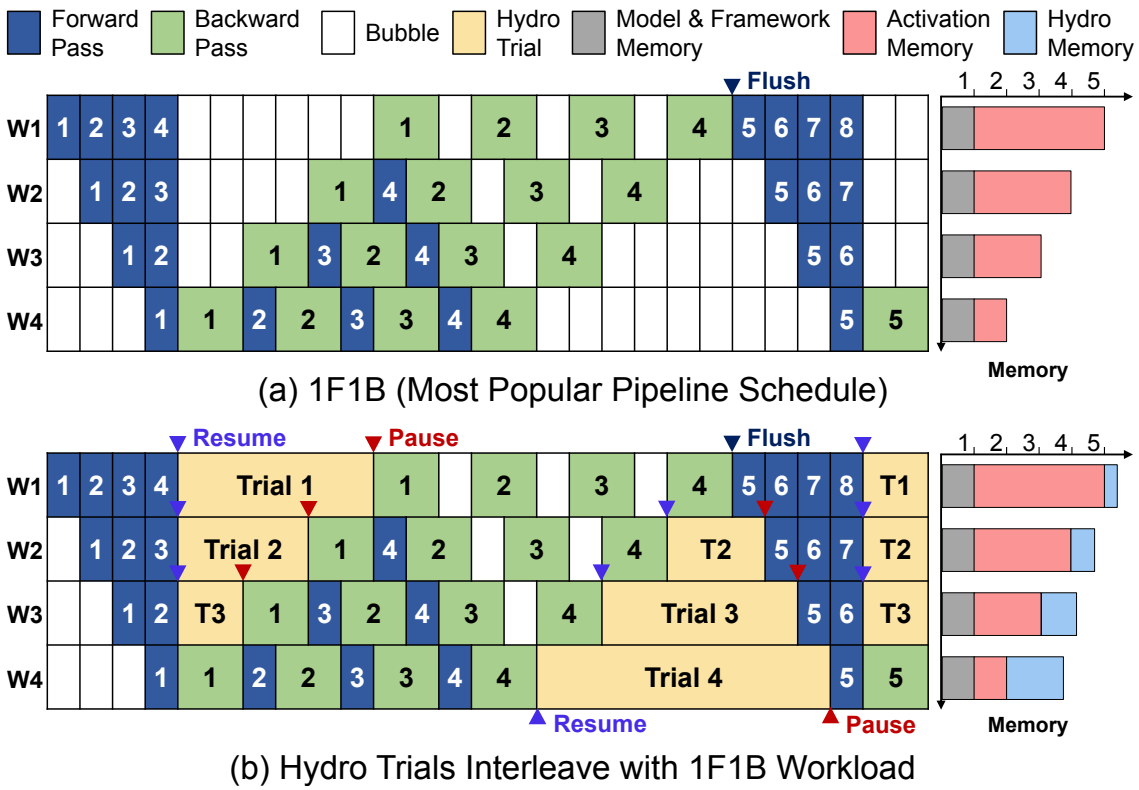


FIGURE 3.9: Illustration of (a) 1F1B Pipeline and (b) Hydro Bubble Squeezer, with four pipeline stages and four micro-batches. Note the right-side memory diagrams can only reflect the relative relation of the *same* color blocks across workers.

starvation of other jobs. Additionally, the pipeline parallelism [182, 183] is usually adopted to support a larger model by splitting it into several stages and placing them across multiple workers. However, bubbles inherently exist in the synchronous pipeline parallelism [126], such as the commonly used 1F1B [184, 185] strategy. Besides, the imbalance peak memory issue (Figure 3.9) between different pipeline stages further exacerbates the resource inefficiency [186].

Hydro designs Bubble Squeezer, which leverages bubbles to greatly extend the tuning job resources in an interleaving execution way, almost without hurting the training throughout of large models. `HydroTrials` are perfectly suitable for the bubble interleaving execution due to the following unique features: (1) *Throughput insensitivity.* Unlike general DL training jobs, tuning jobs are more tolerant of the slowdown of partial trials. This inspires us to squeeze the spare resources of the bubbles and execute trials in a pause-and-resume way. (2) *Deterministic resource pattern.* General small-scale workloads (e.g., debugging) have unknown and unpredictable resource requirements. However, Hydro profiles and records the resource consumption of `HydroTrials`, mitigating the potential out-of-memory (OOM) issues if they are colocated with large models. (3) *Elastic trial size.* Based on Model Shrinker, the scaled model has a much smaller memory footprint (Figure 3.3) than the original

model, which means we typically do not need to swap out its GPU memory during colocation. Besides, we can dynamically adjust the trial fusion count according to the remaining GPU memory with Trial Binder.

To clearly illustrate how Bubble Squeezer works, we first introduce the 1F1B pipeline parallelism in Figure 3.9 (a). It transfers intermediate activations of the partial model at the forward and backward passes between different workers using point-to-point communication [187], thus each worker cannot continuously utilize the GPU. For Worker 1, after the forward pass of the last micro-batch (blue block 4), it has to wait for the backward pass of the first micro-batch (green block 1), leaving GPU idle for a long time. Other workers also present similar bubble patterns but occupy less GPU memory since fewer activations of micro-batch needed to store.

In Figure 3.9 (b), Hydro interleaves four `HydroTrials` of different sizes with the large model training workload. Each trial executes in a pause-and-resume paradigm to squeeze the bubbles. Since Hydro Tuner has traced and canonicated each layer with `hydro.nn`, we further register hooks on each module of the trial to support on-demand pause and resumption in the forward and backward passes of each layer. When a large model training job exists, Hydro coordinates with datacenter scheduler to acquire more GPUs from this model and tags them as *ephemeral* resources. For the large model, we also implement a corresponding hook inside its training framework (i.e., DeepSpeed [177]) to report its training progress and resource consumption. Each worker executes its corresponding pipeline under DeepSpeed's pipeline parallelism. Therefore, we implement a fine-grained synchronization mechanism to guarantee that `HydroTrials` only could be executed within the bubbles, by intercepting the status of the CUDA streams of the NCCL kernels. Hydro can further adjust the fusion count to adaptively fit in the remaining memory and improve GPU utilization. At the beginning and end of the bubble of large model training, we control the resumption and pause of trial model training by Linux signals. The fine-grained suspend-resume control eliminates the performance interference caused by CUDA kernels running simultaneously.

In general, the effectiveness of Bubble Squeezer varies depending on multiple factors, and we present the scenarios where it works best. Regarding the HPO job aspect, Hydro is more effective when using (1) *multi-fidelity tuning algorithms* because they allow most trials to be terminated in a few epochs using the ephemeral resources and execute immediate top trials on exclusive resources to avoid possible blocking caused by interleaving slowdown. In addition, (2) *models with fewer layers* are preferred as they are prone to complete the entire iteration within the bubble time and require relatively less memory to support a higher fusion number. As for pipelined large model aspect, Hydro can achieve better performance when the pretraining job has (3) *more pipeline stages across more servers*, which implies a higher bubble ratio and more

ephemeral resources. A large model pretraining job typically can support multiple different HPO jobs interleaving simultaneously and accelerate dozens, even hundreds of HPO jobs (depending on its resources and duration scale) during its pretraining process. In addition, there may be cases where some scaled models are still too large to be allocated on any GPU of the pretraining model. Due to the high memory swap overhead in our scenario, Hydro does not support offloading techniques like Bamboo [126]. As a result, Bubble Squeezer is unable to support such models.

3.5.2 Heterogeneity-Aware Allocator

HPO workloads generally have diminishing resource requirements [112]. They usually explore plenty of trials at the beginning and gradually decrease the search concurrency. At the final stage, only a few trials are exploited. Hence, tuning with fixed GPU resources can lead to underutilization. Existing HPO systems [118, 119] support autoscaling to dynamically adjust the tuning resources. However, they do not consider the GPU heterogeneity in the datacenter.

Inspired by Gavel [46], a novel heterogeneity-aware cluster scheduler for general DL jobs, we design a resource allocator to allocate appropriate GPUs to trials, which can improve the cluster-wide efficiency without sacrificing the job makespan. Hydro supports both resource autoscaling and heterogeneity-aware allocation. Specifically, if there is any node or GPU idle for over 1 minute (customizable), Hydro will interact with the cluster scheduler to release the resource. Other affiliated resources like CPU will also be released as a bundle. Additionally, Hydro creates *TargetTrial* with the eager transfer mechanism and makes the target model training process well hidden inside the tuning time. Since the *TargetTrial* typically trains alone without fusion, it may not be able to fully utilize the GPU resources. So Hydro will place it on an GPU of old version (e.g., V100) if its `SM Activity` rate (measured by NVIDIA DCGM [188]) is lower than 50% (customizable). Similar action will be applied to surrogate models if their allocated resources are underutilized and there exist other HPO jobs pending in our service queue.

3.5.3 Elastic Executor

Elastic Executor is designed to improve the job efficiency by leveraging all assigned GPU resources. It supports two elastic mechanisms: (1) dynamic split and (2) automated distributed training. Specifically, when an idle GPU emerges, the *fused* `HydroTrial` will not directly increase its GPU count by conventional distributed training. Instead, Hydro will evenly split this `HydroTrial` into multiple `HydroTrials` and exclusively place them on the idle GPUs to reduce the communication overhead.

TABLE 3.2: Summary of (1) workloads used in the evaluation and (2) single-fidelity tuning improvements over Ray. *Model Quality*: ppl indicates perplexity (the lower the better) and acc denotes accuracy (the higher the better). * For XL tasks, we estimate the time cost of Ray based on simulation and use the official hyperparameter setting as the model quality baseline.

Task	Search Space	Model	Dataset	Optimizer	# of GPU	# of Trial	Avg. Time Reduction	Avg. Quality Difference	Size
Language Modeling	`lr`: $\mathbf{U}_{Q\log}(10^{-5}, 10^{-1}, 10^{-5})$ `gamma`: $\mathbf{U}_{Q}(0.01, 0.9, 0.01)$	GPT-3 XL [22]	OpenWebText [189]	AdamW	128	100	78.5 ×	−0.48 ppl	XL*
		Transformer [167]	WikiText-103 [190]	Adam	8	200	8.7 ×	−0.15 ppl	M
Image Classification	`lr`: $\mathbf{U}_{Q\log}(10^{-4}, 1.0, 10^{-4})$ `momentum`: $\mathbf{U}_{Q}(0.5, 0.999, 10^{-3})$ `batchsize`: [128, 256, 512] `gamma`: $\mathbf{U}_{Q}(0.01, 0.9, 0.01)$	WideResNet-50 [191]	ImageNet [192]	SGD	32	200	20.3 ×	+1.18% acc	XL*
		MobileNetV3 Large [172]	Flowers102 [193]	Adam	16	500	12.3 ×	+0.05% acc	L
		VGG-11 [173]	CIFAR-100 [194]	SGD	8	500	10.8 ×	+0.09% acc	M
		ResNet-18 [113]	CIFAR-10 [194]	SGD	8	1000	16.2 ×	+0.02% acc	M

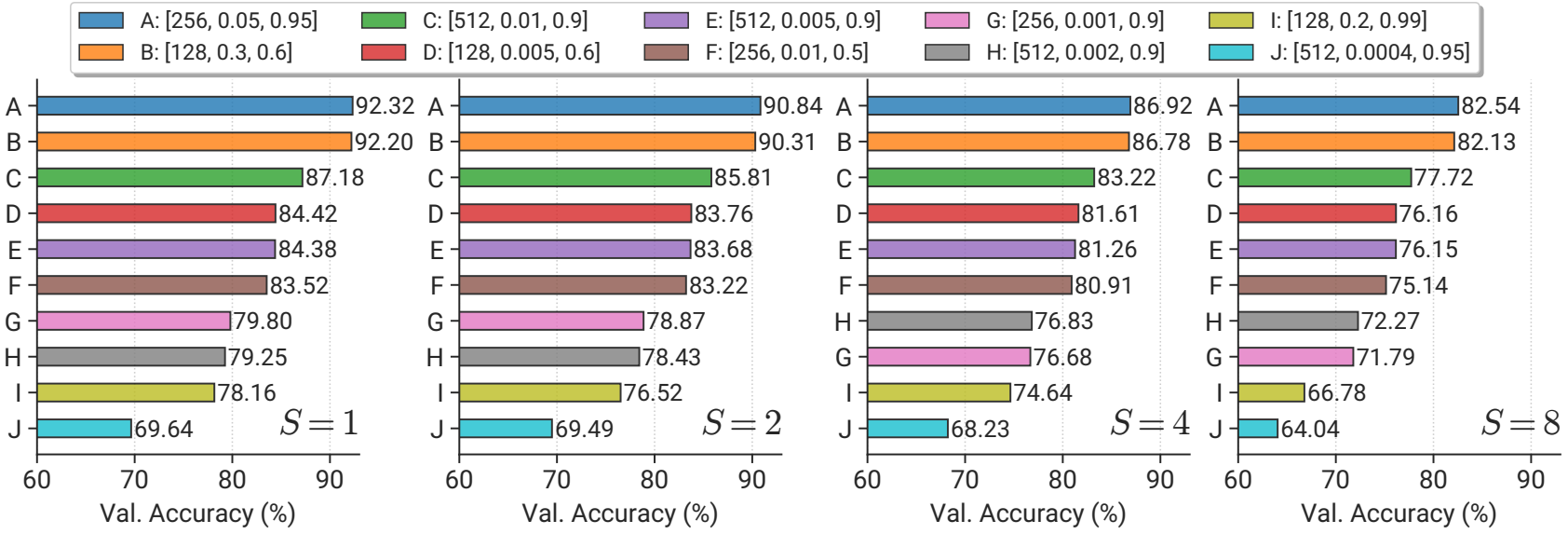


FIGURE 3.10: Hydro Tuner mechanisms validation. (a)∼(d) *Scaling validation*: randomly select 10 hyperparameter sets ([`batchsize`, `lr`, `momentum`]) to visualize the transfer effect of multi-dimensional hyperparameters across different scaling ratios $S = 1, 2, 4, 8$ on ResNet-18.

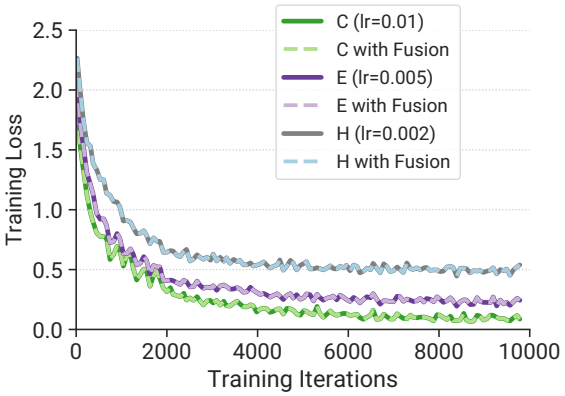


FIGURE 3.11: Hydro Tuner mechanisms validation. *Fusion validation*: loss curves of the standard model (solid line) and inter-trial fused model (dash line).

Furthermore, since the memory footprint of some large models is high even though scaled, Hydro supports two types of elastic strategies for *unfused* surrogate models: (a) *Evenly distribute*: allocating idle GPU resources to all unfused surrogate models evenly. (b) *Performance-aware* (default): allocating idle GPU resources to the top performing trial. For the target model, Hydro automatically increases the number of workers to enable distributed training.

3.6 Evaluation

Hydro is implemented on top of Ray [9, 10] with about 12K LoC. For Hydro Tuner, Model Shrinker relies on torch.fx [170] and mup [131], while Trial Binder is built with HFTA [127] and nvFuser [178]. As for Hydro Coordinator, we modify DeepSpeed [177] to further support Bubble Squeezer and validate the interleaving execution as a prototype. And the Elastic Executor is based on Ray Train as well as PyTorch FSDP [195].

We evaluate Hydro Tuner and Hydro Coordinator independently for a fair comparison. Our experiment search space does not include weight decay because Hydro is unable to transfer regularization hyperparameters, but it is sufficient to achieve qualified performance without tuning it.

3.6.1 Experiment Setup

Testbed. We conduct our experiments on a GPU datacenter of Shanghai AI Laboratory. Each node has 8 NVIDIA A100 80GB GPUs, 2 AMD EPYC 7742 CPUs (128 cores) [196] and 1TB memory. GPUs are interconnected to each other by NVLink and NVSwitch [28], and inter-node communication is achieved via NVIDIA Mellanox

200Gbps HDR InfiniBand [29]. All the experiments are conducted on A100 GPUs, unless explicitly stated in §3.6.5.

Workloads and search spaces. We evaluate Hydro tuning performance over six popular CV/NLP models, as listed in Table 3.2. Specifically, *GPT-3 XL* is a large language model architecture belonging to GPT-3 family. It contains 1.3B parameters and we use an open source implementation by GPT-Neo [125, 197]. We further enable mixed precision training for *WideResNet-50* and two language modeling tasks. For the dataset, we crop Flowers102 into 224×224 images, whose input size is the same as ImageNet. And we swap its train and test dataset split to get a larger training dataset to make it similar to more general jobs. Moreover, we denote single-node tasks as M-size, and distributed tuning tasks as L/XL-size.

We adopt three kinds of optimizers for above models, including SGD [198], Adam [199], and AdamW [200]. We use StepLR to decay the learning rate (`lr`) of each parameter group by `gamma` at every fixed step for all tasks. Additionally, we design two groups of search spaces for CV and NLP tasks respectively (Table 3.2), where $\mathbf{U}_Q(lower, upper, q)$ represents uniformly sampling a quantized (increment of q) float value between *lower* and *upper*. Similarly, $\mathbf{U}_{Q\,\log}$ uniformly samples in different orders of magnitude. Note that the search space of *MobileNetV3 Large* excludes `momentum` due to the incompatibility of Adam.

Tuning algorithms. Hydro supports multiple popular single-fidelity and multi-fidelity tuning algorithms, such as Random [144], HyperBand [135], ASHA [132]. Since our work focuses on system aspect optimization instead of tuning algorithms, we select two representative tuning algorithms in our evaluation: (1) *Random* (single-fidelity): fully evaluates each randomly generated trial; (2) *ASHA* (multi-fidelity): eliminates unpromising trials via asynchronous successive halving strategy. They are common hyperparameter tuning paradigms in practice. Besides, their asynchronous and prior-independent nature makes them more suitable for large-scale distributed tuning with numerous trials [112].

Baselines. We consider the following two systems as baseline: (1) *Ray* [9, 10]: performs HPO with the vanilla Ray Tune library; (2) *Ray+ES*: applies two advanced techniques in Ray Tune (*Elastic training* and *GPU Sharing*). Our implementation of *Ray+ES* refers to HyperSched [112] and Fluid [32]. Specifically, we place multiple trials on the same GPU using NVIDIA MPS [26] and allocate more GPU resources to the top performing trials if idle GPUs are available. We do not employ A100 MIG [25] sharing due to its similar performance with MPS but less flexibility [127]. Additionally, since existing popular HPO systems (Table 3.1) mainly differ in the application scenario and API design, and their system performance on the same tuning algorithm is similar, the Ray-based systems are sufficient for representing SOTA.

TABLE 3.3: Summary of multi-fidelity tuning improvements.

Model	# of GPU	# of Trial	Avg. Time Improvement	Avg. Quality Difference
GPT-3 XL	64	100	33.4 ×	−0.43 ppl
Transformer	4	200	5.8 ×	−0.09 ppl
WideResNet-50	16	200	9.7 ×	+0.87% acc
MobileNetV3 Large	8	500	8.0 ×	+0.08% acc
VGG-11	4	500	9.4 ×	+0.19% acc
ResNet-18	4	1000	14.5 ×	+0.05% acc

3.6.2 Surrogate-based Tuning Validation

Before performing end-to-end evaluations, we first give an intuitive experiment to validate the effect of surrogate-based tuning, which is the foundation of Hydro. As shown in Figure 3.10 (a)∼(d), we randomly choose 10 hyperparameter configurations (denoted as A∼J) on the ResNet-18 model and build surrogate models with Hydro using different scaling ratios $S = 2, 4, 8$, where $S = 1$ represents the target model. We train each model for 100 epochs on the CIFAR-10 dataset with a fixed seed=1. Since the HPO job is essentially a ranking problem of hyperparameter configurations, we mainly care about whether the order is maintained especially for the top configurations, namely hyperparameter transfer effect. From the result, it is obvious that the performance ranking of hyperparameters transfers well across different scaling ratios. Admittedly, configurations G and H are swapped when $S \geq 4$, but it has no influence on the final tuning result since they perform poorly and top configurations keep a consistent ranking. Besides, the wider model always outperforms the narrower one under the same hyperparameters, which is inline with MU parametrization theory and demonstrates that surrogate model can effectively transfer multi-dimensional hyperparameters.

Additionally, we also validate the inter-trial fusion effect, which is another key mechanism of Hydro. Figure 3.11 shows the training loss curves of trials C, E, H and their fused versions. We select these three trials because their `batchsize` and `momentum` are consistent and only differ in `lr`. As we can see, the convergence curves of the fused model well overlap with the original standalone training curves, which demonstrates that inter-trial fusion is a mathematically equivalent transformation and does not affect the model convergence.

3.6.3 End-to-End Performance of Hydro Tuner

To cover most hyperparameter tuning scenarios in practice, we conduct end-to-end experiments across 6 workloads with different settings and 3 common tuning paradigms

TABLE 3.4: Summary of tuning performance with a deadline.

Deadline (s)	# of GPU	Model	Avg. Accuracy		
			Ray	Ray+ES	Hydro
900	4	VGG-11	65.42%	66.39%	**68.68%**
		ResNet-18	89.66%	90.71%	**91.32%**

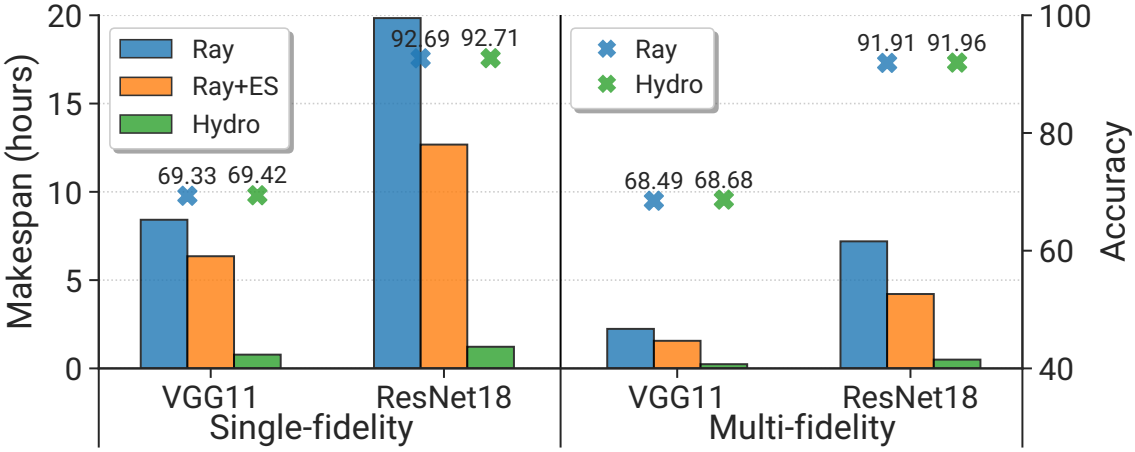


FIGURE 3.12: Summary of the end-to-end results. Bars indicate tuning makespan and points represent final model accuracy.

(case I∼III). Note that Hydro Tuner adopts a fixed resource size (without enabling Hydro Coordinator) for fair comparisons.

Case I: single-fidelity tuning. When a user seeks for extremely excellent model performance with ample resources, single-fidelity tuning is applied to avoid missing the best hyperparameter configuration. Table 3.2 summarizes the Hydro improvement on single-fidelity tuning over different sizes of workloads, where we apply $S = 16$ for XL models and $S = 8$ (default value) for other models. Since HPO jobs require completely training massive trials, we perform each experiment twice and report their average results on time reduction and tuned model quality over Ray. Besides, we obtain Ray tuning time of XL experiments based on simulation due to their unacceptable tuning cost, and adopt the official hyperparameter configurations [22, 201] to train the model as quality baselines. The target model training time is included in Hydro.

From the table, we can see that Hydro substantially outperforms Ray by 8.7∼78.5× in time reduction, while obtaining better final model quality. The time reduction mainly derives from two aspects: (1) *Less resource demand of trials.* For instance, the scaled GPT-3 XL trials do not require distributed training. For smaller models, Hydro further applies inter-trial fusion to improve trial concurrency and resource utilization. (2) *Smaller model trains faster.* Each trial has fewer FLOPs (Figure 3.3) to compute, which is more obvious on larger models. Additionally, we also observe

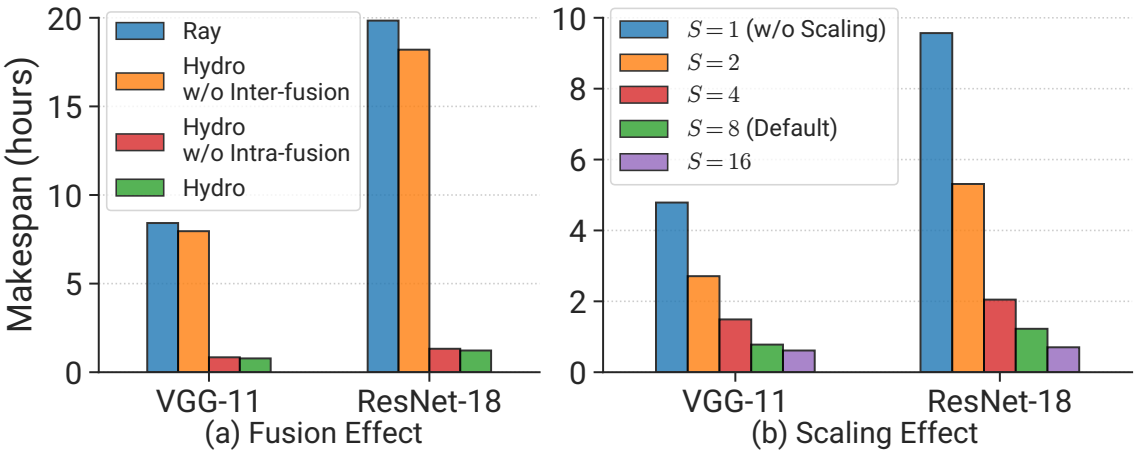


FIGURE 3.13: Ablation study. (a) Effect of inter- or intra-trial fusion. (b) Makespan of different scaling ratios.

that the effect of Hydro is more evident for larger models, with more intensive trials and fewer resources. This reflects Hydro is more suitable for large-scale HPO jobs with limited resources, which is hard to handle by existing systems.

Case II: multi-fidelity tuning. When a user desires to obtain a good model with a relatively lower cost, multi-fidelity tuning is applied to search hyperparameters efficiently. Table 3.3 reports the Hydro performance on multi-fidelity tuning. We keep the same experiment settings as Case I, except using half GPU resources. Besides, we configure ASHA [132] with $bracket = 1, grace = 3, reduction = 3$. We observe that Hydro can achieve 5.8∼33.4× reduction over Ray. Hydro can further benefits ASHA due to its much higher concurrency, which prevents the inaccurate promotion issue of ASHA [202]. Furthermore, we find that Hydro can also slightly improve the final model quality, which is mainly due to the different model initialization and more balanced layer-wise training rate configuration by Hydro parametrization. The results are also in line with Figure 3.1 that Hydro delivers a lower loss.

Case III: tuning with a deadline. When a user wants to get a model as good as possible by a fixed deadline, budget-bounded ASHA is applied. We simply evaluate two models with a deadline of 15 minutes as shown in Table 3.4. Hydro outperforms other baselines in final model accuracy within a limited time since it can well hide the target model training time inside the surrogate model tuning with Eager Transfer.

End-to-end result visualization. Figure 3.12 summarizes the makespan and accuracy of VGG-11 and ResNet-18 across different tuning algorithms and baselines. We note that Ray and Ray-ES share the same accuracy point since elastic and GPU sharing have no effect on the final model quality. The surrogate-based tuning (Hydro) can significantly reduce the search makespan without sacrificing the model accuracy. We only select these two models for presentation because of their relatively obvious

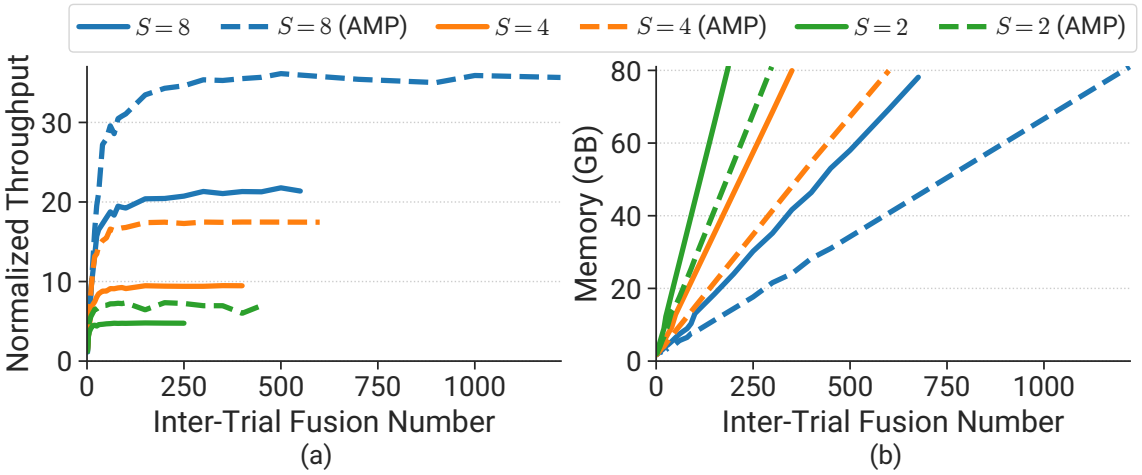


FIGURE 3.14: Sensitivity analysis of S and AMP on ResNet-18. (a) Accumulated throughout. (b) GPU memory footprint.

efficacy of Ray-ES. Ray-ES has less improvement over Ray for larger models like WideResNet-50, since it cannot benefit from GPU sharing and the elastic improvement is limited (only for later stage).

3.6.4 More Evaluation on Hydro Tuner

Ablation study of fusion. Figure 3.13 (a) reveals an interesting observation that Hydro can only achieve very limited improvement over Ray if inter-trial fusion is disabled, even though we have scaled the model by 8×. This is because GPUs are underutilized for such small models and there is no evident training speedup although we scale the model. Hence, it is important to combine Model Shrinker and Trial Binder to achieve the desired performance. Additionally, we also evaluate the effect of intra-trial fusion. However, we find its improvement is limited on small models.

Sensitivity analysis of scaling. Figure 3.14 clearly presents the effect of the scaling ratio S on GPU memory and accumulated fused trial throughput, where the normalization base is the throughput of the target model. We find that the peak throughput increases linearly alone with S. GPU memory also shows a similar pattern. In Figure 3.13 (b), we further evaluate the effect of the scaling ratio S on the overall tuning time. Hydro can continuously obtain benefits from higher scaling ratios. Besides, the final model accuracy maintains stable.

Sensitivity analysis of AMP. Figure 3.14 analyzes the effect of mixed precision training (i.e., AMP [203]), where solid and dashed lines represent the settings without and with AMP, respectively. We can find that the peak throughput can be further improved via enabling AMP. Besides, its effect on memory is also obvious, improving nearly 2× maximum fusion count.

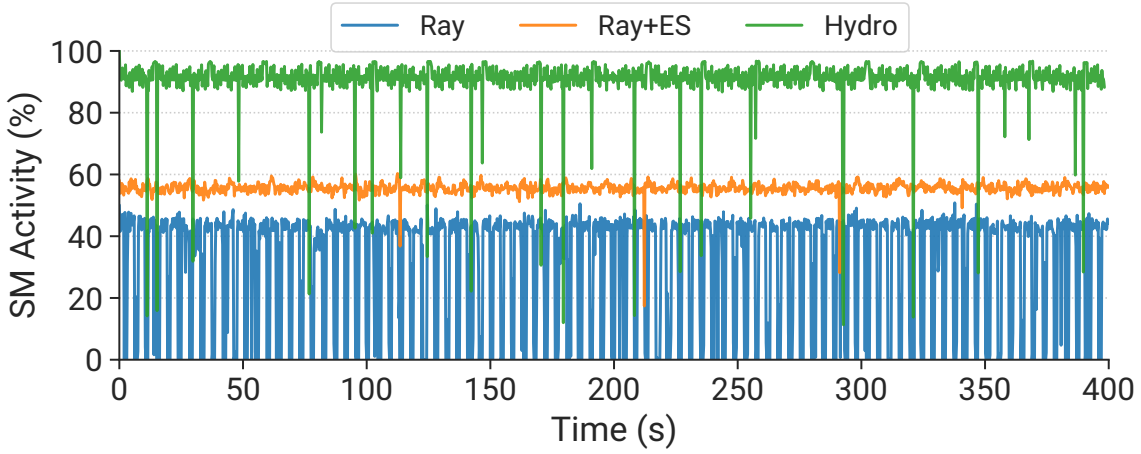


FIGURE 3.15: GPU utilization of HPO systems on ResNet-18.

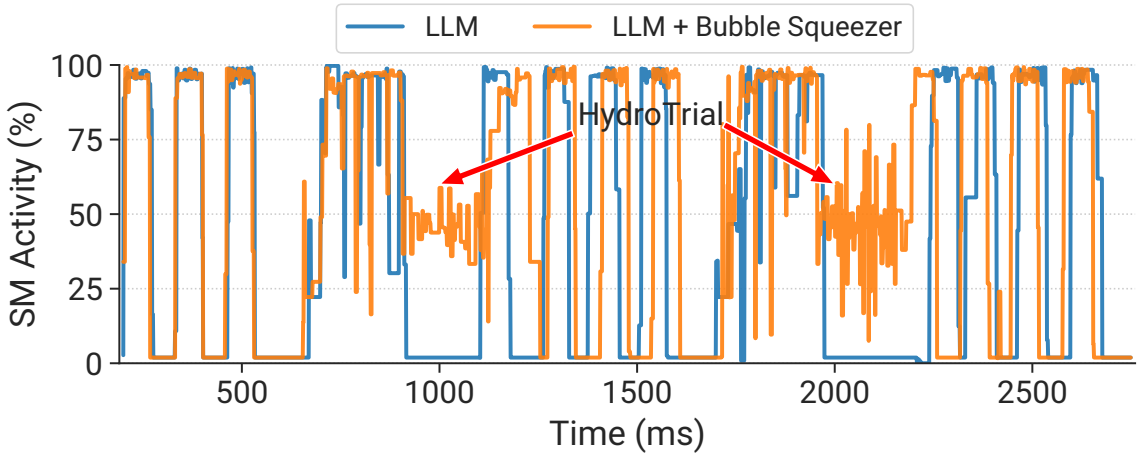


FIGURE 3.16: Visualizing Bubble Squeezer effect via DCGM. Two iterations of the first pipe stage are presented. The execution periods of the HydroTrial are highlighted by red arrows.

Impact on GPU utilization. Figure 3.15 plots the GPU utilization traces on one GPU for 300 seconds using different HPO systems. We employ NVIDIA DCGM [188] to record `SM Activity` as GPU utilization. It is obvious that Hydro can achieve much higher GPU utilization than other baselines owing to the superior capability of inter-trail fusion [127].

Overhead analysis. We perform the overhead analysis on the ResNet-18 multi-fidelity tuning workload. Its overhead mainly derives from two aspects: (1) *profiling* accounts for 0.8%; (2) *defusion* (including trial restart) accounts for 3.3%. The associated overhead is minor when weighed against the substantial enhancements in the tuning efficiency of Hydro.

3.6.5 Hydro Coordinator Evaluation

Bubble Squeezer. To evaluate the impact of Bubble Squeezer, we interleave `HydroTrials` with a large GPT model over 32 A100 GPUs containing 4 pipeline stages on 4 nodes, which is implemented based on DeepSpeed [177] along with MegatronLM [183, 204, 205]. We measured the SM activity with and without Bubble Squeezer in Figure 3.16. Two traces are collected separately and we align them at the beginning of the figure. For the original GPT training, since the only active kernel in the bubble is NCCL kernel for communication, the SM activity is extremely poor (about 2%) during the bubble. Hydro utilizes the unused SMs and achieves a relatively high SM utilization at about 50%, with no evident slowdown to the GPT model training. Here the `HydroTrial` is ResNet-18 model with fusion count $F = 16$, obtaining around 15% of exclusive throughput. We also measure the throughput influence of direct colocation and find it causes unacceptable interference (about 12% slowdown for the large model). Additionally, we further simulate the end-to-end performance of Bubble Squeezer. Here we set that the Hydro tuning job can only apply 1 exclusive GPU since most resources are occupied by the large model. We find the makespan of the tuning job can be greatly reduced by 2.7× with the free lunch.

Heterogeneity-Aware Allocator. We create a tiny cluster partition with 2 A100 and 2 V100 nodes (32 GPUs in total) to evaluate the impact of Heterogeneity-Aware Allocator. Besides, we uniformly sample 20 middle-size HPO jobs from Table 3.3 and randomly generate their job arrival time within one hour. Compared to resource-agnostic allocation, we find Heterogeneity-Aware Allocator achieves approximately a 1.3x reduction in the average job completion time.

3.7 Related Work and Discussion

AutoML systems. Automated Machine Learning (AutoML) refers to the process of automating the tasks associated with optimizing ML model performance. In general, AutoML comprises two essential components: HPO and Neural Architecture Search (NAS). NAS systems (e.g., Retiarii [7], ModularNAS [206]) aim to discover the optimal model architecture for a specific task. On the other hand, HPO focuses on optimizing the hyperparameters of a fixed architecture, usually separate from NAS. Our work primarily concentrates on HPO.

Prior HPO systems like HyperSched [112], Rubberband [118] and Seer [119] support elastic training to allocate more GPU resources to promising trials, which is also supported in Hydro. Elastic training can make use of idle GPUs but fails to improve single GPU utilization. On the other hand, Fluid [32] further leverages NVIDIA MPS

[26] technique to allocate multiple trials on a single GPU. HFTA [127] achieves inter-trail fusion on a shared accelerator. They can improve hardware utilization but only work well on tiny models (e.g., AlexNet [207], PointNet [208]). Based on the unique surrogate-base tuning nature, Hydro significantly extends the fusion application scope via model scaling and achieves automatic model fusion with minimum manual effort.

Pipeline parallelism and interleaving execution. Recent studies exploit bubbles induced by pipeline parallelism from multiple angles. Bamboo [126] fills redundant computations into bubbles to provide resilience and fast recovery for preemptible cloud instances. EnvPipe [209] selectively lowers the SM frequency of bubble periods to save energy. Unlike them, Hydro leverages bubbles to train HPO trials via interleaving execution, which is inspired by some prior works. For instance, Wavelet [210] and Zico [211] reduce the GPU peak memory based on interleaving. Muri [212] supports multi-resource interleaving to reduce contention.

Limitations. Despite the extraordinary performance, Hydro's surrogate-based tuning paradigm does have three limitations: (1) Hydro parametrization does not support regularization hyperparameters, such as weight decay and dropout, as elucidated in §3.4.1. (2) Hydro does not allow for any customized initialization techniques because Hydro implements its own automatic layer-wise re-initialization mechanism, which plays a crucial role in parameterization. (3) Hydro does not support fine-tuning since its theory is built atop i.i.d. samples (requiring the same dataset). Nevertheless, Hydro can deliver qualified models for most cases.

3.8 Summary

This chapter presents Hydro, a surrogate-based hyperparameter tuning service that provides job and cluster level optimization via automated model scaling, fusion and interleaving. Our experiments show that Hydro can dramatically reduce the tuning makespan and improve the cluster resource utilization.

Part II

Building Practical Systems: Optimization on Machine Learning Systems in Various Domains

Chapter 4

Lucid: A Non-Intrusive and Interpretable DL Scheduling System

4.1 Introduction

Over the past decades, Deep Learning (DL) presents incredible performance and rapid popularity across many applications, including image classification [213], recommendation [214], etc. To facilitate DL model development, IT companies and research institutes often build large-scale multi-tenant DL clusters [1, 23, 45]. The cluster scheduler is dedicated to managing these expensive infrastructures and regulating various DL workloads. Several recent works have proposed schedulers tailored for DL training workloads [1, 18, 23, 30, 43, 45, 56, 215], and demonstrated their remarkable performance in improving computation resource utilization and job training efficiency. However, there exist significant gaps (**G1∼G5**) in deploying them in practice from two perspectives.

First, to achieve better system performance, most state-of-the-art approaches rely on preemption-enabled scheduling paradigms, such as migration [18], elasticity [57] and adaptive training [56]. Nevertheless, owing to their inevitable intrusive mechanism, they meet the following barriers in deployment:

- **G1**: ***Inflexible and error-prone***. In order to realize elastic training and job checkpointing, existing schedulers require users to import specific libraries and modify their codes to implement these mechanisms [18, 43, 46, 56, 57]. Such *user-code intrusive* approaches not only burden users with complex logic of model training control but also potentially incur uncertain bugs. Additionally, they also greatly limit users' flexibility in customizing their codes since the scheduler takes over the

training workflow. As stated by Microsoft [216], "most DNN training workloads today as such are not checkpointable or resizable." The generalization issue also hinders the practical application of intrusive schedulers.

- **G2**: ***High integration and maintenance cost***. It is nontrivial to shift a research prototype into a production-level system. Typically, integrating a scheduler design into a commercial or open-source cluster management system requires an expert team with enormous efforts and costs to handle all the possible issues. Further, to support advanced scheduling features, some schedulers [18, 57, 216] require the modification of the source code of the underlying DL frameworks (e.g., Pytorch [15]) or CUDA library [217]. They need continuous maintenance to accommodate to the fast version iteration of DL ecosystems. The exorbitant integration and maintenance cost are impractical for most companies and research institutes.

- **G3**: ***Model quality degradation of adaptive training***. To strive for extreme training efficiency, some schedulers [56, 215, 218] adaptively adjust the job batch size and learning rate according to the allocated resources. However, this can degrade the quality of the final model in terms of validation performance [219, 220]. In commercial applications, minor quality improvement drives a significant increase in customer engagement and company profits [221]. Therefore, developers are not prone to adopt this mechanism due to the degradation issue.

Second, plenty of schedulers adopt machine learning (ML) based methods [23, 30, 222–224] or optimization-based methods [46, 212, 225–227] to find the optimal scheduling policy. However, they also suffer from significant flaws in practice:

- **G4**: ***Limited scalability***. As workloads become more intensive and clusters become larger-scale, these schedulers [46, 222, 223, 225, 226, 228, 229] meet the scalability bottleneck when deployed in production-level systems. For instance, Gavel [46] spends thousands of seconds solving a 2048-job allocation problem through linear programming, which takes too long to meet the real-time requirement [225]. Reinforcement Learning (RL) based schedulers also confront the same issue: Metis [228] only affords to handle dozens of jobs while production clusters can run thousands of jobs concurrently.

- **G5**: ***Opaque decision making and hard to adjust***. Most ML-based schedulers are built on black-box models such as Random Forest (RF) [230, 231], Gradient Boosting Decision Tree (GBDT) [23, 30] and RL [222, 223]. Developers mainly focus on improving key scheduling metrics (e.g., makespan) while ignoring their *interpretability*. The prediction processes of these model are unintelligible to humans [232–234]. Due to such opacity, system operators cannot guarantee model predictions are reliable and have insufficient confidence to deploy them. In addition, ad hoc debugging and system configuration tuning are also substantial challenges to

both the ML-based and optimization-based schedulers. Improper modifications may cause severe performance degradation [37].

To bridge these gaps, we design Lucid, a non-intrusive and transparent scheduler that can provide better performance than preemptive and intrusive schedulers. The core design of Lucid derives from the following three insights. First, *it is feasible to address the cluster GPU underutilization issue in a non-intrusive manner.* Since GPUs are commonly underutilized across production-level DL training clusters [2, 23, 24], existing DL schedulers pack jobs to increase the utilization through an intrusive manner [18, 30, 31, 46, 235]. However, by comprehensively analyzing job colocations, we find it is possible to achieve efficient job packing without any intrusion. Second, *forecasting job duration from prior history is attainable.* Since a majority of workloads follow recurrent patterns and users tend to submit similar tasks multiple times [2, 23], we can estimate the duration of new jobs based on their profiled features and historical submission data. Third, *system interpretability is indispensable and can deliver performance improvement.* Comprehensive understanding of system behaviors can enhance operators' confidence for practical deployment and provide transparent performance tuning.

Incorporating the above observations, we design Lucid to minimize the average job completion time (JCT), improve the resource utilization and shorten the debugging feedback delay in DL clusters. It consists of three key scheduling modules along with the corresponding interpretable models (Figure 4.3). Specifically, (1) we propose a *two-dimensional* optimized *Non-intrusive Job Profiler* to collect job resource usage features, including GPU utilization, GPU memory footprint and GPU memory utilization. It achieves timely debugging job feedback and highly efficient job metric collection where profiling takes only minutes in a non-intrusive manner. (2) In the job packing stage, we introduce an *indolent* and *dynamic packing* strategy for *Affine-jobpair Binder* to circumvent interference and maximize the cluster-wide job speed. (3) A *Workload Estimate Model* assigns a priority value to each job for the following *Resource Orchestrator*. Besides, Lucid integrates an *Update Engine* for model performance maintenance and *System Tuner* for transparent adjustment and system enhancement.

To extensively assess the performance of Lucid, we conduct evaluations in a physical cluster and perform large-scale simulations with three production traces from SenseTime (Chapter 2) and Microsoft [24]. Experimental results show that Lucid significantly improves the average JCT by 5.2∼7.9× compared with the non-intrusive policy FIFO. Even compared with the state-of-the-art intrusive policy Tiresias, Lucid obtains average JCT and queuing delay improvement by 1.1∼1.3× and 1.8∼9.1× respectively. In addition, Lucid successfully copes with the aforementioned deployment problems (**G1∼G5**) and achieves the following desirable properties:

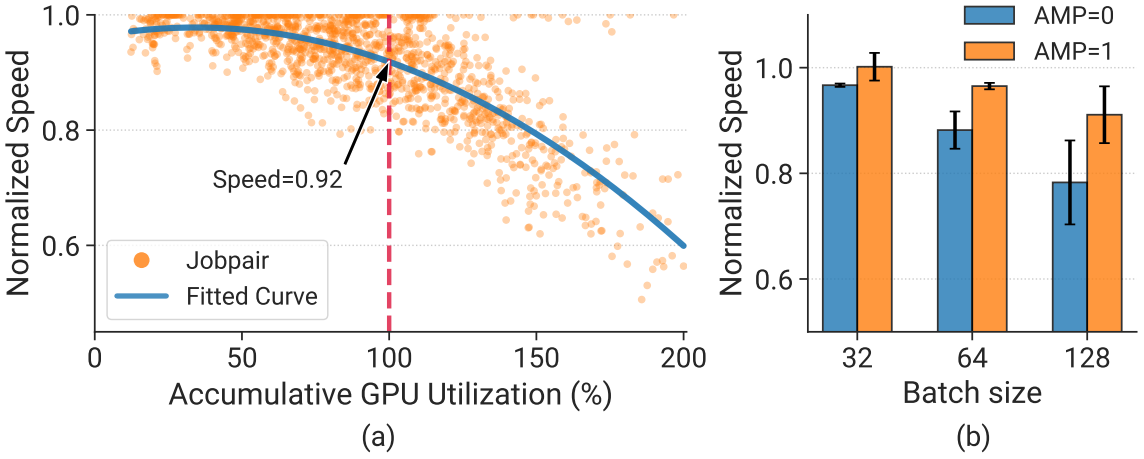


FIGURE 4.1: Motivation. (a) Accumulated GPU utilization of colocated jobpairs against average speeds. (b) Average effect of batch size and mixed-precision to packing performance.

- **A1**: ***Efficient non-intrusive scheduling***. The workflow of Lucid is preemption-free and requires no intrusion to the codes of users' jobs or DL frameworks. Meanwhile, Lucid outperforms several SOTA intrusive schedulers.
- **A2**: ***Low deployment cost***. Lucid can be easily integrated into existing commercial or open-source cluster management systems (e.g., Slurm [17], Kubernetes [16]). It also has no demand for continuous maintenance of DL framework or CUDA library updates.
- **A3**: ***Model performance preservation***. Users take full control over their models and Lucid never tampers with model configurations, fully preserving their original quality.
- **A4**: ***Scalability to large-scale cluster***. Even for massive and complex workloads, the system can obtain the optimal scheduling policies swiftly (within several milliseconds).
- **A5**: ***Transparent system tuning***. All the modules are interpretable, helping developers make guided system configuration adjustments and bringing extra improvement.

To the best of our knowledge, Lucid is the *first* DL job scheduler that considers system interpretability and focuses on system practical deployment. We systematically summarize the deficiencies of existing works (**G1**$\sim$**G5**) and propose an end-to-end solution to overcome them. And we demonstrate the non-intrusive scheduler can outperform intrusive approaches in production-level clusters.

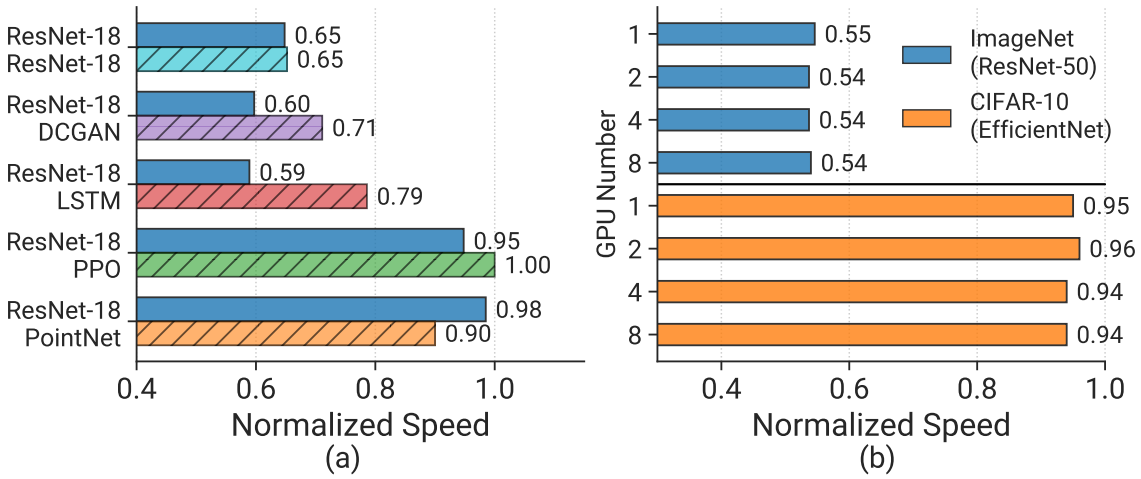


FIGURE 4.2: Packing Examples. (a) Colocate with ResNet-18. (b) Two same jobs packing with different GPU numbers.

4.2 Motivation

In this section, we highlight the characteristics of DL clusters and job colocation that inspire the design of Lucid.

4.2.1 Existing DL Workload Scheduling

It is a common practice for tech companies and research institutes to build multi-tenant DL clusters to facilitate DL model development. A DL cluster scheduler is adopted to regulate the resources and job execution. To improve resource utilization and minimize the average JCT, most existing DL cluster schedulers [1, 18, 30, 43, 45, 56, 215] are *intrusive*: they implement some advanced features through modifying DL frameworks or relying on user-code adaptation. There are two common advanced features: (1) *job packing* (i.e., job colocation, GPU sharing) allows multiple tasks to share the GPU using the NVIDIA MPS [26] or MIG [25] technologies. Based on the repetitive pattern of DNN training, operators can profile a few iterations to obtain the resource utilization features of the job. Existing profiling-based DL job schedulers that rely on intrusive libraries to inspect job execution status. (2) *elastic training* dynamically adjusts the scale of GPU workers and even modifies the batch size and learning rate adaptively to accelerate the job training progress [56, 215]. However, they have several significant drawbacks as mentioned in §4.1 (**G1∼G3**).

TABLE 4.1: Summary of models and datasets used in our experiments. *AMP*: Enable/Disable mixed precision training.

Task	Model	Dataset	Batch size	AMP
✻	ResNet-50 [113]	ImageNet [192]	32, 64, 128	+/-
✻	MobileNetV3 [172]	ImageNet [192]	32, 64, 128	+/-
✻	ResNet-18 [113]	CIFAR-10 [236]	32, 64, 128	+/-
✻	MobileNetV2 [237]	CIFAR-10 [236]	32, 64, 128	+/-
✻	EfficientNet [238]	CIFAR-10 [236]	32, 64, 128	+/-
✻	VGG-11 [173]	CIFAR-10 [236]	32, 64, 128	+/-
❋	DCGAN [239]	LSUN [240]	32, 64, 128	+/-
❇	PointNet [208]	ShapeNet [241]	32, 64, 128	+/-
♦	BERT [120]	SQuAD [242]	32	+/-
✦	LSTM [92]	Wikitext2 [190]	64, 128	+/-
◆	Transformer [167]	Multi30k [243]	32, 64	-
❖	PPO [13]	LunarLander	32, 64, 128	-
❖	TD3 [244]	BipedalWalker	32, 64, 128	-
★	NeuMF [245]	MovieLens [246]	64, 128	+/-

CV: ✻ Img. Classification ❋ Img.-to-Img. Translation ❇ 3D Point Cloud Classification
NLP: ♦ Question Answering ✦ Language Modeling ◆ Language Translation
RL: ❖ Physics Control (Box2D) **Recommendation:** ★ Movie Recommendation

4.2.2 Opportunities for Efficient Non-intrusive Scheduling

Characterizing Job Packing Interference. To understand the interference effect of job packing, we conduct an extensive analysis of various workloads (Table 4.1) with different configurations across various domains, including computer vision, natural language processing, reinforcement learning and recommendation. We place two DL workloads on the same GPU, and measure the performance of all the possible combinations of job packing pairs. All the experiments are performed on our testbed (§4.4.1) equipped with NVIDIA RTX3090 GPUs and implemented with Pytorch 1.10 [15].

Figure 4.1 (a) shows the relationship between the GPU utilization and speed of all measured jobpairs, as well as the fit curve obtained through least-squares polynomial fit. The y-axis represents the average value of two normalized speeds and each orange point represents one colocation measurement. Obviously, there exists a strong correlation between the accumulative GPU utilization and job interference. When the GPU utilization summation reaches 100%, most jobpairs can still obtain over 0.8× speed (around 0.92× on average). More concretely, Figure 4.2 (a) shows some representative cases of job packing (batchsize=64, AMP=0), where the normalized speed indicates the ratio of colocated and exclusive job speed. We can clearly observe that ResNet-18 barely has degradation when colocated with PointNet or PPO, while

nearly 40% speed degradation occurs when colocating with other workloads. Besides, there should be less interference in the future GPU generations (Figure 1.3 (b)).

As for parallel training jobs, different from stereotypes, we find their colocation brings similar benefits to single-GPU jobs. For instance, we depict the same job colocation effect of both the heavy (blue bar) and light (orange bar) workloads in Figure 4.2 (b), where every single GPU allocates consistent 64 mini-batches. We observe that jobs of different scales within a single-node present equivalent performance. Additionally, we also consider the effect of mixed precision training. Figure 4.1 (b) indicates employing such training manner can deliver extra job packing benefits so we further consider AMP in Lucid. We also consider the three-job packing situation and find it typically suffers from acute speed degradation, which is in line with previous work [46].

Non-intrusive Interference-aware Job Packing. All of existing packing-enabled DL schedulers rely on the intrusive paradigm. Specifically, they modify DL frameworks [1, 18, 31] or require user-code adaptation [30, 46, 235] to achieve introspective job packing. However, we find it is feasible to realize interference-aware job packing non-intrusively. According to our characterization, the non-intrusive *GPU utilization* metric should be sufficient for schedulers to make packing decisions (Figure 4.1 (a)) and the packing strategy is applicable to all single-node jobs (Figure 4.2 (b)), covering over 95% workloads (§1.2.3). Notably, *GPU utilization* is defined as the percentage of the time in a given sample interval where one or more kernels are executed on a GPU instead of active unit percentage [30, 94]. In addition to this, we adopt another two non-intrusive features that can also help us make more precise decisions: *GPU memory utilization* (percentage of time that memory was being read or written over the past sample period) and *GPU memory* (memory occupation on the GPU).

Job Duration Estimation. Recent DL cluster analysis works from SenseTime (Chapter 2) and Alibaba [2] find that a majority of workloads have recurrent patterns and users tend to submit similar tasks multiple times. This inspires us to leverage the historical log data to predict job duration. In addition, profiled characteristics of job resource utilization can also help us match them with previous jobs more precisely, contributing to more accurate predictions and better scheduling policies.

4.3 System Design

To provide an efficient and transparent scheduling policy in practice, we design Lucid, a learning-augmented non-intrusive DL workload scheduler for DL clusters. Below we introduce its architecture and the detailed design of each module.

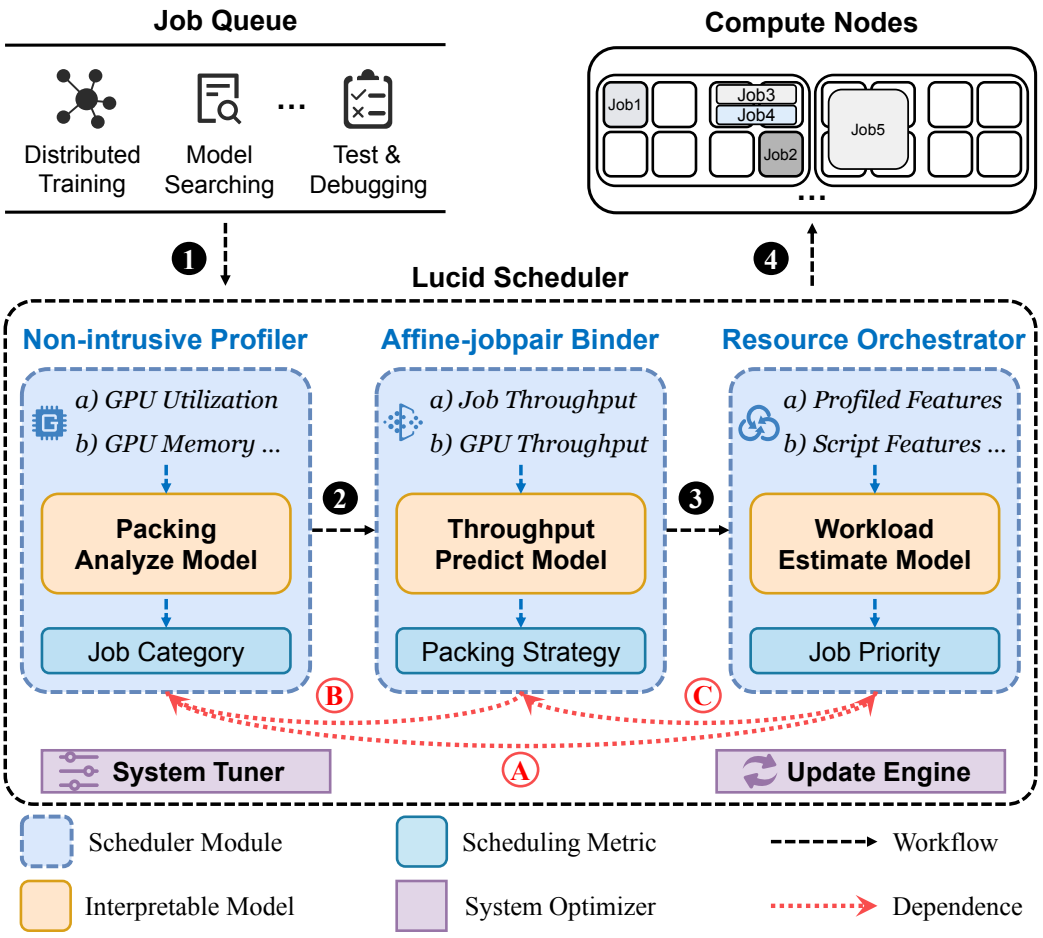


FIGURE 4.3: Overview of Lucid system architecture. Each module contains an interpretable model for key metric prediction. System optimizers are applicable to all components tuning. Scheduling workflow and module dependencies are represented by black and red arrows respectively.

4.3.1 Overview

Principles & Goals. For practical and simple system adoption, Lucid follows three design principles: (a) *Non-intrusive*. The whole scheduling workflow follows a preemption-free manner and requires zero user-effort and DL framework modification (solving **G1**∼**G3**). (b) *Scalable*. The system can obtain scheduling policies promptly for massive and complex workloads (solving **G4**). (c) *Interpretable*. All the modules are transparent and can be clearly adjusted by the cluster operators (solving **G5**). Our primary objective is to *minimize average JCT* for training workloads. This is particularly desirable for DL users. Additionally, Lucid also improves resource utilization and provides timely debugging feedback. Our future work aims to serve more scheduling goals, such as fairness and service-level guarantees.

Architecture & Workflow. Figure 4.3 illustrates Lucid's architecture along with the scheduling workflow. It consists of three key *scheduler modules* (blue blocks)

Algorithm 3 Space-aware Profiling

Input: New Job: $\mathcal{J}$, **Job Profiling Queue: Q**

procedure SPACE-AWARE PROFILE($\mathcal{J}$, **Q**)
  if $\mathcal{J}.gpu \leq N_{prof}$ **then** ▷*Job Scale limit*
    Enqueue $\mathcal{J}$ to **Q**
  SortJobGPUNum(**Q**) ▷*Sort by Least GPU First*
  CheckRunningJobs(T_{prof}) ▷*Evict Overtime Running Jobs*
  for all *Job* ∈ **Q** **do**
    if Consolidate(*Job*) is **True then**
      ConsolidateAllocate(*Job*) ▷*Job Start Profiling*
      Dequeue *Job* from **Q**
      Non-intrusiveProfile(*Job*)
    else
      break

for workload scheduling, as well as two *system optimizers* (purple blocks) for performance enhancement and maintenance. For every module, there is a corresponding *interpretable model* (orange blocks) in charge of forecasting key metrics to assist scheduling. The system workflow of Lucid is presented by black arrows. Specifically, before allocated to the target cluster, jobs need to be profiled first (❶). We adopt a *Non-intrusive Job Profiler* to filter the majority of the test and debugging jobs. Meanwhile, this module also records the resource usage statistics of normal training jobs and classifies them into different categories (❷). After profiling, we design an *Affine-jobpair Binder* to determine whether and how to pack various jobs. It dynamically changes the packing strategy according to the future cluster throughput prediction (❸). Based on the profiled and user-provided features, the *Resource Orchestrator* assigns a priority value to each job and selects jobs for allocation (❹).

Inter-module Dependence. Lucid achieves overall desired scheduling performance via the collaboration of all the system modules. Each single module without assistance from other modules cannot provide desired performance (§4.4.5). We depict their interactions in Figure 4.3 with red arrows: (A) *Orchestrator* adopts features from *Profiler* for better duration estimation. Lucid cannot precisely match previous recurrent jobs without profiled features. (B) The *throughput prediction model* not only determines the packing strategy inside *Binder* but also assists *Profiler* cluster scaling, which efficiently handles burst job submission cases. Jobs have to bear higher profiling queuing delays without *throughput prediction model*. (C) *Binder* requires the duration estimation from *Orchestrator* to optimize packing decisions. It is significant to be time-aware during job packing because long-term job packing sometimes deteriorates the HOL (Head-of-line) blocking issue and prolongs JCT.

4.3.2 Non-intrusive Job Profiler

Lucid adopts the job profiling mechanism to optimize the succeeding allocation strategy. The *Non-intrusive Job Profiler* sets a short-term runtime limit T_{prof} for each job and collects the hardware metrics related to the job profiling, including GPU utilization, GPU memory footprint and GPU memory utilization. These can be conveniently measured through NVIDIA-SMI [94] or DCGM [188] in a non-intrusive way. Then the profiler sends these features to the *Packing Analyze Model*, which follows the non-intrusive principle to proactively predict the effectiveness of packing instead of measuring the throughput after colocation. To facilitate the subsequent job packing and resource allocation, instead of predicting the numerical result of job colocations, Lucid classifies jobs into three distinct categories (*Tiny, Medium* or *Jumbo*) and assigns each job a *Sharing Score (SS)* to indicate its category. Specifically, *Tiny (SS=0)* jobs refer to those with extremely low resource utilization and they hardly suffer from colocation slowdown. Conversely, *Jumbo (SS=2)* jobs require high resource utilization and decisions on their colocation should be cautious. Packing of the *Medium (SS=1)* jobs generally delivers a relatively minor impact on their training speed.

To improve profiling efficiency, we propose a *two-dimensional* optimized profiling strategy that combines both the *space* consideration of workload profiling to minimize queuing delay, and *time* consideration of profiler cluster to maximize resource efficiency:

Space-aware Profiling. Due to the short profiling time limit T_{prof}, the time-scale of the workloads should be similar so we can focus on optimizing their space-scale scheduling, which is never considered by prior profiling-based DL workload schedulers [42, 45, 226]. By prioritizing jobs that request fewer resources, the head-of-line (HOL) blocking problem of small-scale profiling clusters can be efficiently solved. Algorithm 3 shows the pseudo-code of our *Space-aware Profiling* algorithm. Since the limited GPU resource is typically the bottleneck of DL training jobs, we sort jobs according to their GPU demands (line 4). Then we adopt exclusive and consolidated allocation policy (line 8) to reduce resource fragmentation (Chapter 2).

Time-aware Scaling. To guarantee resource availability for profiling, the profiling cluster is typically decoupled from the main computing cluster. However, due to the time-variant pattern of job submissions, the static profiling configuration may lead to severe queuing delay and resource imbalance. To this end, we propose *Time-aware Scaling* that dynamically adjusts the job scale limit N_{prof}, profiling time limit T_{prof} and profiling cluster capacity C_{prof} based on current states as well as future cluster-wide job throughput prediction. For instance, when a burst of jobs occur in a short time, the profiler will temporarily loan some nodes from relatively idle VCs and reduce

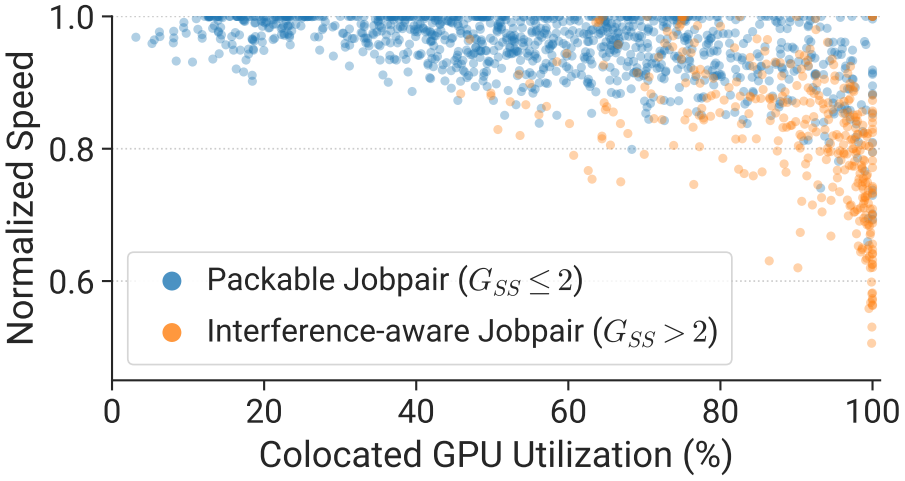


FIGURE 4.4: Indolent Packing. Lucid non-intrusively determines whether jobpairs are suitable for colocated execution (Blue Points) or should be exclusive execution (Orange Points).

T_{prof}. Resources will be returned when cluster throughput decreases and the burst job queue eliminates.

Note that profiling is required for most jobs, except large-scale distributed ones that exceed the job scale limit N_{prof}. Lucid collects the metrics of those large jobs on the fly without profiling. Additionally, we assume the job initialization or data movement time does not exceed T_{prof}, otherwise the profiler cannot obtain correct resource consumption features. To support such jobs, operators should prolong the T_{prof} setting accordingly or endow users the right to mark their jobs as "*Long Cold-Start*" jobs to extend T_{prof}.

Contrary to the common opinion that profiling brings extra queuing delay and resource demand [30], our profiling mechanism possesses the following superiorities: (a) *Timely Feedback*. Plenty of short-term debugging jobs suffer from severe queuing delays (§1.2.3) due to the runtime-agnostic scheduling paradigm of currently deployed clusters [2, 23, 24]. Whilst Lucid's profiler can well resolve this issue and improve the job fairness. (b) *Effortless*. Lucid does not rely on any intrusive metric (e.g., job progress, time-per-iteration) and requires zero code modification. (c) *System performance enhancement*. The profiler can filter out most failed or debugging jobs for the main cluster and thus significantly facilitate the scheduling optimization by diminishing the optimization space.

4.3.3 Affine-jobpair Binder

Different from previous packing-enabled schedulers [1, 18, 30, 46, 235] that apply user-code or DL framework intrusive approaches to identify jobpairs with interference,

Algorithm 4 Lucid Resource Orchestrator

Input: Job Queue: Q, Running Jobs: J

```
1: procedure LucidSchedule(Q, J)
2:     for all 𝒥 ∈ Q do
3:         Pred = WorkloadEstimateModel(𝒥)
4:         𝒥.priority = 𝒥.gpu × Pred                          ▷Assign Priority
5:     SortJobPriority(Q)                    ▷Sort by Job Priority (Ascending Order)
6:     if CheckSharingStrategy() is True then
7:         for all 𝒥 ∈ Q do                                  ▷Job Placement with Sharing
8:             P = CheckAffineJobPair(Q ∪ J)
9:             if P is not ∅ then
10:                if ConsolidateWithShare(𝒥, P) is True then
11:                    ConsolidateWithShareAllocate(𝒥, P)
12:                    Dequeue 𝒥 from Q
13:            else
14:                TryExclusivePlacement(𝒥)
15:    else
16:        TryExclusivePlacement(Q)                          ▷Sharing Disabled
```

Lucid determines the packing jobpairs under the non-intrusive principle according to the profiled features. To this end, Lucid designs the following two strategies in *Affine-jobpair Binder*.

Indolent Packing. Lucid only packs jobs that are not likely to cause interference. Although such an inactive way may miss some optimization opportunities, it can effectively refrain from interference and provide packing incentives for users. Specifically, *Indolent Packing* sets *GPU Sharing Capacity* (G_{SS}) for each GPU, which restricts the summation of packed jobs' *Sharing Score* below G_{SS} (default value = 2). Besides, Lucid sets the following rules for job packing: (1) it adopts a hard limit on GPU memory usage to prevent the out-of-memory (OOM) issue; (2) it never packs jobs with different GPU resource demands due to the straggler effect of parallel training; (3) it combines up to two jobs on a set of GPUs since packing over three jobs generally will not bring extra benefits [46]; (4) it introspectively evicts packed jobs if an unstable resource utilization pattern is detected; (5) distributed jobs will not be packed by default due to network contention. Figure 4.4 depicts the binder decisions of all possible jobpair combinations listed in Table 4.1. It is obvious that Lucid efficiently identifies jobpairs with little interference, where over 98.1% packable jobpairs are interference-free (threshold: 0.85 of normalized speed) and 87.0% packing opportunities are found with such non-intrusive policy.

Dynamic Strategy. Existing works [1, 18, 30, 46, 235] usually keep a fixed strategy on job packing without cluster-wide awareness. However, most clusters [23, 60] present diurnal patterns on the job submission rate (throughput) and cluster utilization. When clusters are relatively idle, the ignorance of cluster throughput may cause

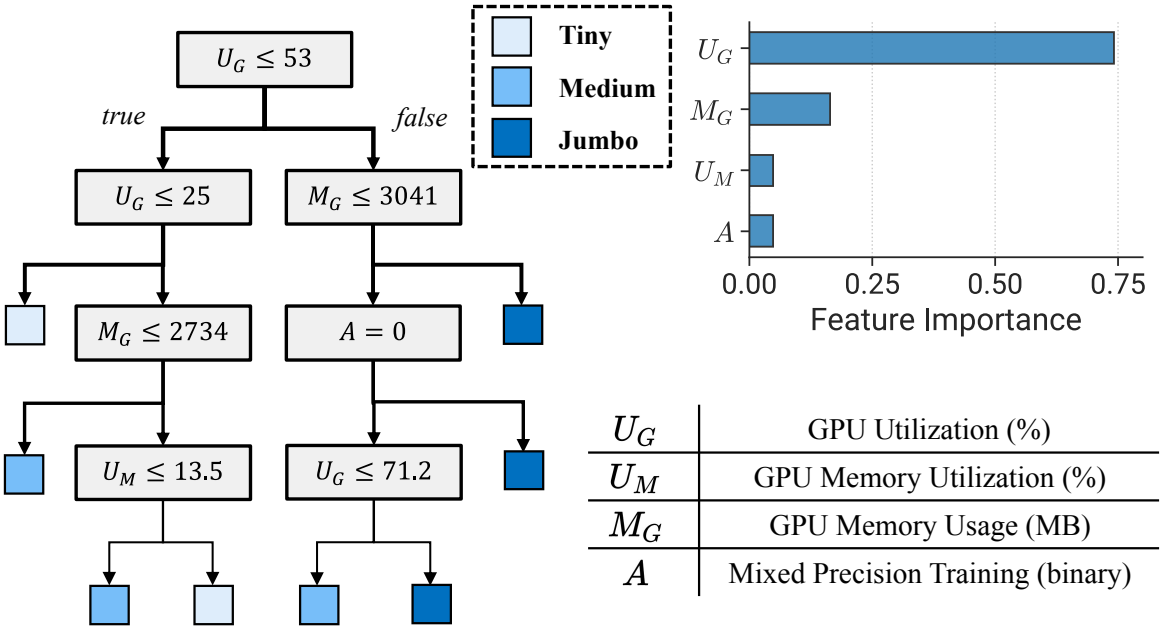


FIGURE 4.5: Packing Analyze Model. ***Left***: Visualization and interpretation. ***Right***: Feature importance and notation.

unnecessary job packing and prolong the job training progress. For this reason, we develop *Throughput Predict Model* to perform a time-series forecast on both the number of cluster jobs and GPU request throughput. Based on its prediction and current cluster states, when the current cluster throughput is relatively low (customizable) and not likely to increase in the future, we can dynamically adjust the packing strategy from *Default Mode* ($G_{SS} = 2$) to *Apathetic Mode* ($G_{SS} = 1$), and even disable job sharing temporarily for faster job completion.

4.3.4 Resource Orchestrator

To minimize the average JCT and increase resource utilization, Lucid employs *Resource Orchestrator* to manage cluster resources and orchestrate workload execution. The main challenge is to solve the HOL blocking problem, where long-running jobs have exclusive access to the GPUs until they are finished, keeping short-term jobs waiting in a queue [18]. The rule of thumb is to prioritize short-term jobs like the Shortest-Job-First (SJF) policy[45], whereas it is impossible to obtain perfect job duration information in reality. Besides, previous intrusive prediction paradigm [18, 43, 46] (i.e. iteration time measurement) can be misleading due to the high cancellation and failure rates of DL training jobs [23, 24]. However, as mentioned in §4.2.2, a majority of workloads are repetitive and we can leverage prior data to train *Workload Estimate Model* to provide job duration estimations for scheduling.

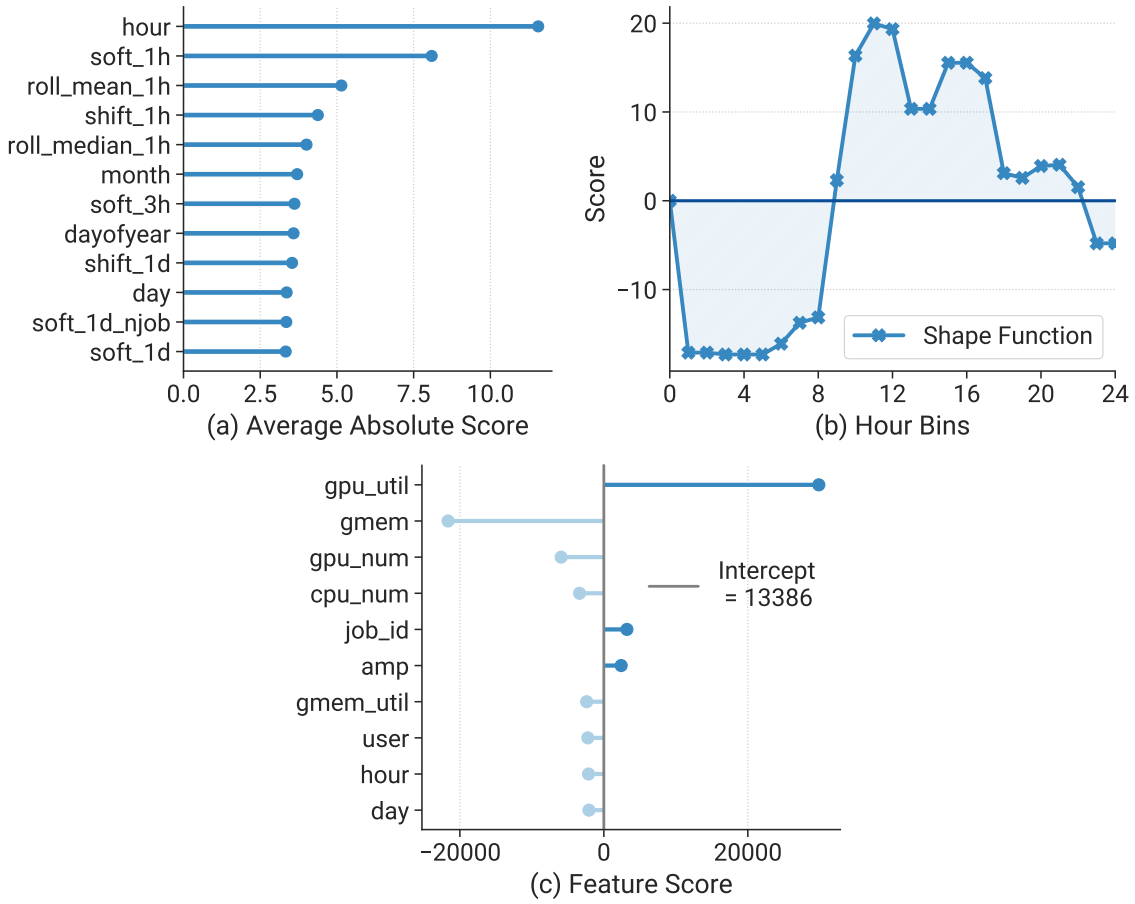


FIGURE 4.6: Throughput Predict Model (a & b): Global interpretation of overall feature importance and the learned shape function of *hour* (blue line). Workload Estimate Model (c): Local interpretation of features' contribution for one prediction.

Resource Orchestrator comprehensively considers both temporal and spatial aspects of DL jobs. Algorithm 4 illustrates the job scheduling and resource allocation procedure. First, *Workload Estimate Model* predicts the duration of each job and then the prediction is multiplied by the number of GPUs as the job's priority value (line 4). This additional consideration of job resource consumption (GPU demand) can efficiently improve scheduling performance [23, 45]. Next, the job queue is sorted according to the priority values. Then it checks whether job packing is allowed at the current moment (line 6). (1) If not, jobs are allocated in an exclusive manner (line 16). We apply the consolidate placement strategy to maximize the training speed of each job and reduce resource fragmentation. (2) If yes, we pack jobs suitable for colocation, and eliminate jobs with little remaining runtime (line 7). Besides, for new jobs without historical information, Lucid can generate an estimation for the new job based on the user's historical behavior. If it is submitted by a new user, Lucid can use the average duration of all the jobs with the same GPU demands as the duration

prediction (Chapter 2). Further, after the new job is terminated, *Update Engine* collects its information and uses the up-to-date data to fine-tune the model. In this way, jobs can be efficiently scheduled with less queuing and interference.

4.3.5 Interpretable Models

In order to provide accurate prediction and transparent interpretation for the cluster scheduling, Lucid employs Primo [37] interpretable models as the foundation for each scheduler module.

Packing Analyze Model. Inspired by LinnOS [247], which models SSD storage latency prediction as a binary classification problem, we introduce *Sharing Score* scheme to simplify interference prediction into a ternary classification problem for high scalability and intelligibility. Specifically, for each workload (Table 4.1) combination, we measure the exclusive and mutual colocation throughput to obtain a normalized speed. Then we assign a *Sharing Score* to each model configuration based on its colocation influence on others. A job is regarded as *Tiny* if its average normalized speed is greater than a customizable tiny job threshold (e.g., 0.95), and *Medium* if the speed is between tiny and medium job thresholds. Otherwise, the job will be labeled as *Jumbo*. We adopt the Decision Tree (DT) model for job category prediction to discover the common relationship between resource usage and job colocation features. DT can provide a transparent decision process and excellent prediction accuracy on this task. Besides, it requires less training data and performs robustly under dynamic system environments [37]. We leverage *minimal cost-complexity pruning* [248] to prune the learned tree to obtain a compact and accurate model.

Interpretation: Figure 4.5 presents the learned *Packing Analyze Model*. In addition to resource usage patterns (U_G, M_G and U_M), Lucid supports an optional metric (A), allowing users to specify whether to apply mixed precision training (e.g., *torch.cuda.amp*) in their job submission command. From this tree, we can clearly understand how Lucid classifies each job. We can also obtain an intuitive cognition of the overall model behavior by observing the depth of each decision path (arrow lines) and the right-side figure (feature Gini importance). Obviously, U_G affects colocation behavior most. Other metrics also assist to make a precise prediction.

Throughput Predict Model. We adopt a novel additive model algorithm GA^{2}M [249, 250] for cluster throughput prediction. GA^{2}M contains a series of *shape functions* $f(\cdot)$ and has the form: $y = \mu + \sum f_i(\boldsymbol{x}^i) + \sum f_{ij}(\boldsymbol{x}^i, \boldsymbol{x}^j)$, where μ is the intercept (averaged target value of training data) and $f_{ij}(\cdot)$ represents the interaction effect of features i and j. It provides comprehensive interpretations for the prediction process since each shape function is unary or binary and their combination is additive. We provide a more detailed explanation of GA^{2}M model in §5.3.2. To obtain precise future

throughput predictions, we extract time-related data such as the trend (increasing or decreasing) and seasonality (periodic pattern) of both cluster GPU demand and job submission through feature engineering. In detail, we encode repetitive patterns (e.g., hour, date) to explore the periodic variations. Besides, we calculate the average, median and weighted soft summation values of throughput under different rolling window sizes (e.g., 1 hour).

Interpretation: Figure 4.6 (a and b) presents the global interpretation of each feature importance and the learned shape function. It depicts the learned model from Saturn trace, which outperforms a series of complex black-box models (Table 4.7). From Figure 4.6 (a), we find the *hour* and a series of augmented features related to 1 hour ago play the most important roles in contributing to the model prediction. Furthermore, Figure 4.6 (b) illustrates the learned shape function of the *hour* feature, where each bin indicates a different hour of a day except that bin 0 is given a default value. This figure presents an obvious diurnal pattern which is excellently aligned with our experience, giving reliable and accurate advice on cluster configuration adjustment.

Workload Estimate Model. GA^{2}M is also adopted for job duration prediction. Specifically, the model extracts all features (e.g., user name, job id, GPU demand) and the actual job duration from the traces and encodes those categorical features. For the extremely sparse and high-dimensional features like job names, we utilize the Levenshtein distance [87] to convert them to relatively dense numerical values and leverage affinity propagation [251] to bucketize similar ones. For the temporal features like job submission time, we parse them into several time attributes, such as month or hour.

Interpretation: Figure 4.6 (c) presents the feature interpretation of one job prediction from the Venus cluster in SenseTime. The prediction result is the sum of every feature score and the intercept constant. Through the local interpretation, developers can clearly check the model behavior on each prediction.

4.3.6 System Optimizer

System Tuner. A cluster scheduler typically contains multiple parameters adjusted by system operators for better performance or different scheduling objectives. Tuning those parameters requires rich domain knowledge and manual efforts. Inappropriate adjustments may lead to severe performance degradation. The DL clusters in different companies and institutes have diverse workload types and distributions. Hence, the corresponding manual system tuning is necessary to obtain the optimal scheduling performance. Because of the nature of the data-driven policy, Lucid can be clearly adjusted via prior job and cluster information based simulation. Furthermore, to

TABLE 4.2: Summary of traces in large-scale simulations.

Trace	Source	#GPUs	#Jobs	Avg. Duration
Saturn [23]	SenseTime (Sep. 2020)	2,080	101,254	13,006s
Venus [23]	SenseTime (Sep. 2020)	1,080	23,859	5,419s
Philly [24]	Microsoft (Oct. 2017)	864	12,389	25,533s

optimize the performance of interpretable models, we adopt the Pool Adjacent Violators (PAV) [252] algorithm to pose a monotonic constraint [37] on the learned feature shape function based on the model interpretability.

Update Engine. In practical production-level clusters, the environments are dynamically changing, bringing workload and cluster distribution drifts. Therefore, frequent model fine-tuning or retraining is necessary to resolve the performance deterioration issue induced by stale models. To this end, we design *Update Engine* to adapt to the changes. It collects real-time system states, job logs, and uses up-to-date data to fine-tune Lucid models periodically.

4.4 Evaluation

In this section, we evaluate Lucid on a physical cluster and perform large-scale simulations with three production traces.

4.4.1 Experimental Setup

Implementation. We implement Lucid with approximate 4700 lines of Python code. It leverages the gRPC [253] to achieve the communication and control between the scheduler and workers. To evaluate the performance of Lucid in a large-scale cluster with long-term traces, we also implement a simulator to record job events and resource usage. The simulator is provided with measured resource utilization and job speed information of all possible tasks, including exclusive and colocated jobs. We confirm the simulation fidelity in §4.4.2. All experiment results without explicit comments are derived from the simulation. Besides, we implement Lucid interpretable models based on Primo [37]. For experiment workloads, we implement all models listed in Table 4.1 with Pytorch [15].

Testbed. We conduct physical experiments on a cluster of 4 servers and 32 GPUs. Each server is equipped with dual-sockets Intel Xeon Gold 6326 CPUs (64 threads, 256GB memory) and 8 NVIDIA RTX 3090 GPUs (24GB memory). All experiments are performed in the environment of Ubuntu 20.04, Pytorch 1.10, CUDA 11.3 and

cuDNN 8. Simulation experiments resemble the physical server configuration and adjust the cluster scale according to the actual traces.

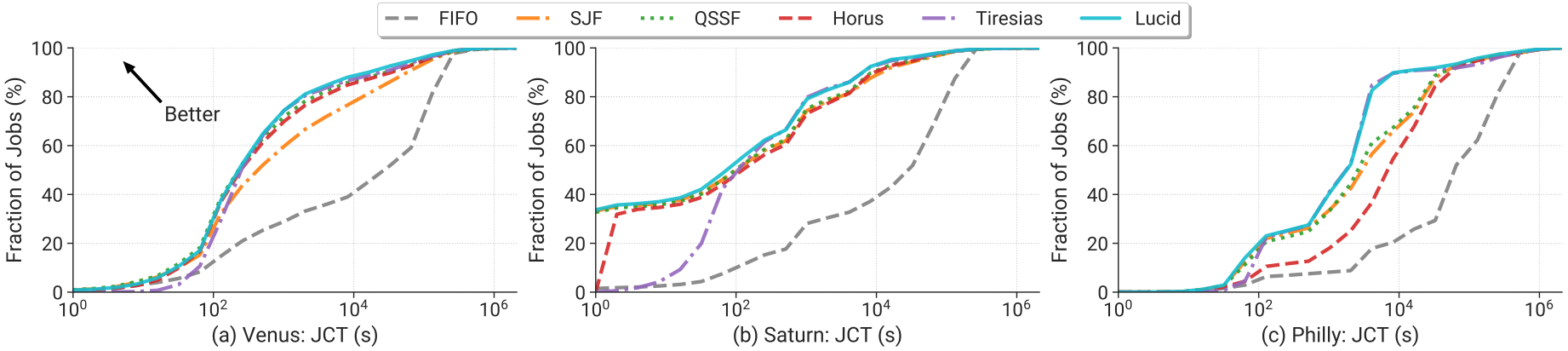


FIGURE 4.7: CDF of JCT using different scheduling approaches across three clusters: Venus, Saturn and Philly.

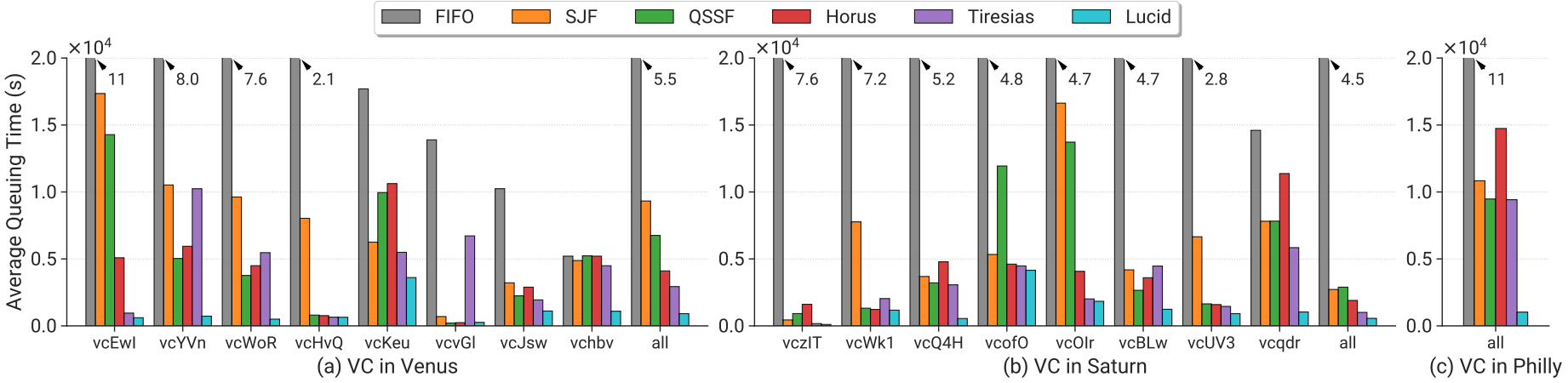


FIGURE 4.8: Average job queuing delay using different scheduling approaches across each VC, where *all* indicates the whole cluster.

TABLE 4.3: Comparison between physical experiments and trace simulation results regarding makespan and average JCT.

Scheduler	**Static (*Makespan*)**		**Continuous (*Avg. JCT*)**	
	Physical	**Simulation**	**Physical**	**Simulation**
FIFO	11.56 hrs	11.34 hrs	8.17 hrs	7.97 hrs
SJF	11.27 hrs	11.02 hrs	4.59 hrs	4.46 hrs
Tiresias	9.23 hrs	9.68 hrs	4.03 hrs	4.16 hrs
Lucid	**8.45 hrs**	**8.17 hrs**	**3.64 hrs**	**3.49 hrs**

Traces. To investigate the performance of Lucid on different job distributions and various cluster scales, we adopt three real production-level traces for comprehensive experiments, as summarized in Table 4.2. For two SenseTime traces, we use data from April-August as the training and validation datasets, and September data as testset for interpretable models. As for the Microsoft trace, we adopt the first week of October as testset and afterward (October-December) as the training and validation datasets. In order to reflect the actual effect of the scheduler in practice, we keep the original job submission traces without any rescaling or modification. According to the released cluster configuration, Saturn and Venus divide the clusters into 20 and 15 VCs respectively. Since Microsoft does not provide their VC configuration information, we set a reasonable cluster scale (108× 8-GPU nodes) without making further VC subdivisions. As for workload type, we refer to the GPU utilization distribution in Alibaba PAI [1, 2] and use a higher utilization trace for evaluation, as shown in Figure 4.11 (a, orange line) Venus-M. To be closer to reality, the long-term and large-scale jobs would be more likely large model training (e.g., BERT, ResNet-50 in Table 4.1) and vice versa. We apply hierarchical sampling to randomly assign each workload a job type derived from Table 4.1.

Baselines. We consider the following baselines.

(1) *First-In-First-Out (FIFO)*: a conventional policy widely adopted by many popular cluster management systems (e.g., Yarn [58], and Kubernetes [16]). It is simple but typically performs poorly due to its runtime-agnostic scheduling paradigm.

(2) *Shortest-Job-First (SJF)*: an ideal policy to minimize the average JCT without preemption by prioritizing short-term jobs to overcome HOL blocking. It is impractical as it requires perfect job information which is impossible to attain.

(3) *Quasi-Shortest-Service-First (QSSF)* (Chapter 2): a data-driven approach to prioritize short-term jobs through prediction. It achieves efficient scheduling without preemption but relies on a black-box ML model which is hard to troubleshoot.

TABLE 4.4: Performance comparison of different scheduling approaches across 3 clusters with regard to average JCT, queuing delay and tail delay. P99.9 indicates 99.9% percentile.

		FIFO	**SJF**	**QSSF**	**Horus**	**Tiresias**	**Lucid**
Average JCT (hrs)	*Venus*	18.57	5.86	5.15	4.41	4.09	**3.58**
	Saturn	14.21	2.36	2.41	2.13	1.89	**1.79**
	Philly	36.85	9.41	9.03	10.49	9.02	**6.84**
Average Queue (hrs)	*Venus*	15.30	2.59	1.88	1.14	0.82	**0.25**
	Saturn	12.61	0.76	0.80	0.53	0.28	**0.16**
	Philly	30.45	3.01	2.63	4.09	2.62	**0.29**
P99.9 Queue (hrs)	*Venus*	163.07	89.47	352.89	58.80	55.39	**26.15**
	Saturn	56.39	39.20	137.82	36.03	26.62	**19.28**
	Philly	117.55	101.60	125.57	223.47	98.80	**71.22**

(4) *Horus* [30]: a packing-enabled and data-driven policy that predicts job resource usage through model analysis. It is intrusive as it obtains ONNX [254] graph representation through user-code intrusion and relies on a black-box ML model.

(5) *Tiresias* [45]: a preemptive policy that prioritizes least attained service jobs (i.e., consumed GPU resources). Based on this design, short-term jobs are prone to finish earlier without any prior information. This is also intrusive as it requires user-code modification to achieve job preemption.

We also consider the state-of-the-art elasticity-based scheduler Pollux [56] and discuss its impact on model quality in §4.4.7. We do not evaluate its performance in large-scale traces (§4.4.3) due to its scalability issue. Specifically, it takes 30 minutes to handle a 160-job trace (used in their evaluation) and over 3 hours for a 320-job trace. It can not obtain the result within a reasonable time for our $10^5 \sim 10^6$ scale job traces.

4.4.2 End-to-End Evaluation on a Physical Cluster

To evaluate the performance of Lucid in practice, we conduct an end-to-end experiment on a physical testbed. To generate the real workload traces, we randomly sample jobs from the Venus trace. Specifically, we generate a 100-job *static* trace where all jobs are available at the beginning of the experiment, as well as an 120-job *continuous* trace where jobs are submitted following a Poisson distribution [46]. To evaluate the scheduling performance under different job distributions, the continuous trace samples more long-term jobs. Lucid profiles each job for at most 60 seconds and enables job packing in the following resource allocation. We compare Lucid against FIFO, SJF and Tiresias policies (Table 4.3). Lucid successfully improves the average JCT by 2.3× on the continuous trace and makespan by 1.4× on the static trace.

TABLE 4.5: Scheduling performance of large-scale (>8 GPUs) and small-scale (≤8 GPUs) jobs in Venus.

	Avgerage JCT (hrs)			**Average Queue (hrs)**		
	FIFO	**Tiresias**	**Lucid**	**FIFO**	**Tiresias**	**Lucid**
Large-scale Job	9.96	6.08	**4.59**	6.22	2.34	**0.86**
Small-scale Job	19.55	3.75	**3.46**	16.34	0.54	**0.19**

To verify the fidelity of our simulator, we further compare the results of physical experiments with simulations. We find the simulator can successfully reproduce the actual performance with an error rate $< 4.6\%$ on both makespan and average JCT. This demonstrates the high fidelity of our simulator.

4.4.3 End-to-End Evaluation on Large-scale Simulations

We use a simulator to assess the performance of Lucid on production-level clusters over weeks to months (Table 4.2).

Overall Performance. Figure 4.7 shows CDF curves of the average JCT in each cluster with different scheduling algorithms. It is evident that the Lucid curve almost overlaps with the curve of preemptive and intrusive baseline Tiresias for long-term jobs, but Lucid performs better for short-term jobs. This demonstrates the preemption-free policy can obtain comparable performance as the preemptive policy. From Table 4.4, Lucid improves the average JCT by up to 1.3× compared with Tiresias, saving 2.2 hours for DL training jobs on average.

Figure 4.8 presents the VC-level analysis of average job queuing delay across three clusters. We select the top-8 VCs with the highest average queuing time in Venus and Saturn since the other VCs have little delay. Besides, Philly is not partitioned in our experiment and thus has only 1 VC. We observe that Lucid presents stable performance across each VC, while Tiresias is inferior in some VCs (e.g., `vcvGI` in Venus). This derives from the high preemption overhead and redundant checkpoint-resume decisions of Tiresias. Table 4.4 shows Lucid achieves 1.8∼9.1× improvement on the average queuing delay compared with Tiresias.

To check the effect of job packing on the resource utilization, we sample the cluster-wide active GPUs every minute and record their average values. Compared with the sharing-agnostic policy Tiresias, Lucid obtains 9%∼17% GPU utilization and 7%∼24% GPU memory usage improvement.

Tail Performance. Most existing schedulers focus on improving the overall system performance while ignoring the worst cases. This may sacrifice partial jobs and cause

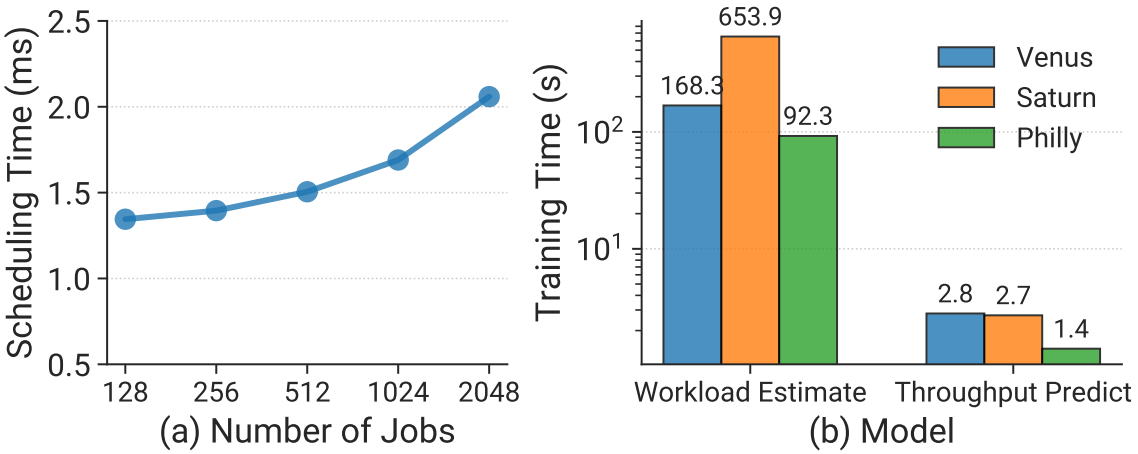


FIGURE 4.9: Scalability Analysis. (a) Scheduling latency (unit: ms) under various numbers of jobs. (b) Model training time (unit: s) across three clusters (y-axis in log scale).

unfairness. Table 4.4 provides the queuing delay of 99.9% percentile jobs for each algorithm. Lucid consistently outperforms Tiresias by 1.4∼2.1× across three clusters. The extraordinary tail performance of Lucid demonstrates its capability in handling long-tail and starvation issues.

Debugging Feedback. As mentioned in §1.2.3, there exist massive debugging and test jobs in production clusters. These jobs generally have very short duration and developers need timely feedback to modify their codes accordingly. This can be successfully achieved based on the profiler design of Lucid. Compared with Tiresias, Lucid greatly mitigates the number of queuing short-term jobs (≤60s) by 4.1∼24.8×, which efficiently improves user experience.

Fine-grained Analysis. To evaluate the scheduling effect on different scale workloads, we summarize their average JCT and queuing delay in Table 4.5. Lucid obviously outperforms Tiresias for both large and small jobs, which demonstrates large jobs will not experience starvation in Lucid scheduling.

4.4.4 Scalability Analysis

For practical deployment of DL schedulers, it is significant to consider their scalability to handle massive workloads and large-scale cluster resources.

Scheduling Latency. We have successfully performed the end-to-end evaluation of Lucid across three production-level clusters with thousands of GPUs and long-term traces as shown in Table 4.2. According to our experiment records, the average job queue length is 10∼12 and the maximum length is 119∼340 among these clusters. As shown in Figure 4.9 (a), we measure the scheduling decision latency under more

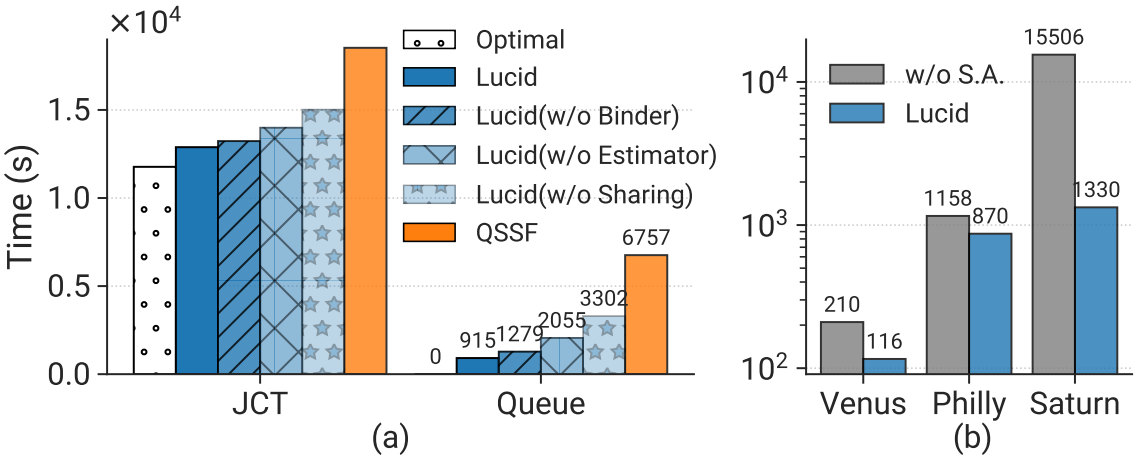


FIGURE 4.10: Ablation Study. Effect analysis of (a) binder and estimator; (b) space-aware profiling (S.A.), y-axis in log scale.

intensive job quantities, where the inference latency of Lucid models is included. Even given 2048 jobs, the job allocation policy can be obtained within 3 ms, which is sufficient for DL job scheduling. Conversely, when dealing with 2048 jobs, Gavel [46] needs to take around 30 minutes to solve the linear programming problem [225]. Shockwave [220] and Muri [212] also take seconds to minutes overhead on solver computation. Compared with Lucid real-time scheduling, round-based paradigm and excessive decision time seriously limit their deployment.

Training Overhead. In addition to short scheduling latency, the ML model re-training overhead is another concern for system application in practice. Lucid adopts *Update Engine* to collect the latest data and update models periodically (e.g., daily or weekly). Figure 4.9 (b) depicts the training time of *Workload Estimate Model* and *Throughput Predict Model*, where the training set contains $10^5 \sim 10^7$ samples across three clusters within half year. Owing to our transparent and simple model designs, even dealing with million-scale training data, it only takes up to 11 minutes to obtain the model. Besides, *Packing Analyze Model* is cluster-agnostic and only takes less than 1 second for training. The low decision latency and training overhead verify the scalability of Lucid.

4.4.5 Micro-benchmarks

We explore the effects of each component in Lucid via ablation studies, and perform sensitivity analysis of workload and system.

Impact of Binder. To examine the effect of *Affine-jobpair Binder* introduced in §4.3.3, we compare and measure the performance of Lucid when disabling *Indolent Packing* (w/o Binder) or job packing (w/o Sharing) on the Venus cluster. As shown in

TABLE 4.6: Sensitivity Analysis of Profiling Time Limit T_{prof}.

	Profiling Stage		**Overall**	
	Finish Rate	Queuing Delay	JCT	Queuing Delay
$T_{prof} = 100$	27.65%	21	13,087	1,074
$T_{prof} = 200$	44.61%	73	12,886	915
$T_{prof} = 300$	53.73%	175	13,160	1,222
$T_{prof} = 600$	64.40%	509	13,270	1,422

Figure 4.10 (a), *Indolent Packing* can deliver additional 1.4× queuing delay reduction compared with the naive bin-packing policy. When job packing is totally disabled, Lucid can still obtain over 2.0× reduction in queuing delay compared with the SOTA non-intrusive QSSF. This superiority derives from the unique profiling design and accurate job duration estimation.

Impact of Estimator. We further evaluate the benefit of workload duration estimation in *Resource Orchestrator*. As shown in Figure 4.10 (a), we disable the estimator-based optimization (w/o Estimator) in both the job binder and orchestrator stages. It is obvious that job runtime-awareness further reduces 2.2× job queuing delay compared with the runtime-agnostic job sharing method. On the other hand, the variant Lucid (w/o Estimator) still outperforms QSSF owing to (1) Lucid profiler design efficiently prioritizes massive short-term jobs to finish first, which greatly reduces the average queuing delay; (2) Lucid binder still can pack training jobs with low GPU utilization, which takes the majority (Figure 1.3). Moreover, we also depict the Optimal upper bound (all jobs without any queuing, equals to average JCT of FIFO/SJF/QSSF minus their corresponding average queuing delay) of non-intrusive schedulers with white dotted bar in Figure 4.10. It is clear that the combination of all modules in Lucid delivers close to optimal performance, as if there were no queuing delays.

Impact of Profiler. We also investigate the influence of *Non-intrusive Job Profiler* (§4.3.2). Based on the *two-dimensional* profiling strategy, most jobs will be profiled while 23.3%∼55.4% jobs finish early during the profiling stage across three clusters. Besides, the average queuing delay in each profiling cluster is around 1 minute, indicating the profiler can handle most jobs with no severe latency. Figure 4.10 (b) further shows the effect of *Space-aware Profiling* (S.A. in short, y-axis represents queuing time). To conduct fair comparison, we disable the *Time-aware Scaling* mechanism and set T_{prof} to 500s and N_{prof} to 36 for each cluster. The space-aware approach can provide up to 11.6× improvement compared with the naive profiling mechanism adopted in other works [42, 45, 226].

Sensitivity Analysis of Workload Distribution. One major concern of Lucid is whether it only applies to low cluster-wide GPU utilization scenarios. Figure 4.11

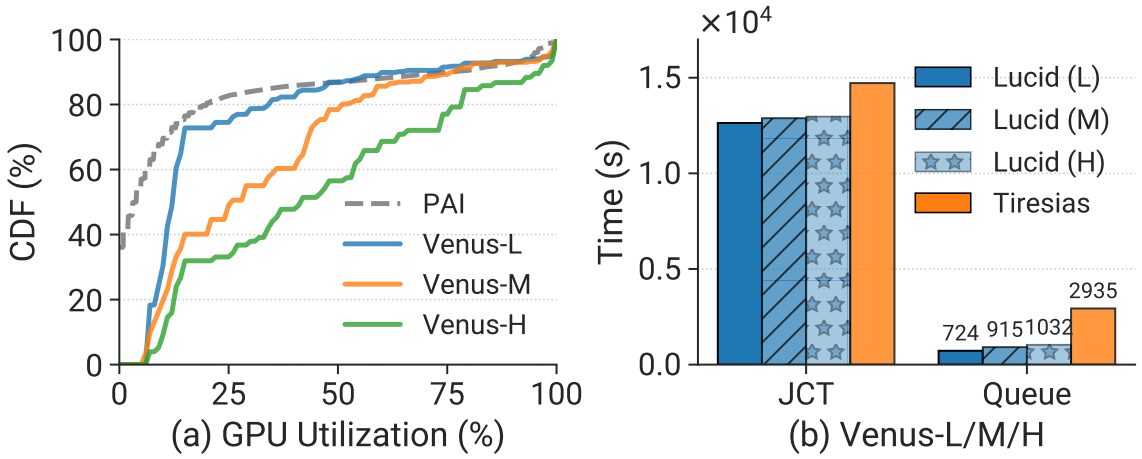


FIGURE 4.11: Sensitivity Analysis. (a) GPU utilization distributions of Alibaba PAI cluster [1, 2] and generated Venus traces with Low/Median/High utilization. (b) Lucid scheduling performance under various workload distributions.

(a) shows the GPU utilization distribution of an Alibaba cluster (i.e. PAI, gray line) in practice. We generate three types of traces for evaluation: Venus-M is applied in our end-to-end experiments (§4.4.3); Venus-L is designed to mimic the Alibaba cluster utilization scenario; Venus-H represents a high GPU utilization trace. As shown in Figure 4.11 (b), even if all three traces are heavier than PAI, Lucid obtains better scheduling performance (1.8∼4.2× in queuing delay reduction) compared with Tiresias. This verifies Lucid can maintain excellent performance in various scenarios.

Sensitivity Analysis of System Configuration. System hyperparameters can affect scheduling performance. To this end, we explore Lucid's sensitivity to T_{prof} (profiling time limit), binder thresholds and model update interval. **(1)** T_{prof}. Table 4.6 shows the scheduling performance under different T_{prof} settings (100s∼600s) in Venus. We observe that the higher T_{prof} allows more job completion but also incurs longer queuing delays during the profiling stage. It affects profiler behaviors a lot but performs stable on overall JCT. We set the default value of T_{prof} as 200s because the time is sufficient for most job profiling and will not incur a heavy queuing delay in the profiler. **(2) Binder Thresholds**. The thresholds for (Medium, Tiny) jobs are heuristic knobs adjustable by system operators. Operators can set lower thresholds for higher cluster efficiency, or higher thresholds for less interference. We try several reasonable settings by varying Medium (0.75∼0.85) and Tiny (0.90∼0.97), and find the average JCT is robust (<3.6% difference) in Venus. It is because Lucid *Indolent Packing* strategy can efficiently prioritize non-interference jobs and lightweight jobs occupy the majority. We choose (0.85, 0.95) as the default value because it can well balance job packing opportunity and interference. **(3) Model Update Interval**. Compared with the static model without any update, Lucid periodical model update (weekly) can reduce queuing delay by 4.8% in Venus September evaluation period.

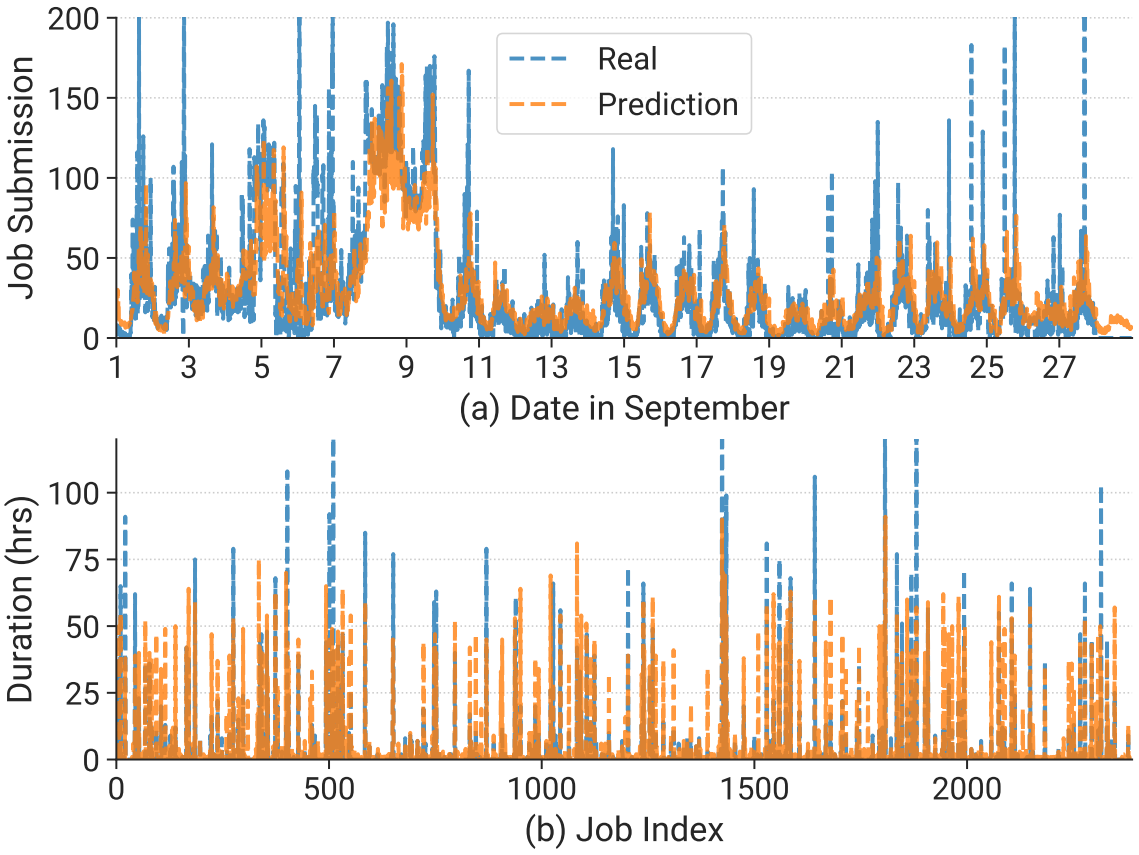


FIGURE 4.12: Prediction Visualization. (a) *Throughput Predict Model* for job submission prediction in Saturn. (b) *Workload Estimate Model* for job duration estimation in Venus.

TABLE 4.7: Model Performance. Lucid outperforms popular black-box models across *Throughput Predict Model* (MAE) and *Workload Estimate Model* (R^2 score) in Venus.

Models	RF	LightGBM	XGBoost	DNN	Lucid
Throughput Predict	4.607	4.491	5.807	5.132	**4.125**
Workload Estimate	0.101	0.230	0.332	0.181	**0.413**

More frequent updates (daily) can bring an additional 1.6% improvement. Weekly update interval typically is sufficient in most scenarios to update workload information at a low maintenance cost.

4.4.6 Interpretable Model Evaluation

Since Lucid is a learning-augmented DL job scheduler, the performance of ML models is critical to the scheduler. For system transparency and simplicity, we apply interpretable models for all the prediction tasks (§4.3.5).

Model Performance. Figure 4.12 (a) presents cluster-wide job throughput prediction on Saturn September. We observe that our prediction can precisely reflect the actual trend with small estimation errors, which laid the foundation for dynamic system scaling and tuning. Figure 4.12 (b) depicts Lucid duration estimation on each job in Venus. Due to too many jobs, we randomly sample 10% jobs for clearer visualization. It is evident that Lucid can well distinguish long-term and short-term jobs, although there exist some gaps between actual duration and final prediction. Our experiment demonstrates such performance is sufficient for providing good scheduling decisions.

Many researchers have the prejudice that there exists a trade-off between accuracy and interpretability. In fact, interpretability often begets accuracy, and not the reverse [233]. We provide comprehensive evaluations of Lucid models with a series of popular ML algorithms: Random Forest (RF) [255], LightGBM [86], XGBoost [256] and DNN [257]. We use the default hyperparameters for baseline algorithms. Table 4.7 presents MAE (Mean Absolute Error, lower is better) scores of *Throughput Predict Model* and R^2 (Coefficient of Determination, higher is better) score of *Workload Estimate Model* job duration estimation in Venus. We find our models deliver the best performance, bringing better scheduling policy and cluster performance. For relative simple ternary classification task of *Packing Analyze Model*, DT is sufficient to provide equivalent accuracy (94.1%) with other more complex baselines.

System Adjustment. Lucid provides simple and intuitive explanations for system tuning. Based on guided tuning, we adjust the configurations of *Non-intrusive Job Profiler* according to the trace data of the previous month. Compared with heuristic tuning results, it reduces the average queuing delay at the profiling stage by 2.8∼8.7× with negligible influence on job filtering and debugging feedback. For the model troubleshooting, we pose monotonic constraint on the `gpu_num` feature in *Workload Estimate Model*, which obtains 2.6% R^2 score improvement and reduces 3.9% queuing delay.

4.4.7 Comparison with Elastic Scheduler

We further compare Lucid with the state-of-the-art elastic scheduler Pollux [56] under increasing workload intensity in terms of the rate of job submissions. We use the author-provided traces for evaluation, where intensity=1.0 represents 160 jobs in total. Figure 4.13 (a) presents the results that Lucid can deliver better performance when the workload becomes more intensive. Pollux is more suitable for lighter workload intensity because its adaptive job batch size and resource scaling techniques are limited when clusters are overloaded. More importantly, Pollux cannot guarantee no accuracy degradation for all models while Lucid can well preserve model quality as shown in

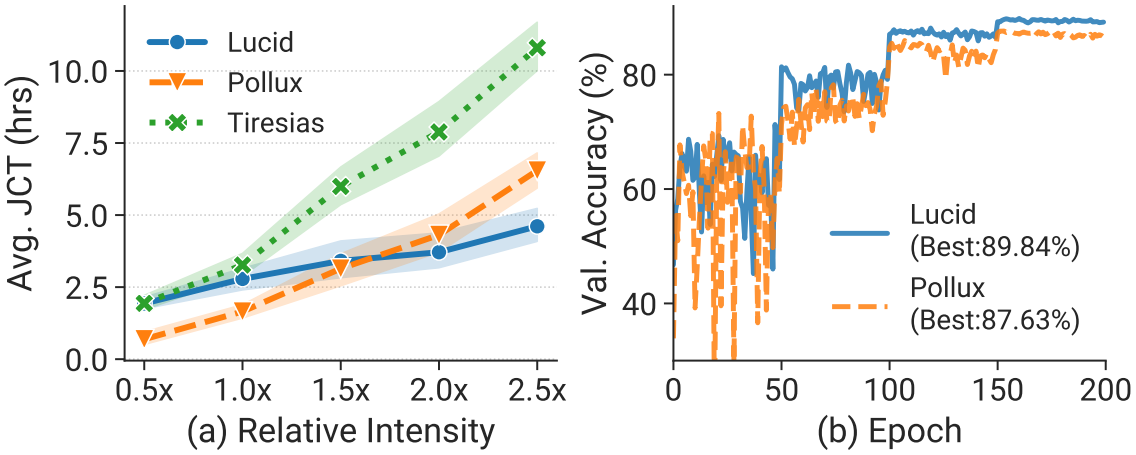


FIGURE 4.13: Comparison with Pollux. (a) Average JCT under various workload intensities. (b) Validation accuracy of an EfficientNet job with (Pollux) or without adaptive training.

Figure 4.13 (b). Pollux induces over 2% accuracy decrease in EfficientNet training which is often unacceptable in practice (**G3**) [220].

4.4.8 Takeaways

Lucid exhibits excellent performance in our extensive evaluations. We summarize some key points that could improve the scheduler performance and hope to inspire future scheduler design.

- **Workload awareness — Profiler**. Existing works [42, 45, 226] typically regard retrieving job runtime information as the only function of the profiler. However, because short-term jobs take the majority of DL workloads, we find that the profiling mechanism works well on such workload distribution, which will not incur huge extra queuing delays or resource demands. Based on our profiler design, most debugging jobs are filtered during the profiling stage, which significantly facilitates the scheduling optimization by diminishing the optimization space. Besides, Lucid can deliver better duration estimation compared with QSSF based on additional profiled features.

- **Resource awareness — Binder**. Many works, like Tiresias, ignore the opportunity of leveraging underutilized GPUs. Lucid provides an interference-aware job packing mechanism in a non-intrusive way that efficiently improves resource utilization and reduces job queuing (Figure 4.10). Besides, Lucid realizes the resource demand changes over time, thus dynamically adjusting the packing strategy and profiling resource scale to improve cluster efficiency.

- **Runtime awareness — Orchestrator**. Based on our observation that a majority of workloads have recurrent patterns and users tend to submit similar tasks multiple times, Lucid can provide job runtime estimation to optimize the scheduling plan. On the contrary, Tiresias (i.e., Discretized Least Attained Service) adopts runtime-agnostic scheduling (FIFO in each queue), which can incur frequent superfluous preemption. The job checkpointing and cold-start overhead are also high, which takes 62 seconds per preemption on average [45]. According to our evaluation in Venus, preemption causes an additional 13% queuing overhead.

4.5 Related Work

DL Job Schedulers. Schedulers tailored for DL training workloads have been actively researched in recent years [1, 18, 23, 30, 43, 45, 56, 215] and many of them adopt job packing to improve resource utilization. Gandiva [18] leverages online-profiling to introspectively determine whether to co-locate jobs on an accelerator. AntMan [1] enables more fine-grained GPU sharing with dynamic scaling techniques. Salus [31] implements two primitives *fast job switching* and *memory sharing* for more efficient GPU sharing. Horus [30] converts user models into ONNX [254] graph representations and extracts workload features to determine job packing. Distinct from these works, Lucid supports job packing in a non-intrusive scheduling paradigm.

Beyond GPU sharing, Gavel [46] and Gandiva$_{fair}$ [44] focus on leveraging the heterogeneity of GPU generations to improve resource utilization. CODA [258] designs a feedback-based adaptive CPU allocation algorithm for DL training jobs. Similarly, Synergy [227] allocates CPU and memory resources according to the workload sensitivity to these resources. Muri [212] exploits multi-resource interleaving to improve resource utilization and reduce JCT. Lucid currently only considers homogeneous GPU as the dominant resource. Inspired by these novel works, we believe Lucid can be extended to support heterogeneous GPU and affiliated resource (e.g., CPU, networking) scheduling optimization in the future.

Prediction-based Schedulers. Conventional cluster management systems [16, 58, 76] collect job runtime estimations provided by users to schedule workloads, which is inaccurate and often results in cluster inefficiency. Prior works leverage historical job information to predict job durations and optimize scheduling decisions. The prediction can base on the recurrent jobs [69, 82–84], or job structure knowledge [77–81]. For more general cases, some schedulers [71, 85, 101] make the prediction from the history of relevant jobs. In DL clusters, Helios (Chapter 2) characterizes SenseTime workloads and finds that using a LightGBM [86] model to predict job duration can improve scheduling performance. MLaaS [2] also notices the prevalence of recurring jobs in

Alibaba and uses Decision Tree to predict job duration, delivering less than 25% prediction error for 78% instances. Lucid further leverages profiled features to enhance prediction precision and considers its interpretability.

Interpretability of Systems. Interpretability is important for users to trust ML model behavior and deploy ML-driven systems. Metis [34], DeepAid [259] and Lemna [232] design toolkits to improve system transparency by interpreting black-box ML models. Furthermore, Unicorn [260] adopts causal inference to find effective repairs. In recent resource management research, Sinan [261] employs LIME [262] to identify important features of its hybrid model and Sage [263] dedicates to performance degradation reasoning of microservice. In contrast to them, Lucid adopts Primo [37] framework which directly builds interpretable models instead of putting effort to understand the black-box process.

4.6 Summary

In this chapter, we propose Lucid, a non-intrusive deep learning workload scheduler based on interpretable models. Specifically, we design a *two-dimensional* optimized profiler and *indolent packing* strategy for efficient job metric collection and interference avoidance. Besides, Lucid orchestrates resources based on estimated job priority values and promotes model performance maintenance. Compared with the state-of-the-art intrusive scheduler Tiresias (obtains an average job completion time of 9.02 hours on Microsoft trace), our experiments demonstrate that Lucid successfully reduces it to 6.84 hours, which is 1.32× better.

Chapter 5

Primo: Building Interpretable Learning-Augmented Systems

5.1 Introduction

Over the years, machine learning (ML) has been widely adopted to optimize systems across many fields, e.g., storage [247, 264, 265], network [266–268], security [269–271], compiler optimization [181, 272, 273] and cluster scheduling [261, 274, 275]. These learning-augmented systems demonstrate marvelous performance compared with conventional heuristic or mathematical optimized systems.

However, most of these applied models are very complex and treated as black-boxes to developers, which brings significant gaps in deploying them in practice. **First, building a production-level learning-augmented system can incur huge costs.** From the experience at Microsoft [33], the model training process could take days to weeks with massive data. Some systems require frequent model updates to adapt to dynamic environment changes, whose cost often exceeds enterprise expectations. For some scenarios with limited data samples, developers have to use techniques to synthesize training samples [276–278], which inevitably introduce bias to the model and cause performance deterioration in practice [266, 273, 279]. Moreover, the inference process of these complicated models can pose heavy computational pressure to systems which have high real-time requirements [265, 280, 281], which can significantly restrict parallel capabilities and affect scalability in practice.

Second, the prediction process of these black-box models are unintelligible to humans. Developers lack understanding and trust of the model's behavior [282–284], which makes it difficult for them to perform model adjustments and ad hoc debugging in practical scenarios. Some efforts have been made to improve system transparency through interpreting black-box models [34, 232, 259]. They typically

build *surrogate* models to obtain explanations for individual predictions, thus validating model behaviors and diagnosing system mistakes. However, they cannot provide an interpretation fidelity guarantee, and therefore the corresponding explanations are unreliable and potentially misleading [233, 285]. In addition, they cannot address the aforementioned system cost issue.

In this work, we aim to resolve the above challenges and facilitate *transparent, accurate* and *lightweight* system deployment in practice. We introduce Primo (**Pr**ior-based **i**nterpretable **m**odel **o**ptimization), the *first* unified framework that assists developers to design and optimize learning-augmented systems with interpretable models. The design of Primo is based on two key insights. First, *simple interpretable models have the capability of handling complex system problems.* Interpretable models do not sacrifice prediction accuracy [250, 286, 287], and simple model structures with low resource overhead are very suitable for real-time systems. Their effectiveness is often underestimated [233]. Second, *prior experience and domain knowledge can be leveraged by developers to further optimize the interpretable models* [133, 137], which is hard to achieve for black-box models.

Primo makes several innovations to enhance learning-augmented systems. First, to provide comprehensive support for different systems, Primo introduces two interpretable model algorithms: `PrAM` is designed for better prediction accuracy and `PrDT` applies to systems with strict latency or computation constraints. Primo can help developers select a suitable model *automatically* based on their system requirements, including latency, accuracy, and resource budget. In addition to training models directly, Primo also supports distilling existing complex models, which applies to exploration-based systems with reinforcement learning (RL) [288–291].

Second, to fully exploit the potential of interpretable models, we design several built-in mechanisms to optimize model performance leveraging *prior* information. (1) Primo implements *Bayes Optimization* (BO) to find the optimal model pruning strategy and hyperparameter configurations for higher prediction accuracy and lower computation overhead. It fully takes advantage of prior search information to minimize the search space and training cost. (2) Primo also facilitates model post-processing for developers with their domain knowledge. Specifically, it provides two tools for model adjustment through adding *Monotonic Constraints* and transparent debugging with *Counterfactual Explanations*.

Based on these innovations, Primo provides not only precise and comprehensive interpretations for developers to understand and adjust models, but also better prediction accuracy and smaller overhead. To extensively evaluate these benefits in real scenarios, we apply Primo to three state-of-the-art learning-augmented systems, including two *online* systems (LinnOS for flash storage [247] and Pensieve for video steaming [292]), and an *offline* system (Clara for SmartNIC offloading [266]). For LinnOS,

TABLE 5.1: Comparisons of different strategies for learning-augmented systems (☆: Performance improvement).

System Strategies	Interpretation Fidelity	Local Interpretation	Global Interpretation	Transparent Adjustment	Cost	Accuracy ↗	Roustness ↗	Latency ↘
Black-box models (e.g., *DNN, RL*)	✘	✘	✘	✘	$$$	★	★	★
Interpreting black-box models [232, 262]	✘	✔	✘	✘	$$$	★	★	★
Building interpretable models (Primo)	✔	✔	✔	✔	$	★☆	★☆	★☆

Primo provides a 2.8× system performance improvement, and reduces model training time by over 100×, as well as inference latency by over 20×. For Clara, Primo beats a series of black-box models in prediction accuracy and saves over 10× training cost. For Pensieve, Primo achieves better generalization ability and a 79× inference latency reduction. We believe Primo can bring similar benefits to other learning-augmented systems as well.

To summarize, we make the following contributions:

- To the best of our knowledge, Primo is the first framework to provide inherent interpretability for learning-augmented systems development.
- We design built-in mechanisms and adjustment tools for developers to achieve *transparent*, *accurate* and *lightweight* system deployment in practice.
- For the first time, we demonstrate that simple interpretable models can outperform complex black-box techniques in various real systems.

5.2 Background and Motivation

5.2.1 Learning-Augmented Systems

Learning-augmented systems apply machine learning techniques to optimize system performance [33]. They typically build various ML models to obtain preeminent system policies from historical execution data, such as Support Vector Machines (SVM) [266, 274], Random Forest (RF) [293, 294], Gradient Boosting Decision Tree (GBDT) [23, 261]. With the popularity of deep learning (DL) algorithms, they were also introduced to further enhance systems, e.g., Deep Neural Network (DNN) [247, 265], Convolutional Neural Network (CNN) [281, 283], Recurrent Neural Network (RNN) [266, 295] and Reinforcement Learning (RL) [292, 296]. We classify them into the following categories.

Taxonomy. Learning-augmented systems typically follow similar design workflows to integrate ML models into system operations. Based on the optimization type, they can be classified into two categories. (1) **Prediction-based** systems utilize the *supervised learning* paradigm (e.g., classification, regression) to optimize system problems. (2) **Exploration-based** systems usually adopt *reinforcement learning* to learn optimal policies in an explore-exploit way. Since there are relatively fewer unsupervised learning-based systems in practice, we consider them as our future work (§5.8).

Based on system requirements and application scenarios, learning-augmented systems can be divided into the following two types. (1) **Online** systems require the ML model to make prompt predictions for real-time data. Developers need to consider model inference latency and computation overhead, in addition to prediction accuracy. (2) **Offline** systems usually do not need to deploy ML models for real-time serving and have no latency or computation requirements. These systems are performance-critical and the objective of ML models is to improve prediction accuracy.

Primo is designed as a unified framework, providing respective optimization mechanisms for different types of learning-augmented systems.

5.2.2 Challenges and Motivation

While plenty of work has demonstrated the potential of ML techniques in improving system performance, there exist several challenges in the development and deployment of learning-augmented systems in practice.

Model development. First, building a qualified ML model for the target system has the following two challenges:

- **C1: high training and tuning cost.** As stated by Microsoft AutoSys [33], costs of ML model training often exceeds enterprise expectations. Real system environments are dynamically changing and stale models will cause performance deterioration. Therefore, frequent model fine-tuning or retraining is necessary, which could take days to weeks [33, 297] in order to outperform heuristic algorithms. If there are not enough GPU resources, the update time will become even more intolerable for DL models.
- **C2: susceptible to the quantity and quality of data.** A large amount of high-quality training data are essential to produce satisfactory ML models. However, in some cases, insufficient data [266, 273, 279] or excessive data collection cost [297, 298] hinder developers from training qualified models. Possible solutions include data augmentation and synthesis [276–278]. Nevertheless, owing to the sophisticated distribution of real-world data, the generated data inevitably introduce

bias and shift to the learning model [266], which could compromise the system performance in practice.

Model deployment. Second, deploying ML models in practice has interpretability and inference overhead issues:

- **C3: opaque decision making process.** Developers mainly focus on improving key system metrics (e.g., I/O latency [247], user experience [292]) when designing and evaluating ML models, while ignoring their *interpretability*. As a result, most of these learning models are *black-boxes* whose prediction processes are unintelligible to humans [232–234]. Due to such opacity, system operators cannot guarantee model predictions are risk-free and have insufficient confidence to deploy them.
- **C4: difficulty in troubleshooting and adjustment.** In order to achieve expected performance in production environments, system operators typically need to adjust the learning models according to the actual scenarios [34, 283, 284], including input features alteration, model structure modification, data augmentation, etc. All these actions require the operators to have a profound understanding of the system and the corresponding ML technique [33, 232], which is difficult when the model is complex. In addition, ad hoc debugging is another substantial challenge to learning-augmented systems for black-box models. Improper modifications may cause severe performance degradation.
- **C5: exorbitant deployment overhead.** The model deployment overhead is another key factor for system operators' consideration [283]. The latency and computation requirements of some systems [265, 280, 281] are far more strict than conventional AI tasks. High inference overhead can cause side effects to production workloads and limit their parallelism capability [247], which can further restrict deployment scalability.

5.2.3 Existing Solution: Interpreting Black-box Models.

One possible solution to address the above challenges is model interpretation. The essential idea is to leverage existing interpretation methodologies to interpret the black-box models, making them more intelligible and transparent. A variety of interpretation tools (e.g., Lime [262], Captum [299], Shap [300]) were designed to explain the mechanisms of DNN models for CV and NLP tasks. Similar studies were also performed for other domains. For instance, Lemna [232] employs a mixture regression model [301] to interpret RNN models in DL-based security applications. Metis [34] proposes to interpret networking systems with the decision tree or hypergraph. However, we argue that the idea of interpreting black-box models is not sufficient for learning-augmented systems for the following reasons.

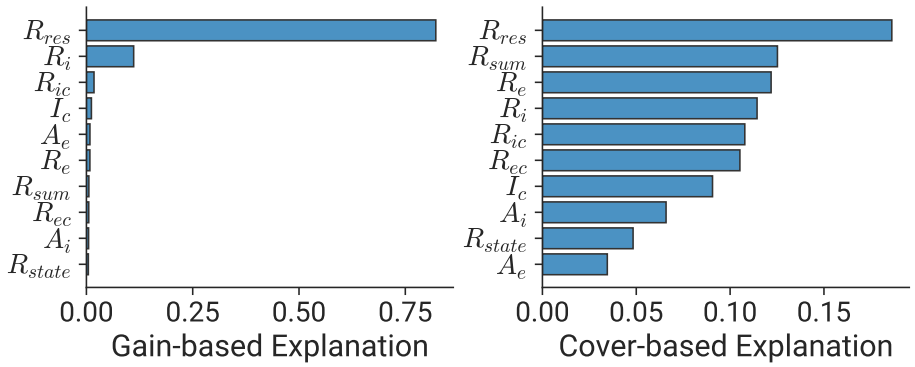


FIGURE 5.1: Interpretations of XGBoost model for Clara-MS task based on different metrics. Higher value indicates the feature is more important.

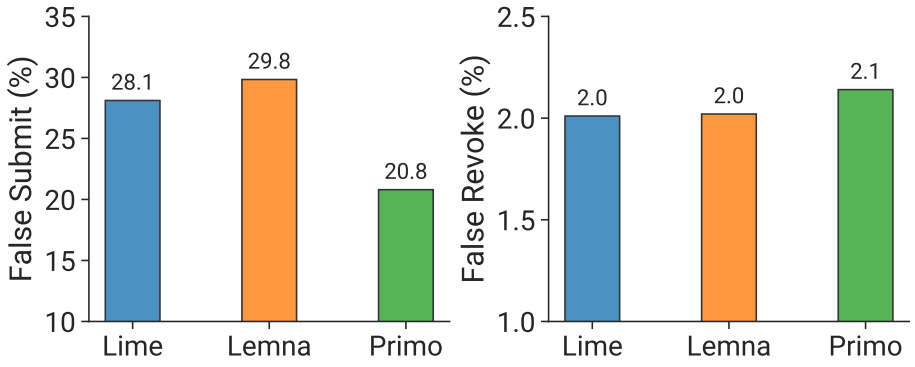


FIGURE 5.2: Comparison of Primo global surrogate performance with Lime and Lemna in the LinnOS scenario.

(1) ***No fidelity guarantee***. These tools typically interpret black-box models in a *post hoc* way, where another *local surrogate* model is created to explain the original model. They cannot have a fidelity guarantee with respect to the original model. Therefore, the corresponding explanations are often unreliable, and can be misleading [233, 285]. The fidelity of some widely applied interpretation methods (e.g., attention-based explain[302]) are still in dispute [303, 304]. We present an example of contradictory XGBoost explanations.

Clara adopts XGBoost to predict the optimal core counts for different NFs in Clara-MS. XGBoost contains a built-in api `get_score` to get the feature importance value of each feature and it can be regarded as the model interpretation. However, we argue that the interpretation has low fidelity. As evident from the Figure 5.1, interpretations based on different metrics present contrasting patterns, where both *gain* and *cover* are important intermediate metrics during model generation. For instance, R_{sum} is the second important feature according to *cover*-based explanation while seems ignorable from *gain*'s perspective. This makes developers feel confused which interpretation should trust. More seriously, this could mislead them to make wrong decisions, such as incorrectly omitting important input features while feature engineering.

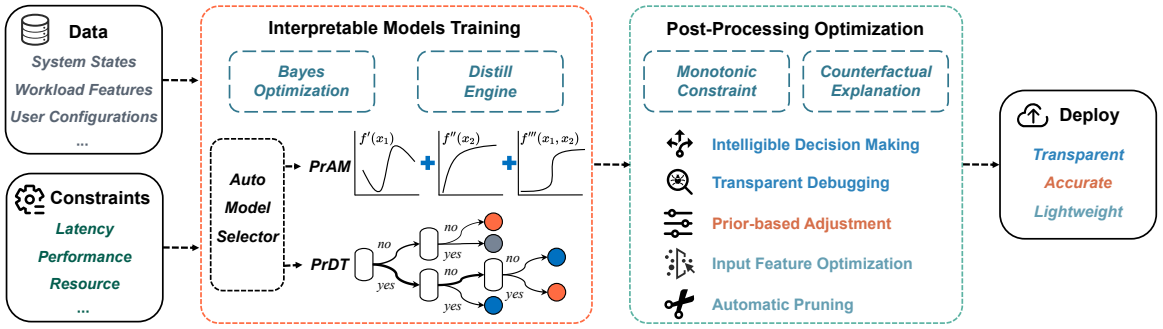


FIGURE 5.3: The workflow of learning-augmented system development using Primo.

(2) ***Limited interpretation.*** Most existing tools (e.g., Lime, Lemna) focus on explaining individual predictions (local interpretation) instead of the entire model behavior (global interpretation). Thus, the interpretation results typically cannot yield enough information for system troubleshooting. A common question is that *If using the existing interpretation tools for the model surrogate, can they also deliver equivalent effects?* Our answer is no. We evaluate two popular black-box interpretation methods: Lime [262] and Lemna [232]. Lime performs local interpretation through linear regression of data subset and Lemna adopts a mixture regression model to obtain interpretation in a similar way. Both of them are designed for local interpretation of several instances. However, we further explore whether they have the potential for the global model surrogate. Hence, we train the Lime and Lemna model on the LinnOS dataset, and compare the performance with Primo. As shown in Figure 5.2, Lime and Lemna cause much higher (7.3%∼9.0%) false submit rates, which is the most significant metric for the system performance. As stated in §5.4, the overhead of failover (false revoke) is relatively negligible and all three methods perform well pertaining to this metric. Overall, existing interpretation tools are insufficient for the global model surrogate.

(3) ***Requiring domain knowledge.*** Different systems may employ different models and algorithms. There is no unified tool that can provide comprehensive support for interpreting arbitrary models. Consequently, domain knowledge and manual efforts are required to implement the tools and understand the explanation results. This poses a huge challenge for developers to design a learning-augmented system.

(4) ***Incapability of handling other challenges.*** Those tools only focus on model interpretation and understanding (**C3** & **C4**), but ignore other challenges discussed in §5.2.2.

5.2.4 Our Approach: Building Interpretable Models

A more promising direction, which is adopted in Primo, is to train *interpretable models* directly for learning-augmented systems. Interpretable models refer to the models that are inherently intelligible, where their explanations provided by themselves are faithful to what the models actually compute [233, 285]. Common interpretable models include linear regression, logistic regression, decision tree, decision list, etc. They have great potential to enhance different types of learning-augmented systems.

According to our observation from recent state-of-the-art learning-augmented systems [305], the scale of models in these systems tend to be relatively smaller than popular production-level AI models (although they are still too complex for humans to understand). For instance, the number of neurons in a RL-based system is typically less than 10K [284]. This is because most data samples in learning-augmented systems are well structured, with good representations in terms of naturally meaningful features. In such scenarios, a much simpler interpretable model can give comparable performance to complex black-box models [233]. Therefore, developers can employ interpretable models for their systems, which require less data, training and tuning cost (**C1** & **C2**). The models give more information for system operators to understand (**C3**), troubleshoot and adjust (**C4**), and the inference speed is much faster than the original black-box model (**C5**).

Summary. The benefits of Primo compared with other methods are summarized in Table 5.1. It can provide not only highly precise and comprehensive interpretations for developers to understand and adjust models, but also higher accuracy and robustness, and smaller training and inference overhead. These greatly facilitate model deployment in practice.

5.3 Primo Design

We introduce Primo, a unified framework that assists developers to design practical learning-augmented systems. Particularly, (1) we employ *transparent* and *deterministic* interpretable models to circumvent the uncertainty issues of black-box model inference. (2) We integrate new tools for developers to leverage prior knowledge to optimize interpretable models *automatically*. (3) We design a built-in mechanism to search optimal hyperparameters in a *fast* and *convenient* way, without extra effort from the developers. Based on these designs, Primo can address all the challenges in §5.2.2.

5.3.1 Framework Overview

Primo optimizes both the *training* and *post-processing* stages of building learning-augmented systems. Figure 5.3 illustrates the development workflow with Primo. In the model training stage, Primo provides two interpretable model algorithms (`PrAM` and `PrDT`) designed for different system scenarios[1]. Primo helps developers automatically select suitable algorithms based on their system requirements including latency, accuracy, and resource budget. It supports training the interpretable model directly, or converting an existing complex black-box model into a simple interpretable model through the *Distill Engine*. We also leverage *Bayes Optimization* to find the optimal model pruning strategy and hyperparameter configurations for higher prediction accuracy and lower computation overhead. After the model is trained, Primo offers several optimization tools in the post-processing stage, e.g., prior-based model adjustment through adding *monotonic constraints*, transparent debugging with *counterfactual explanations*. Below we detail the mechanism of each component.

5.3.2 Interpretable Models

As introduced in §5.2.1, different system scenarios have different requirements for the learning models. To this end, Primo employs two types of interpretable model algorithms: `PrAM` is designed for better prediction accuracy and `PrDT` applies to systems with strict latency constraint or computation sensitivity. Primo supports automatic model selection based on the demands specified by the developers.

`PrAM`: Addictive Model based Method Our first interpretable model, `PrAM`, is based on the Standard Generalized Additive Models (GAMs) [306]. GAMs consist of a series of *shape functions* $f_i(\cdot)$ and an intercept μ_0 (Equation 5.1). Since each shape function considers only one univariate term (the ith feature $\boldsymbol{x}^i$) and their combination is additive, GAMs are interpretable: we can clearly understand the contribution of each single feature to the final prediction.

Compared with linear interpretable models (e.g., logistic regression), GAMs can cope with more complex prediction tasks because shape functions are typically nonlinear and have better fitting capability. To further increase model performance, we adopt the state-of-the-art GAM algorithm: GA^{2}M [249], which additionally considers the interactions of two features and maintains the interpretability. GA^{2}M has the following form:

[1]Other interpretable models can also be conveniently integrated into this framework, which will be considered in our future work.

$$g(E[y \mid \boldsymbol{x}]) = \underbrace{\mu_0 + \sum f_i\left(\boldsymbol{x}^i\right)}_{\text{GAM}} + \underbrace{\sum f_{ij}\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right)}_{\text{Interactions}} \tag{5.1}$$

where $g(\cdot)$ is a link function that adapts GA^{2}M to different tasks, e.g., regression (identity), classification (logistic function); $f_{ij}(\cdot)$ represents the interaction effect of features i and j, which can be visualized as a two-dimensional heatmap.

In our implementation, `PrAM` extends the open-source library EBM [307] to obtain the optimal model with high compactness and accuracy. Explainable Boosting Machine (EBM) [307] is a open-source implementation of Generalized Additive Models plus Interactions (GA^{2}M) [249] written in C++ and Python. Similar to popular GBDT algorithms (e.g. LightGBM [86]), the EBM training procedure begins by bucketing data from continuous features into discrete bins of histogram [250], which can significantly accelerate model training. Then EBM starts to learn shape function $f_i(\cdot)$ for each feature. Common choices for shape functions are regression splines, step functions and binary trees. For better prediction accuracy, it chooses the boosted trees, where each successive tree tries to predict the overall residual from all the preceding trees [308]. Furthermore, EBM optimizes the traditional boosting (greedy search) approach by leveraging *cyclic gradient boosting*, which learns a shallow tree for each feature in a round-robin fashion [250].

For simplicity and accuracy, `PrAM` leverages BO to automatically adjust model configurations, including the number of histogram bins, the number of considered interactions, learning rate, etc. It helps developers easily obtain the optimal model which is compact and accurate.Compared to the complex DL models, `PrAM` can not only provide interpretability, but also takes less training resources (without the need of GPUs) and training data samples, significantly reducing the training time and cost.

`PrDT`: Decision Tree based Method Our second interpretable model `PrDT` is constructed from Decision Trees (DTs). DTs are binary tree-structured models where each *branch node* tests a condition and each *leaf node* makes a prediction [309]. Because DTs are non-parametric and can be essentially expressed as an equivalent rule list, they are transparent and simple to interpret how a prediction is obtained. Besides, the decision-making processes of DTs can be visualized so developers can easily adjust the trees according to the system requirements. They present powerful prediction capability for both classification and regression tasks, even compared with complex black-box models.

In addition to the excellent interpretability and accuracy, DTs have extremely low computation overhead and inference latency. Consequently, they are applicable to many scenarios with strict latency and resource constraints [247, 310]. Besides, DTs

also exhibit other benefits, including robust performance under dynamic system environments, requiring less training data and no data preprocessing overhead during inference.

It is necessary to optimize the complexity of a DT to avoid the overfitting issue, which can affect the model generalization, accuracy and computation overhead. Instead of adding constraints (e.g., maximum depth, minimum number of samples for a leaf node) during DT training, `PrDT` trains a full decision tree without any limitation to capture more information from the training dataset. We adopt *minimal cost-complexity pruning* [248] to prune the full tree in the *post-processing* stage, we adopt Minimal Cost-Complexity Pruning (CCP) [248] to prune the full tree in the *post-processing* stage. This algorithm aims to minimize a cost-complexity metric $R_\alpha(T)$ which is defined as

$$R_\alpha(T) = R(T) + \alpha \cdot N(T) \tag{5.2}$$

where $R(T)$ and $N(T)$ denote the misclassification cost (error rate) and complexity penalty (the number of leaves) of the decision tree T respectively. The trade-off between accuracy and sparsity of the tree is controlled by the *complexity parameter* α: as α increases, more leaves are pruned and the total impurity increases.

Figure 5.4 presents the impact of the *complexity parameter* α on the model performance and complexity for the LinnOS system (§5.4). When α is too small (before the red dashed line), `PrDT` has poor performance because of the severe overfitting. With a bigger α, developers could trade-off the model accuracy and complexity based on their system requirements. The optimal pruning factor can differ by many orders of magnitude for different systems according to our experiments. It is hard for system developers to identify an appropriate value of α intuitively. To address this, Primo adopts bayes optimization to perform post pruning automatically (§5.3.3).

5.3.3 Model Training

Primo supports two training modes. (1) *Direct*: the developer can train an interpretable model from scratch. This applies for most **prediction-based** systems. (2) *Distill*: the developer can generate an interpretable model from the original black-box model through the *Distill Engine*. This is mainly for **exploration-based** systems. To obtain high-quality models, both modes support the integration of *Bayes Optimization* for efficient model structure and hyperparameter search.

Bayes Optimization There exists a trade-off between the model complexity and accuracy for both interpretable models. In order to find accurate and succinct models,

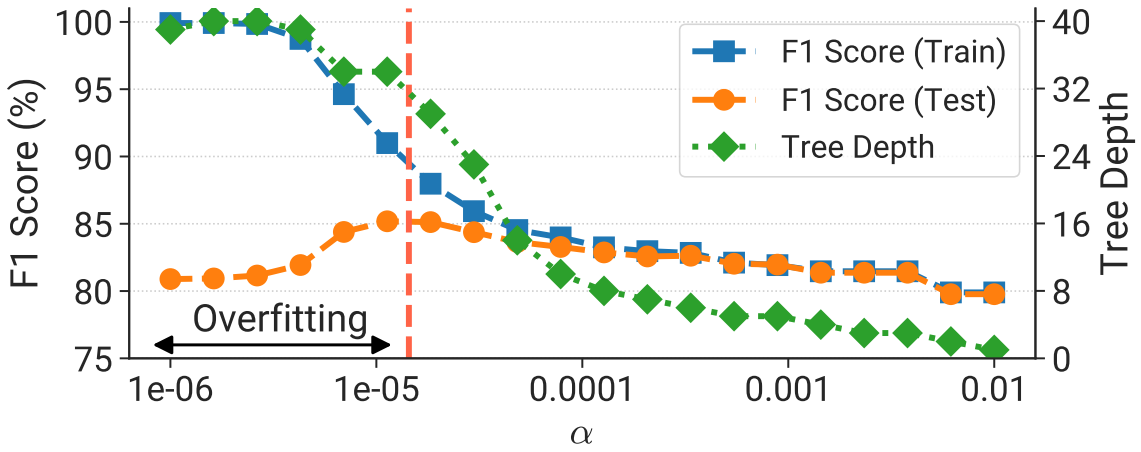


FIGURE 5.4: Performance (F1 Score) and complexity (Tree Depth) of the `PrDT` model under different α in LinnOS.

Primo leverages Bayes Optimization (BO) [137], an iterative algorithm to automatically search for the optimal model configurations.

Objective function. For both `PrAM` and `PrDT`, we build a universal model scoring function $S(\theta)$ to quantify the model performance and complexity as the search objective:

$$S(\theta) = P(\theta) + \lambda \cdot C(\theta)^{\gamma} \tag{5.3}$$

where $P(\theta)$ represents the model performance (e.g., classification accuracy) under hyperparameters θ during validation; λ is a knob that controls the model complexity according to users' preference; $C(\theta)$ is a metric for model complexity. For `PrDT`, $C(\theta^{\texttt{PrDT}}) = N_{leaves} \times N_{depth}$, where we consider both the number of tree leaves and tree depth since unbalanced-deeper trees typically cost longer condition inference time. For `PrAM`, $C(\theta^{\texttt{PrAM}}) = N_{interactions} \times N_{maxbins}$, where both the number of feature interaction terms and maximum number of bins in the feature histogram are included. Besides, the normalization factor γ regulates the effect of the model complexity.

Prior-based hyperparameter search. Specifically, Primo employs Gaussian Process (GP) as the probabilistic surrogate model of the objective function $S(\theta)$ in Equation 5.3. The prediction of GP follows a normal distribution: $p(S \mid \theta, \Theta) = \mathcal{N}(S \mid \hat{\mu}, \hat{\sigma}^2)$, where Θ indicates the hyperparameter search space. To determine which point should be evaluated next, Primo adopts expected improvement (EI) as the acquisition function to trade-off exploration and exploitation [133]. In each iteration, Primo generates a set of hyperparameters and evaluates them on the interpretable model to obtain new results which are used to update the surrogate model. Compared with Grid Search (GS) and Random Search (RS) [144], BO is more efficient since it fully utilizes the prior information to minimize the search space. For instance, as shown in Figure 5.4,

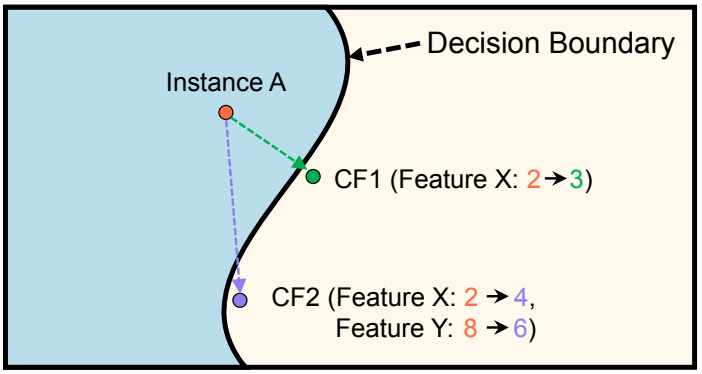


FIGURE 5.5: Illustration for the counterfactual explanation.

BO can rapidly reduce the search space to a smaller size (10^{-5}∼10^{-2}) for a better focus.

Distill Engine In some scenarios, the learning models require special optimization. For instance, LinnOS [247] leverages *biased training* to reduce the false submit rate while causing the higher false revoke rate. Primo introduces the *Distill Engine*, which can build an interpretable surrogate model to approximate the behavior of the original black-box learning model using knowledge distillation [311, 312].

Another application of the *Distill Engine* is RL policy extraction. Both `PrAM` and `PrDT` work well for **prediction-based** systems using supervised learning, but are less supportive for **exploration-based** systems due to their incompatibility with RL. A series of works [313–315] have demonstrated the feasibility of converting NN-based learning policies to an interpretable models. Primo adopts Viper [313] to perform RL policy extraction. Specifically, we collect the trajectories of $\{\boldsymbol{s}_i, a_i\}$ pairs (i.e., system states $\boldsymbol{s}_i$ and actions a_i of learned policy $\pi(\boldsymbol{s}_i, a_i)$) generated by the original RL model and perform supervised learning to build the interpretable models. To obtain a robust policy, we augment the poor-performing pairs and train the model iteratively until it is converged.

5.3.4 Post-Processing Optimization

After the interpretable model is built, developers can use their prior knowledge to further optimize the model and enhance the system performance. Primo designs two tools to assist developers in model post-processing. Note that these operations are *optional* since generally the trained interpretable models already achieve satisfactory performance.

Monotonic Constraint In many learning-augmented systems, the input features exhibit a monotonic relationship with the output values (e.g., higher video bitrate

TABLE 5.2: Summary of case studies for Primo evaluation.

	System Scenario	**ML Algorithm**	**Type**	**Primo**
LinnOS [247]	Flash Storage I/O	DNN	Online	PrDT (Direct)
Clara [266]	SmartNIC Offloading	Mixture (LSTM, GBDT, SVM)	Offline	PrAM (Direct)
Pensieve [292]	Video Streaming	RL	Online	PrDT (Distill)

selection with better bandwidth). But the corresponding model often presents a non-monotonic pattern due to the sub-optimal construction strategy or noisy training data (e.g., outlier data points, biased synthetic data). This can lead to unstable performance and intelligibility degradation in practice. To this end, Primo leverages a method from DP-EBM [250], which adds monotonic constraints to boosted trees via post-processing. Specifically, we model this task as an isotonic regression problem [316] with respect to a complete order. The objective is to minimize $\sum_i w_i \left(y_i - \hat{y}_i\right)^2$ subject to $\hat{y}_i \leq \hat{y}_j$ and weights w_i are strictly positive. We adopt the Pool Adjacent Violators (PAV) [252] algorithm to obtain an optimal solution maintaining monotonicity, and use it to replace the original shape function of `PrAM`. Our tool only needs developers to provide the feature name or index and the subsequent model adjustment process is transparent and automatic.

Counterfactual Explanation To make modifications to the models, developers need to answer some challenging questions, e.g., which feature related shape function should be adjusted? how to determine the modification degree? To help them make reasonable decisions, we design the Counterfactual Explanation tool in Primo to generate additional insights for model adjustment. As illustrated in Figure 5.5, this tool aims to find smaller change (green arrow) to the feature values that can alter the prediction to a predefined output within the dataset. It typically uses the k-nearest neighbors (kNN) algorithm to find k training instances with the minimum L_2 distances [317]. To address the inefficiency of the brute-force kNN approach, we propose to use Ball Tree [318] to partition data in a series of nesting hyper-spheres, thus the distance between a prediction point and the centroid is sufficient to determine a lower and upper bound on the distance to all points within the hyper-sphere node. This approach considerably reduces the query time when dealing with large-scale and high-dimensional datasets. And developers could perform guided model adjustment easily.

Primo Experiments. *In the following three sections, we will present three case studies to demonstrate how Primo can optimize state-of-the-art learning-augmented systems. Table 5.2 describes these three scenarios. To summarize, in addition to model transparency, Primo provides higher prediction accuracy, smaller inference overhead and better model generalization ability. These greatly facilitate model deployment in practice.*

5.4 Case Study 1: LinnOS

As the first case, we consider LinnOS [247], a learning-based operating system that accelerates storage applications. LinnOS adopts a 3-layer neural network (31-256-2, in total of 8706 parameters) for *each* SSD to precisely predict its performance. To achieve this, it collects the traces of real workloads running on the SSD and obtains fine-grained information (per I/O), including recent queue lengths and latency. Instead of predicting the concrete latency values, LinnOS simplifies it as a binary (fast / slow) classification task through setting an inflection point (IP). LinnOS infers the SSD speed using a lightweight neural network. The motivation behind LinnOS is that SSD read latency is unstable because some SSD internal operations (e.g., write-triggered garbage collection, buffer flushing) are contending with user read I/Os. In addition, there are the same replicas in other SSDs within the storage array (e.g., flash RAID) and we can utilize the built-in *failover* logic to circumvent slow read I/Os. The overhead of switching the *read* operation to a redundant SSD (namely *failover*) is ignorable compared with I/O pending time.

Implementation. We implemented `PrDT` models into the Linux kernel v5.4.8 (same version with LinnOS) within the *block* layer (written in C). We use the same SSD I/O traces in LinnOS, which were collected from Microsoft Bing Index servers and have been preprocessed by the authors.

Although the LinnOS workflow files are open-sourced on the Chameleon Cloud [319], the experiment results are unreliable due to the unstable and random SSD I/O accesses (also argued by the authors). Consequently, we shift our implementation and evaluation on a bare-metal server from CloudLab [320]. It contains four homogeneous enterprise-level 1.6TB SSDs. One of them serves as the system drive and the rest three SSDs are used for performance evaluation. Additionally, due to the rapid development of the SSD technology in recent years, the Microsoft traces collected in 2016 cannot give sufficient load pressure for evaluation. So we execute a constant writing task in the background for each SSD (LinnOS only records read I/Os). According to our numerous tests, the additional load will not cause fluctuations in the evaluation results.

For the `PrDT` interpretation results, Primo can provide a more precise visualized file of the `PrDT` which covers the number of samples at each flow and the Gini impurity value of each node rather than Figure 5.6.

Primo automatically selects the `PrDT` model for LinnOS, since it has comparable accuracy and lower inference latency than `PrAM`. For comprehensive evaluation, we consider two models with different optimization objectives: *efficiency*-oriented (Primo-E) and *performance*-oriented (Primo-P). We compare Primo with two baselines. (1) Base:

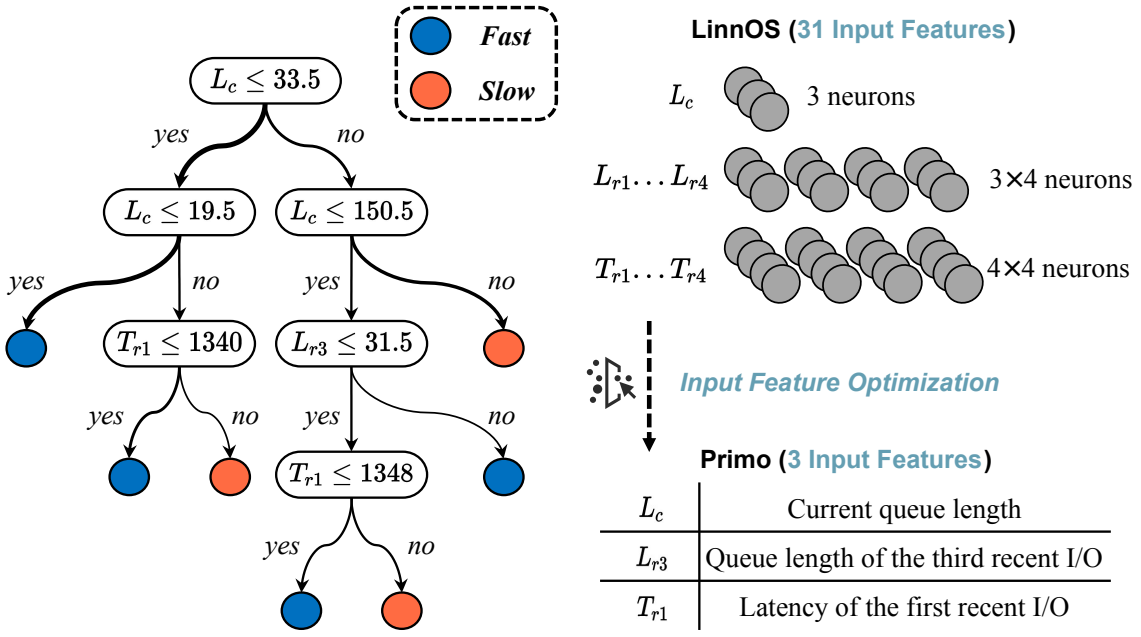


FIGURE 5.6: (**Left**) Learned `PrDT` model for an SSD. The thicker arrow line denotes the higher frequency. (**Right**) Primo optimizes the input features of LinnOS. Each feature represents a *digit* in LinnOS while a complete *number* in Primo.

the vanilla Linux I/O mechanism. (2) LinnOS: we set the inflection point of LinnOS as a constant percentile (at p85 latency) and apply the biased loss to the model training (all keep the same).

5.4.1 System Interpretation

The primary goal of Primo is to provide interpretation for the target system. Figure 5.6 (left) presents the learned decision tree (Primo-E) for one SSD. The explanation of each notation can be found on the right side. From this tree, we can clearly understand how Primo makes decisions for each prediction (*Local interpretation*). We can also obtain intuitive cognition of the overall model behavior (*Global interpretation*) through observing the thickness of each decision path (arrow lines).

Specifically, the top-2 layers of the DT show Primo first classifies I/O requests from the *current queue length* (L_c), indicating this feature can significantly affect the prediction results. Developers can perform adjustments to L_c thresholds to optimize system behavior. Because the 4-layer DT only contains 7 leaves (terminal nodes), each prediction needs to take at most 4 condition tests at the branch nodes and the majority of test instances only need to execute 2 condition tests. This inference overhead is much smaller than the original DNN model with 8706 parameters in LinnOS. Moreover, as shown in Figure 5.6 (right), Primo only takes 3 input features without any preprocessing, which further reduces the model complexity and deployment overhead. On the contrary, LinnOS needs to perform input data preprocessing for all 9

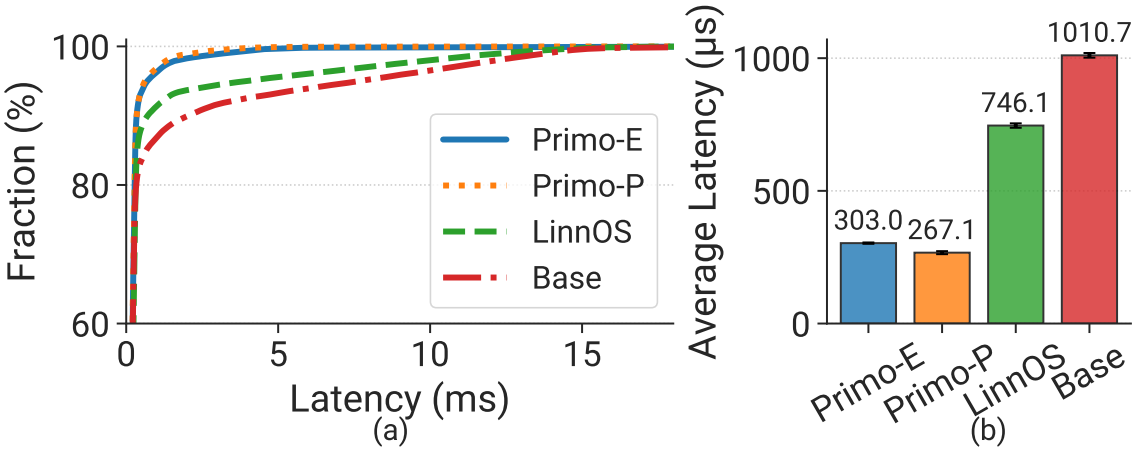


FIGURE 5.7: Overall performance comparisons. (a) CDF of I/O latency. (b) Average I/O latency.

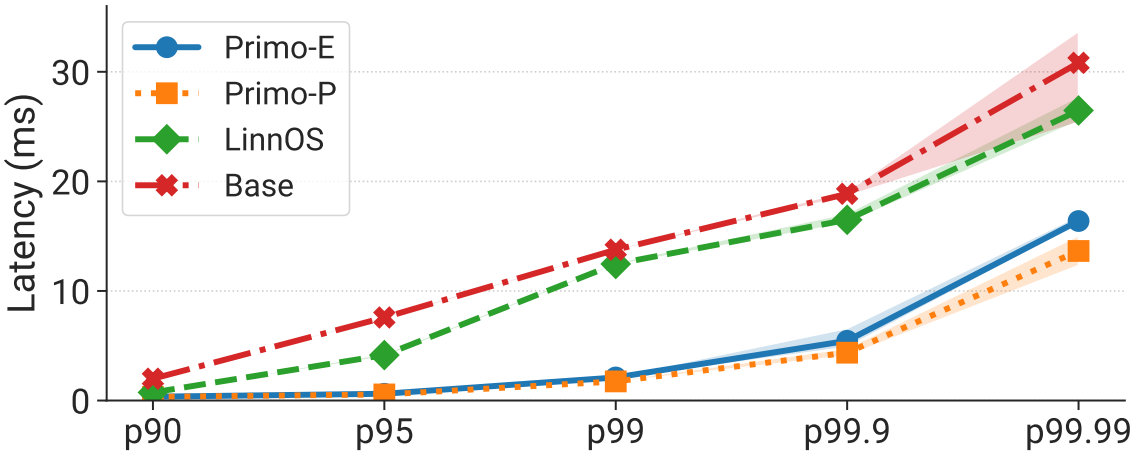


FIGURE 5.8: Tail percentiles of I/O latency.

metrics to form a 31-dimensional input feature (e.g., $L_c = 15$ needs to be converted into a {0, 1, 5} vector). This operation is necessary for *every* I/O read operation, remarkably exacerbating the inference overhead.

5.4.2 Performance Analysis

We evaluate the performance of Primo in the LinnOS flash storage I/O scenario from the following two perspectives:

Overall performance. Figure 5.7 shows the Cumulative Distribution Function (CDF) and average I/O latency (with the standard deviation) of each method over three independent experiments. It is obvious that both Primo-E and Primo-P significantly outperform LinnOS. Compared with the base I/O mechanism, LinnOS reduces 26.2% I/O latency on average, while Primo decreases the I/O latency by 70.0∼73.6%.

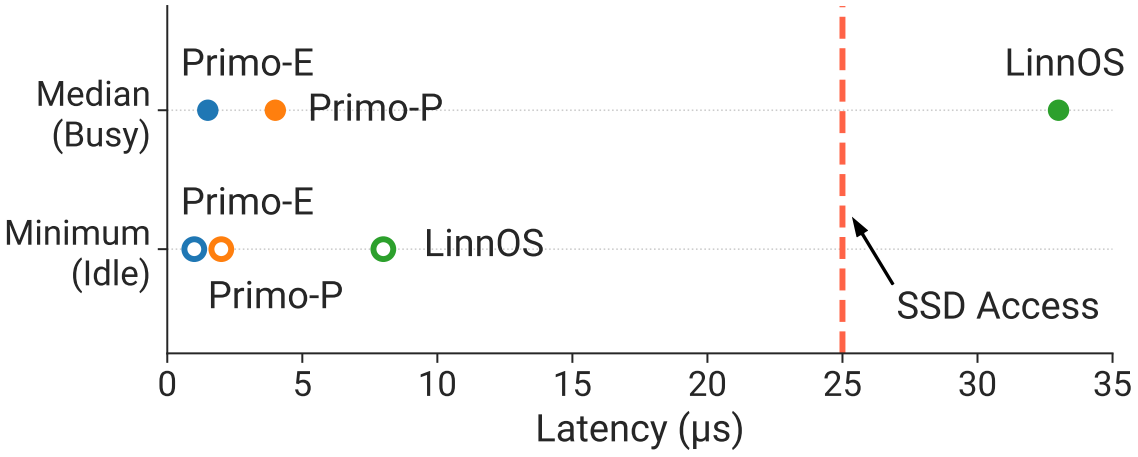


FIGURE 5.9: Model inference latency. Empty circles represent the minimum inference latency when the system is idle. Solid circles represent the inference latency of the median I/O operation when the system is busy. The vertical line indicates the basic SSD access latency (reading 4KB data in the idle state).

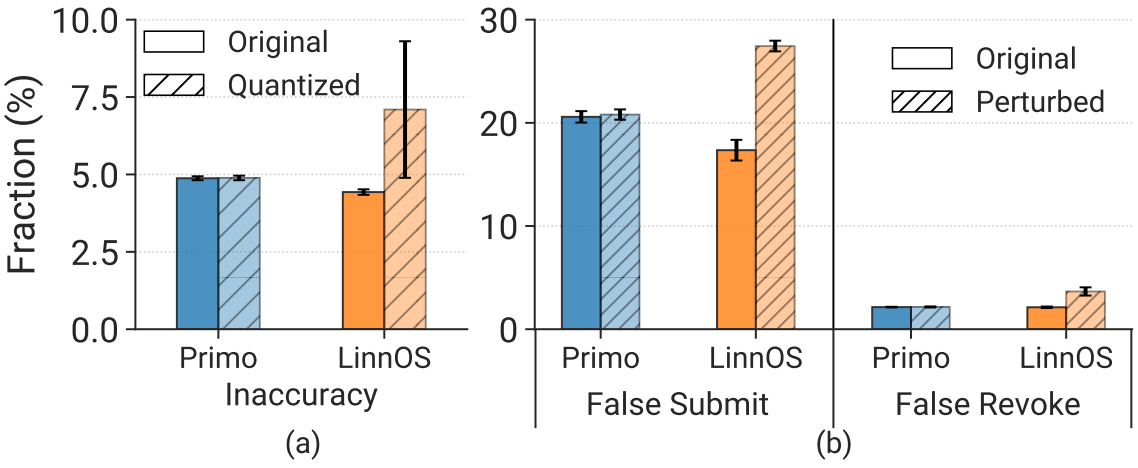


FIGURE 5.10: (a) Quantization impact. (b) Robustness test.

It indicates Primo can achieve an additional 2.5∼2.8× (Primo-E / Primo-P) improvement over LinnOS.

Tail performance. The tail behavior is critical to system performance. Figure 5.8 presents the average I/O latency and the range at tail percentiles (from p90 to p99.99). We find LinnOS fails to reduce tail latency on the tail, and the curve almost overlaps with the Base case. On the contrary, Primo-P achieves 7.9×, 4.3× and 2.3× performance improvement over the vanilla I/O mechanism at p99, p99.9 and p99.99 respectively. Additionally, Primo-P also performs much better at p90 (2.2×) and p95 (7.5×) compared to LinnOS.

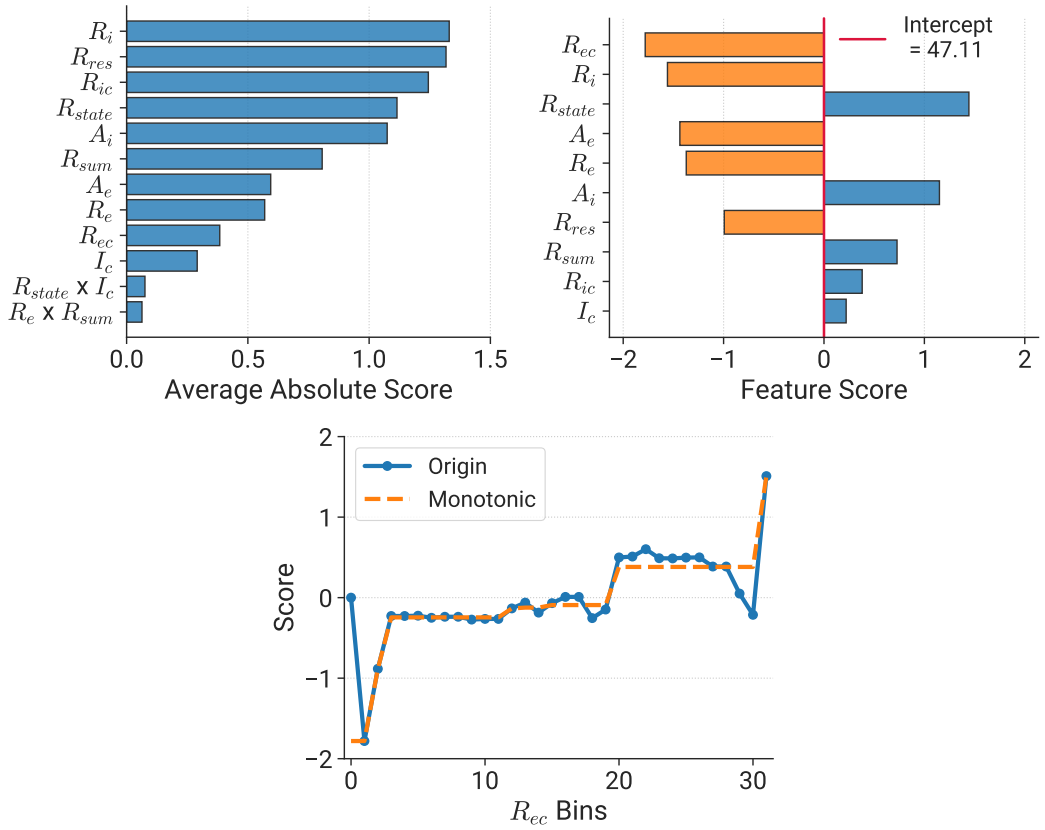


FIGURE 5.11: Interpretation and visualization of the `PrAM` model in Clara-MS. (**Left**): Global interpretation of overall feature importance. (**Middle**): Local interpretation of each feature's contribution to individual predictions. (**Right**): Visualization of the learned shape function of R_{ec} (blue line), and with the monotonic constraint post-processing optimization (orange line).

5.4.3 Effectiveness Analysis

We perform the effectiveness analysis from the following perspectives to investigate why Primo can outperform LinnOS.

Inference overhead. In Figure 5.9, we measure the extra inference latency of Primo and LinnOS. (1) When the system is idle, we measure the minimum inference latency. We observe that LinnOS takes $8\mu s$, while the overhead of Primo-E is almost negligible ($\leq 1\mu s$), making the deployment more lightweight. (2) When the system is busy with heavy I/O operations, LinnOS requires a median inference latency of $33\mu s$ due to the high frequency of preprocessing and inference. This is even higher than the basic SSD access latency ($25\mu s$). In contrast, Primo remains relatively lower inference latency with smaller overhead.

Quantization. Since floating points are not well supported in the Linux kernel, the model weights of LinnOS and the thresholds of Primo are converted to integers by

quantization. This can achieve smaller inference latency at the cost of accuracy degradation. Figure 5.10 (a) shows the quantization impact on the prediction accuracy. It is evident that the accuracy drop of LinnOS is over 2% and varies significantly among different SSD models. In comparison, Primo-E has negligible accuracy degradation, as the node threshold values are naturally integers or the decimal part is 0.5.

Robustness. A good model should exhibit high robustness against system state drifting. To measure the robustness of those methods, we synthesize some perturbed samples by adding Gaussian noise to the test dataset. The noise is added to all 4 recent I/O queue lengths ($\sigma = 5$) and I/O latency ($\sigma = 100$)[2]. Figure 5.10 (b) illustrates the false submit and false revoke rates of LinnOS and Primo-E under the original and perturbed test datasets. Reducing the false submit rate is far more important since the *failover* overhead of false revoke is negligible. It is obvious that Primo keeps stable accuracy under perturbed input while LinnOS presents severe performance degradation. The robustness of the interpretable model in Primo derives from fewer input features and the inherent stability of the tree structure compared with the DNN model.

Summary:

Key Observation from LinnOS

The high inference overhead of the DNN model can hinder its deployment in practice, meanwhile seriously damaging the effect of system performance improvement. Primo successfully overcomes this bottleneck.

5.5 Case Study 2: Clara

Clara [266] is an offline tool that generates offloading insights for network functions (NFs) on SmartNICs. Since the performance characteristics of offloaded programs are opaque prior to porting and offloading strategies are difficult to reason about, developers need to first cross-port NFs to the SmartNIC, perform workload-specific benchmarks, and then iteratively tune the ported programs to achieve higher performance. Clara can analyze a legacy NF in its unported form and suggest porting strategies for the given NF to achieve higher performance. The main challenge of adopting those ML techniques in Clara is that *insufficient* SmartNIC programs can be served to produce training data. Clara has to utilize YarpGen [278] to generate abundant synthesized programs. Clara contains several components for the generation

[2]Since the current queue length L_c is the most significant feature, we do not modify its value to avoid changing the real label.

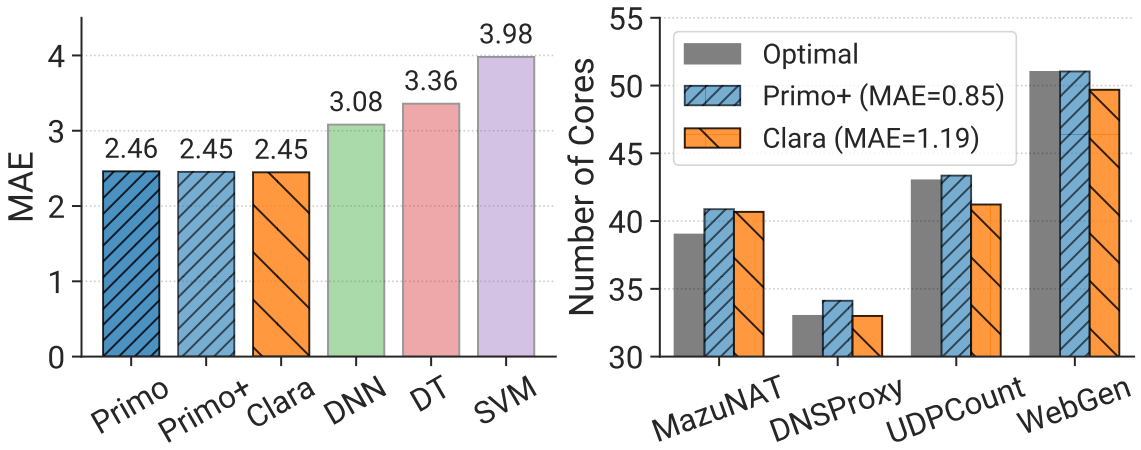


FIGURE 5.12: Evaluation on Clara-MS. (**Left**): Mean Absolute Error (MAE) of testset. (**Right**): Prediction of 4 real NFs.

of different offloading insights. Each component adopts a ML algorithm as described below:

- **Multicore Scale-out analysis (Clara-MS).** SmartNICs use multicore parallelism to improve packet processing performance. Clara adopts *GBDT* [256] to predict the optimal number of cores for each NF.
- **Algorithm Identification (Clara-AI).** Certain packet processing algorithms in the host NF can benefit from ASIC accelerators in the SmartNIC. Clara adopts *SVM* [321] to identify such code blocks.
- **Cross-platform Prediction (Clara-CP).** Clara trains an *LSTM* network [92] to predict the number of compute and memory instructions that a NF can be compiled to.

Implementation. We follow the author-provided data preprocessing pipeline to deal with synthesized and real traces for training and evaluation. We conduct Clara evaluation on a Ubuntu 20.04 server with one Intel Core i9-10900 CPU, 64 GB memory and an NVIDIA RTX 2080Ti GPU. Note that because it is an offline system and we focus on model accuracy, Netronome SmartNIC is not required in our experiment. We employ the `PrAM` model to replace all the three ML models in Clara, as it has better accuracy than `PrDT`. To analyze the effectiveness of transparent model adjustment in Primo, we also evaluate the model performance with post adjustment (denote as PRIMO+). We compare Primo with the original models in Clara (LSTM, GBDT and SVM), as well as some alternative baseline algorithms (CNN, DT, TPOT [322] (namely AutoML), K-Nearest Neighbor (kNN)).

TABLE 5.3: Notation descriptions of Clara-MS.

Notation	Description
A_i	IMEM Access
A_e	EMEM Access
I_c	Compute Intensity
R_i	IMEM / Overall Intensity Ratio
R_e	EMEM / Overall Intensity Ratio
R_{ic}	IMEM / Compute Ratio
R_{ec}	EMEM / Compute Ratio
R_{sum}	(IMEM + EMEM) / Compute Ratio
R_{res}	(IMEM − EMEM) / Compute Ratio
R_{state}	Non-Stateful / Stateful Compute Ratio

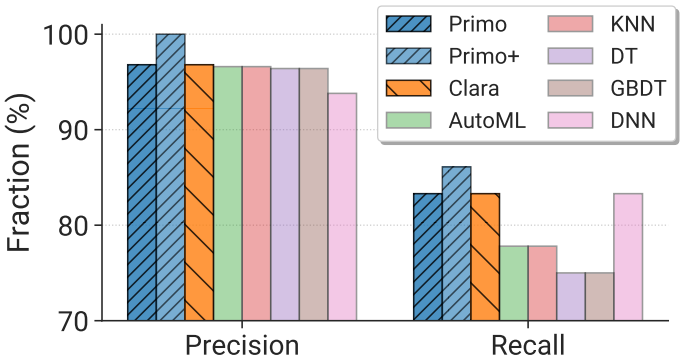


FIGURE 5.13: Model precision and recall rates in Clara-AI.

Notation. We clarify the notations used in system interpretations (Figure 5.11). Table 5.3 shows processed features used in Clara-MS, where IMEM indicates SRAM-based internal memory and EMEM indicates DRAM-based external memory on SmartNICs. Instructions can be classified into Stateful (e.g., loads and stores to global variables in memory) and Non-Stateful (e.g., compute instructions, or accesses to function-local variables). Moreover, Overall Intensity represents the sum of IMEM, EMEM and Compute Intensity.

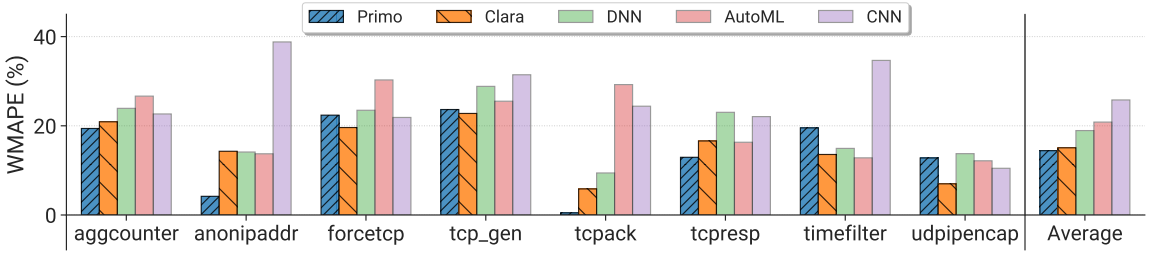


FIGURE 5.14: Weighted mean-absolute percentage error (WMAPE) over 8 types of NFs in Clara-CP.

5.5.1 System Interpretation

As shown in Figure 5.11, Primo provides comprehensive interpretation for the Clara-MS task, including global and local interpretation, as well as transparent shape functions. From the left figure, we find R_i, R_{res} and R_{ic} are the most important features that contribute most to model prediction. Developers should pay more attention to shape function optimizations for these features. We also notice the impact of feature interactions is relatively less important, indicating that we can reduce their priority in model optimization. The middle figure presents the interpretation of the individual prediction for *UDPCount* NF. The final prediction equals the sum of every feature score and the intercept constant (Equation 5.1). Through the local interpretation, developers can clearly check the model behavior for each prediction to make the corresponding adjustment. Moreover, the right figure (blue line) illustrates the learned shape function for R_{ec}, which allows developers to dive deeper into fine-grained model adjustment (such as the orange line).

5.5.2 Performance Analysis

Since Clara is an offline system, for each task, we mainly evaluate the model accuracy rather than the inference cost.

Clara-MS. As shown in Figure 5.12 (left), our interpretable model in Primo achieves similar accuracy as the GBDT (XGBoost [256]) model in Clara, and outperforms other ML models over the synthesized test dataset. Figure 5.12 (right) further presents the accuracy of Primo for 4 real NFs. Compared to Clara, Primo achieves 1.4× less prediction errors and at most 5% error to the optimal configurations.

Clara-AI. In Figure 5.13, Primo achieves the equivalent precision and recall rates as the SVM model in Clara, and beats other ML algorithms. Through successfully identifying CRC-based NFs, Primo could improve peak throughput by 1.6× and decrease latency by 25% [266].

Clara-CP. Clara uses the LSTM model to predict the number of instructions for unported codes. Figure 5.14 shows the accuracy of the Clara-CP task over 8 representative real NFs and the `Average` column represents the WMAPE results across all the NFs. We observe Primo (14.4%) delivers better performance than Clara (15.1%). This demonstrates the capability of Primo to cope with complex program embeddings.

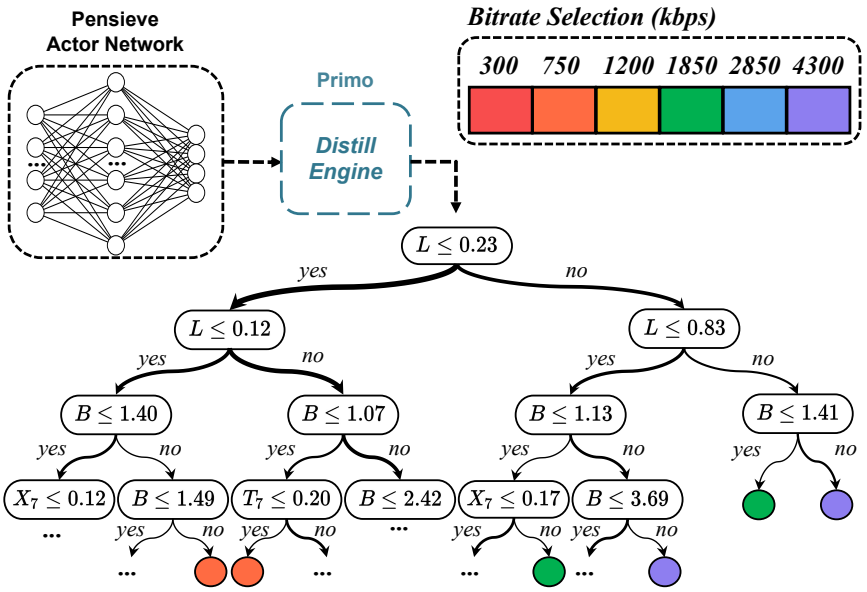
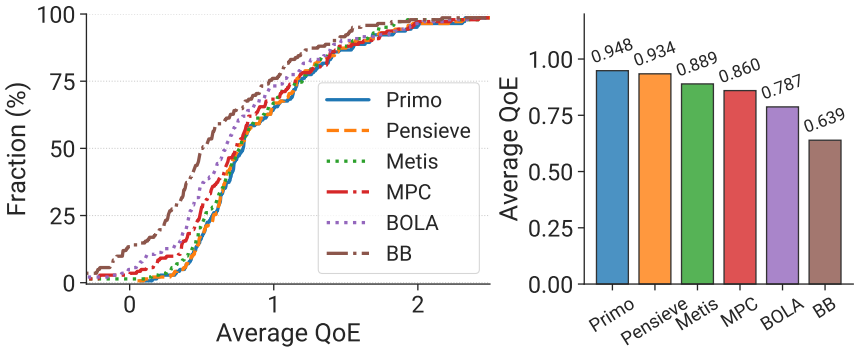


FIGURE 5.15: Visualization of the interpretable model distilled from the Pensieve policy. For simplicity, we only present the top 5 layers, and the ellipsis indicates subsequent nodes.

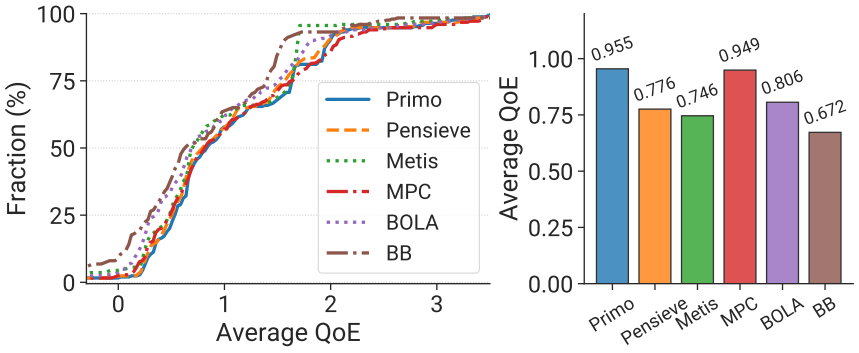
5.5.3 Model Adjustment

To overcome the training data insufficiency issue, Clara uses YarpGen [278] to generate synthesized programs. This inevitably introduces certain data distribution drifts from the actual scenario. Specifically, there exist instruction distribution differences (0.0303 of Jensen-Shannon divergence and 0.0354 of Bhattacharyya distance) between real-world and synthesized click programs [266]. Such drifts could compromise the model performance. The transparency of the Primo model allows developers to discover and fix undesirable behaviors caused by the synthesized data. In addition, Primo designs two post-processing tools to help developers adjust the models based on their domain knowledge:

Monotonic Constraint. As introduced in §5.3.4, developers can leverage Primo to generate a new shape function with monotonic constraint and rectify the incorrect behaviors of the models automatically. For instance, in Figure 5.11 (right), the developers know the desired number of cores should be proportional to the memory/compute intensity, i.e., R_{ec} (EMEM/Compute Ratio). Then they can replace the original shape function (blue line) with the monotonic shape function (orange line). They can check each shape function and decide whether it is necessary to apply such adjustment based on their prior knowledge. To evaluate the effectiveness of this strategy, we apply the *Monotonic Constraint* tool to two shape functions (A_i & R_{ec}) and yield the adjusted model PRIMO+. As shown in Figures 5.12, PRIMO+ achieves better



(A) Norway HSDPA



(B) FCC Broadband

FIGURE 5.16: Overall performance of Primo compared with other methods on the Norway HSDPA and FCC Broadband traces.

prediction accuracy. This shows the monotonicity of the Primo model can be achieved via simple post-processing and appropriate adjustment can bring better performance.

Counterfactual Explanation. This tool aims to provide simple and intuitive explanations for model troubleshooting. More concretely, it helps developers to understand why this prediction is wrong and how to adjust the model to fix it. We use Clara-AI as an example to describe its usage and evaluation. Clara employs Sequential Pattern Extraction (SPE) [323] to extract code features as *boolean* sequences (each containing 102 features) to indicate whether the NF program contains code blocks for acceleration. Our Primo model allocates a contribution score for each feature. To fix False Negative (FN) predictions, we utilize this tool to find the closest k instances from the data set with the opposite label. Through the comparison of these instances, we can easily discover the feature with inadvisable learned scores and adjust the score. In this case, we increase the contribution weight of the $84th$ feature appropriately. We can also perform transparent debugging for False Positive (FP) predictions similarly. In Figure 5.13, PRIMO+ further enhances the F1 score from 89.6% to 92.5%.

Summary:

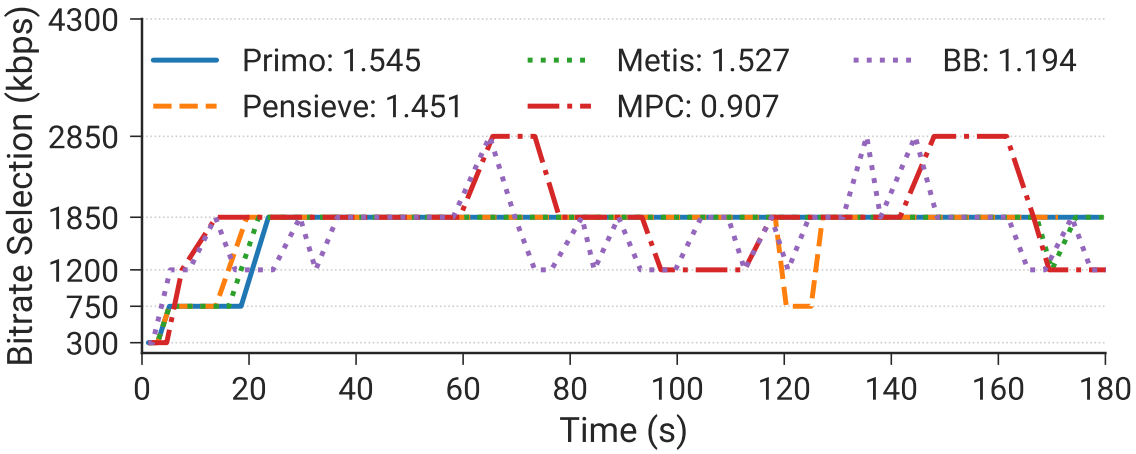


FIGURE 5.17: Profiling the bitrate selections of ABR algorithms over one typical Norway HSDPA trace. Legend presents the average QoE of each algorithm.

Key Observation from Clara

Instead of using various black-box models for different tasks, Primo uses a unified interpretable model achieving equivalent even better accuracy and endows capability of model adjustment to remedy the problem caused by drifted synthesis data.

5.6 Case Study 3: Pensieve

Our third case study is Pensieve [292], Pensieve [292] is a system that learns adaptive bitrate (ABR) algorithms automatically using RL technique. The online videos are stored on servers as multiple chunks (a few seconds of the video) and each chunk is encoded at several discrete bitrates (resolutions). Specifically, Pensieve adopts A3C [324] to perform RL training. Both the actor and critic networks use the same NN structure which contains a 1D-CNN layer and a fully connected layer. The actor takes recent network observations as input and suggests the bitrate for the next video chunk as the output. Content providers employ ABR algorithms to optimize user quality of experiment (QoE) which is defined as:

$$QoE = \sum_{n=1}^{N} q\left(R_n\right) - \mu \sum_{n=1}^{N} T_n - \sum_{n=1}^{N-1} \left|q\left(R_{n+1}\right) - q\left(R_n\right)\right| \tag{5.4}$$

where R_n represents the bitrate of the n^{th} chunk and $q(R_n)$ maps that bitrate to the quality perceived by a user. T_n represents the rebuffering time and the last term penalizes changes in video quality to favor smoothness.

TABLE 5.4: Notation descriptions of Pensieve.

Notation	Description
X_t	Past chunk throughput
T_t	Past chunk download time
N_k	Next chunk sizes
B	Current buffer size
C	Number of chunks left
L	Last chunk bitrate

Implementation. We employ the same server used in Clara for the Pensieve experiment. The video server is based on Apache httpd (Version 2.4.41) and uses Google Chrome (Version 96) as the client video player. We use 142 Norway HSDPA [325] network traces (provided by the authors) for evaluation, in addition, we process another 249 FCC [326, 327] traces (2018 version, follow the same preprocessing pipeline) for model generalization evaluation. Because the original FCC-18 network speed is much faster than HSDPA and we cannot distinguish the performance difference among algorithms, we scale down the network speed by 4 times to keep consistent with FCC-16 with regards to the median values.

We obtain `PrDT` model through distilling from the trained RL actor model (pre-trained model that Pensieve authors provided). After that, we implement `PrDT` (written in JavaScript) into *ABRController* of dash.js [328] (Version 2.4). For the test videos, we use the same video in Pensieve at bitrates in {300, 750, 1200, 1850, 2850, 4300} kbps (which pertain to video modes in {240, 360, 480, 720, 1080, 1440}p). This video is divided into 48 chunks and each chunk represented approximately 4 seconds. Furthermore, we adopt $q(R_n) = R_n$ in Equation 5.4 to set *linear QoE* as the our evaluation metric. For baselines, we compare Primo with the following algorithms: (1) The RL model in Pensieve. (2) Buffer-Based(BB) [329]: selecting bitrates with the goal of keeping the buffer occupancy above 5 seconds. (3) BOLA [330]: selecting bitrates with Lyapunov optimization on buffer occupancy observations. (4) MPC [331]: selecting bitrates with a control-theoretic model. We evaluate robustMPC variant which can better handle errors in throughput prediction. (5) Metis [34]: using a decision tree to explain the Pensieve RL model, which represents the handcrafted DT approach. Evaluations are performed on the simulator provided by Pensieve, except the deployment experiment (latency).

Notations. We clarify notations used in system interpretations (Figure 5.15). Table 5.4 shows environment state variables used in Pensieve to generate bitrate decision, where X_t and T_t denotes past sequences of throughput and download time respectively ($t = 1, \cdots, 8$). Moreover, N_k represents sequence of next chunk sizes ($k = 1, \cdots, 6$).

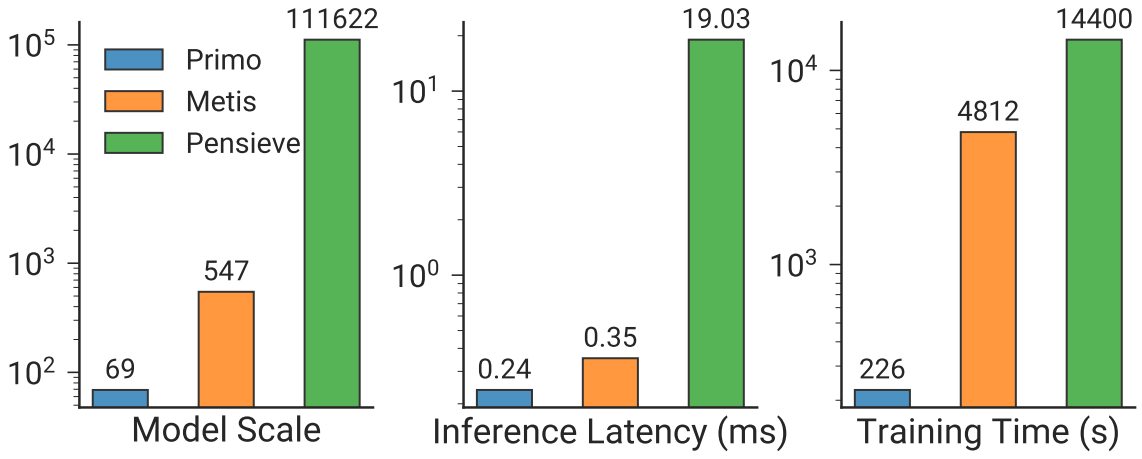


FIGURE 5.18: Comparing three learning-based ABR methods.

5.6.1 System Interpretation

Figure 5.15 illustrates the learning process with the Distill Engine, as well as the decision making process of the interpretable policy. This DT contains 8 layers and 35 leaves in total, which is compact and simple enough for developers to understand its complete operation logic.

Similar to the Primo model in LinnOS (Figure 5.6), the first 2 layers divide decision flows based on the feature L (Last chunk bitrate) which is in line with our perception. In the third layer, Primo proceeds to classify environment states (inputs) according to the feature B (Current buffer size). These observations indicate both L and B are the key features that affect the final bitrate decision, inspiring developers to pay more attention to them when designing ABR algorithms.

5.6.2 Performance Analysis

Overall performance. Since the ABR algorithm could encounter unprecedented network conditions by different clients, it is important to evaluate its generalization ability. So in addition to the *Norway HSDPA* trace used for model training, we also evaluate another trace *FCC Broadband* that is never applied for training. Figure 5.16 presents the QoE distribution and average QoE of each method on the two traces. For Norway HSDPA, the CDF curve of Primo almost overlaps with Pensieve's curve, and the average QoE is even 1.5% higher than Pensieve. This demonstrates Primo has successfully learned the Pensieve policy with a simple decision tree and outperforms other ABR algorithms. Furthermore, for FCC Broadband, Primo presents better generalization than other two learning-based algorithms. Such advantages are attributed to the adaptive pruning strategy in Bayes Optimization, and the imitation process

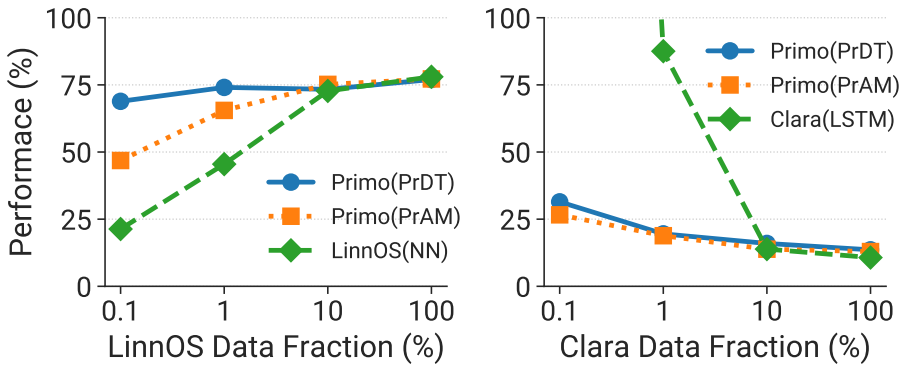


FIGURE 5.19: Model performance with less training data. (**Left**): Recall rate in LinnOS (the higher the better). (**Right**): WMAPE in Clara-CP (the lower the better).

in the Distill Engine. In contrast, although Metis also uses a decision tree to get a surrogate model from Pensieve, it has some performance degradation as its inflexible pruning strategy.

Example analysis. Figure 5.17 profiles the bitrate selection actions of different ABR algorithms over a single network trace. We find two heuristic algorithms (BB and MPC) keep fluctuating during the video streaming, which could cause a terrible user experience. The other three learning-based algorithms have more stable decisions. Pensieve decides to decrease the bitrate at 120s and Metis chooses to reduce the video resolution at 170s. This can cause an unsmooth experience (as the penalty term in Equation 5.4) even though they adjust back the bitrate quickly. In contrast, Primo gives a much more smooth experience.

Training and inference overhead. The Primo model is more compact and simpler. Figure 5.18 (left) compares the model complexity[3] of different methods. We observe that Primo reduces the model scale by 1617× compared to the original Pensieve actor model. Even for Metis which also uses a decision tree, Primo can reduce the tree complexity by 7.9×. For inference, Primo only needs to perform 3∼7 condition tests to make a bitrate decision. It can reduce 70× and 1.5× inference latency compared with Pensieve and Metis respectively, as shown in Figure 5.18 (middle). To generate a model, in Figure 5.18 (right), Primo only needs less than 4 minutes for model distillation, which is 21.3× faster than Metis (under the same setting). Compare with Pensieve 4 hours training time, less than 4 minutes distill time is ignorable. In summary, Primo can greatly reduce overall operating costs in the video streaming scenario.

Summary:

[3]Model complexity refers to the number of parameters for Pensieve model, or number of nodes for Primo and Metis.

TABLE 5.5: Training time comparison with original DL models.

Task	DL Model	Origin	Primo	Improvement
LinnOS	3 × DNN (50 epoch)	564s	5s	112.8×
Clara-CP	LSTM (30 epoch)	1,081s	79s	13.7×

Key Observation from Pensive

With Primo, we obtain an interpretable model that has better performance and generalization ability than the RL policy. Moreover, it achieves much less inference overhead and low distill cost for practical deployment.

5.7 More Evaluation

We run some experiments to further evaluate the benefits of Primo more comprehensively.

Requiring less training data. Due to the simpler model structure and fewer parameters, Primo can have better performance in some scenarios without enough training data like Clara. In Figure 5.19, we compare the performance of two Primo models with the original DL models in LinnOS and Clara-CP using less training data. We use a smaller dataset (10%) in LinnOS as the baseline. Because LinnOS provides abundant data for DNN model training, 10% of original data can provide equivalent performance. It is clear that Primo models maintain better performance with limited training data, especially for the `PrDT` model. Conversely, the original DL models only work with abundant data. This shows Primo has broader applicability for various scenarios.

Short training time. Table 5.5 presents the training time of Primo and original DL models in LinnOS and Clara-CP, which adopt the default numbers of training epochs in their papers. Primo is able to reduce 2-3 orders of magnitude of training time. Even considering the hyperparameter search process, the significant time conservation could maintain since multiple trails can be executed concurrently. Note that GPUs can only provide very limited acceleration (<1.2×) for these two DL models. Additionally, LinnOS requires training a DNN model for *each* SSD and the prototype only considers three SSDs. In a production-level distributed storage system with thousands of SSDs, LinnOS could have a severe scalability issue. In contrast, Primo remarkably saves the training cost, making the deployment more feasible in practice.

Impact of BO. We further perform an ablation study on Bayes Optimization in Primo. Table 5.6 summaries the performance of the Primo model with and without

TABLE 5.6: Ablation study for Bayes Optimization.

Task	Metric	Primo w/o BO	Primo w/ BO	Improvement
LinnOS	F1 Score	0.8089	0.8518	5.3%
Clara-CP	WMAPE	0.1728	0.1442	16.6%
Clara-MS	MAE	1.0155	0.8660	14.7%

BO in LinnOS and Clara. We observe 5.3%∼16.6% accuracy improvement brought by BO. Besides, BO typically simplifies the model scale while obtaining better performance. For LinnOS, BO reduces over 15× tree nodes compared with Primo without BO. This verifies the importance of the BO component in making interpretable models more practical. For search time aspect, BO obtains over 1.2× acceleration compared with naive random search algorithm in Clara-MS task by reducing search trails to reach equivalent performance.

5.8 Discussion

Is the interpretation always correct? Yes. Primo provides the interpretation correctness guarantee for each generated model and each prediction. Existing interpretation tools [262, 299, 300] aim to offer explanations for understanding black-box models, whereas the generated interpretations are sometimes contradictory or even mislead users. In contrast, Primo models are inherently interpretable and developers can totally trust the interpretation.

Can Primo be applied to all systems? Primo has its limitations in some system scenarios. For instance, it does not yet support unsupervised learning scenarios (e.g., anomaly detection in security applications [332]). It cannot outperform black-box models in systems with extremely complex features, e.g., images, speeches. These will be our future work.

Is the post-processing step necessary? These operations are optional because the trained models without post-processing usually have excellent performance. In order to take full advantage of the interpretable models, the post-processing tools help developers leverage their expertise and domain knowledge to further optimize system performance. In a black-box model, it is hard to perform such optimization.

Can Primo work on a larger model? Yes. We have demonstrated Primo can outperform DNN models in various scenarios, including LinnOS (MLP with 8×10^3 parameters), Clara-CP (LSTM+FC with 4×10^4 parameters) and Pensieve (CNN+MLP with 1×10^5 parameters). They represent most model scale of learning augmented systems listed in [305]. For larger models, we evaluate Habitat [298] as an example. It leverages 8-layer MLP models, containing over 8×10^6 parameters, to prediction

DL operation execution time on heterogeneous GPUs. Primo can provide comparable prediction accuracy as Habitat across `conv2d`, `linear`, `lstm` and `bmm` operations.

How to interpret high-dimensional data? Admittedly, when handling high-dimensional datasets, Primo models may become more complicated for users to understand the whole model. However, Primo provides ordered feature importance for interpretation. Generally, users can focus on the top several tree layers of `PrDT` or several significant shape functions according to the global interpretation of `PrAM`.

5.9 Related Work

Interpretability of learning-augmented systems. To our best knowledge, there is no prior work that develops a unified framework for providing inherent interpretability for systems like Primo. Besides, interpretability is often overlooked during the development of learning-augmented systems, and only a few works consider it. Bao [283] is a learning-based system that adopts TreeCNN [333] for query optimization and the decision process can be inspected by developers. Tang et al. [334] proposed an interpretable method that extracts a Finite State Machine from a RL policy for storage resource allocation in Huawei. Grüner et al. [335] generated concise and interpretable rule-sets for unknown proprietary streaming algorithms (e.g. ABR approaches in Youtube, Twitch), which is similar to the Pensieve case study with Primo (§5.6).

Some efforts were also made on building tools for interpreting black-box models [34, 232, 259], as discussed in §5.2.3. Different from these methods, Primo does not seek for interpreting black-box models but directly building transparent models, with higher fidelity and efficiency.

Machine learning and system co-design. It is non-trivial to apply ML techniques for system design and deployment in practice. Autosys [33] introduces a framework to address common design considerations (e.g., learning-induced system failures, extensibility), and reported years of experiences in designing and operating learning-augmented systems at Microsoft. WhiRL [284] facilitates the safe deployment of RL-based systems through verifying whether the learned policy meets the designer's requirements. Some components in Primo are inspired from these works.

5.10 Summary

This work introduces Primo, a unified framework that assists developers to design practical learning-augmented systems with interpretable models. For different scenarios, Primo provides respective models and optimization solutions to meet the system requirements. Based on our case studies, we demonstrate that Primo can achieve *transparent, accurate* and *lightweight* system deployment in practice.

Chapter 6

Conclusions

6.1 Summary

My research focuses on the efficiency and practicality of machine learning systems. Specifically, my research expands along two lines:

(1) Efficiency. I mainly focus on ML workload scheduling system research in both cluster-level [23] and job-level [35]. In the cluster scheduling scenario, I find the unique nature of ML workloads via characterization and design ML models accordingly to deliver better scheduling efficiency (Chapter 2, [23]). Besides, we find existing systems fail to handle large models with billions of parameters and efficiently leverage cluster resources. To this end, we build a holistic system (Chapter 3, [35]) that automatically applies the novel hyperparameter transfer theory together with multiple system techniques to jointly improve the tuning efficiency. It is the first system that supports tuning of large models (e.g. GPT-3), providing nearly two orders of magnitudes improvement.

(2) Practicality. Beyond efficiency, easy to deploy systems into practice is significant. However, most existing work targets excellent system performance while ignoring its complexity and usability. As a result, it is typically nontrivial to shift a research prototype into a production-level system in practice. My work first attains excellent performance under the non-intrusive design principle (Chapter 4, [36]), which successfully copes with deployment problems of previous scheduler work and obtains over 2× improvement compared with SOTA. Besides, I also identify the gap between the research design and industrial system deployment. I develop a unified framework (Chapter 5, [37]) to achieve transparent, performant and lightweight systems, which can accelerate systems by over two order of magnitudes.

6.2 Limitations and Improvement Directions

Helios [23]: (1) We aim to design and integrate more services into our framework to make it more comprehensive. (2) Some attributes in our services may not be available in other clusters. We aim to design new qualified models with limited job information for our services. (3) Our services mainly rely on historical job data, and coarse-grained cluster information. We aim to collect and leverage more fine-grained resource information (e.g., GPU memory usage and computation unit utilization, CPU utilization) as features to build more accurate models for better cluster management performance. (4) We are planning to implement our prediction framework in our production clusters, and evaluate its effectiveness at scale.

Hydro [35]: (1) We can support more DL frameworks like TensorFlow [14] and JAX [160]. (2) We aim to consider more resource dimensions like CPU and network bandwidth besides GPU [212, 227], such as implementing the dataloader fusion of trials to further alleviate I/O contention. (3) We can expand the application scenarios such as cloud environments. It presents an opportunity for dynamic selection of heterogeneous spot instances, which can yield substantial cost savings [118, 126]. (4) We can enable partial model fusion across trails with minor architectural differences (e.g., add/remove/modify a few layers/blocks). Furthermore, Hydro can integrate model matching technique from ModelKeeper [141] to identify the models with similar architectures across jobs from different users and achieve cross-job level fusion, which can significantly improve cluster efficiency.

Lucid [36]: (1) We can support more scheduling objectives like fairness [42, 128, 220] and SLO-guarantee [226] to further improve user experience. (2) We can add heterogeneous GPU selection optimization by more fine-grained profiling for clusters with various GPU generations. Besides, we plan to fully exploit affiliated resources (e.g., CPU).

Primo [37]: (1) We can extend Primo to support learning-augmented systems with unsupervised learning. (2) To obtain a more accurate interpretable model, we can optimize the model training algorithm. Currently, we use CART [248], the most popular and widely applied approach, for decision tree learning in `PrDT`. This is based on the heuristic greedy algorithm where locally optimal decisions are made in each node. In the future, we plan to employ novel decision tree training algorithms (e.g., GOSDT [336], based on dynamic programming method) to solve the sub-optimal problem. (3) For practical deployment, comprehensive programming language support is needed because different systems have their own coding requirements. `PrDT` has a tool for converting Python-based models to other formats. But `PrAM` only supports the conversion to the ONNX [254] format currently. We aim to provide more model format

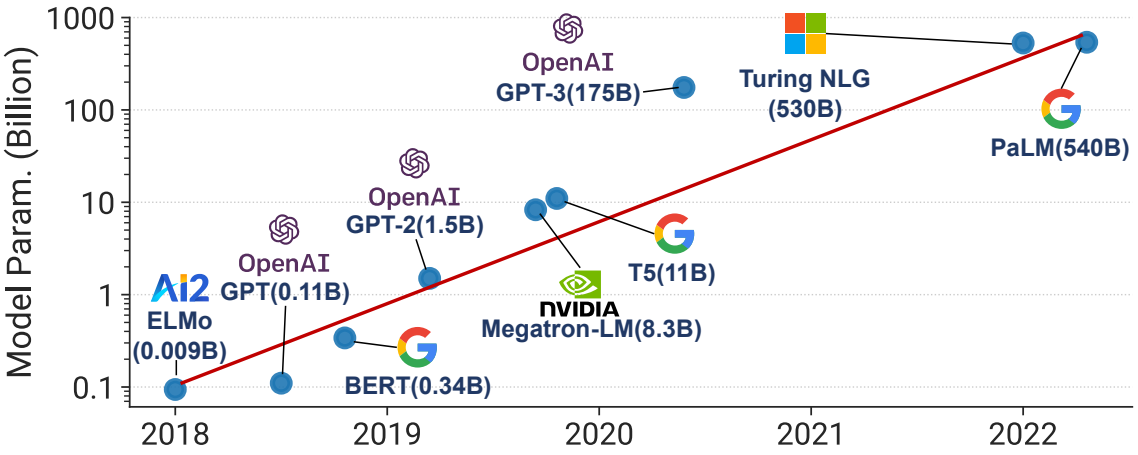


FIGURE 6.1: Trend of sizes of large language models (LLMs) over time.

conversion in the future. (4) Integration with existing RL-based system development frameworks (e.g., Park [337]) to facilitate more practical system deployment.

6.3 Future Directions

My work so far is a first step toward building *practical* and *efficient* machine learning systems. As for my future works, I plan to continue my research along two major directions: (1) **Systems for LLMs** and (2) **Explore More Scenarios**.

With the rapid advancement of large language models (LLMs) as shown in Figure 6.1, I plan to optimize and design systems LLMs.

- **LLM Model Serving.** All the works in this thesis focus on the training of DL models. However, the serving of models is also a challenging problem, especially for LLMs or generative AI models. Achieving low-latency, high-throughput inference and deploying these models across distributed systems pose significant engineering and operational challenges. Inference, unlike training, involves online workloads, making the parallelization and resource allocation for inference tasks a challenging endeavor. Additionally, the reduction of GPU memory footprint for efficient utilization poses another significant challenge. [338, 339] have offered preliminary approaches to tackle these challenges. I intend to further investigate this line.

- **Robust LLM Pretraining System.** LLMs are computationally expensive and require large amounts of data for training. Existing systems cannot handle training failures efficiently. According to the reports from existing companies, huge compute resource were wasted due to various training failures. As the

failure frequency increases with the scale of the training cluster, failures can dramatically slow down the training progress. Therefore, it is essential to design a robust LLM pretraining system to reduce failure frequency and handle failure recovery efficiently. There are some extraordinary works [126, 340, 341] propose several advanced techniques to improve the fault tolerance. I plan to explore more in this direction.

Besides, I aim to explore deeper in more efficient and practical systems for other scenarios.

- **Macro-scale Scheduling for AutoML Workload.** Existing cluster schedulers aim to regulate general DL jobs efficiently. However, such a workload type-agnostic manner can miss many system performance optimization opportunities, especially for AutoML jobs. AutoML automates the end-to-end process of real-world machine learning problems through Hyper-Parameter Optimization (HPO) or Neural Architecture Search (NAS). The search space is often very large and arbitrarily designated by the user. A better scheduling policy can efficiently shorten search time usage and reduce resource consumption. Future research should investigate the methodologies of determining: (1) *how many resources should be allocated to each trial* and (2) *which trial should be early terminated*. Moreover, when multiple AutoML jobs are concurrently executed in a cluster, the dynamic resource allotment to each independent job to provide fairness and efficiency guarantees is another challenging problem.

- **Joint Optimization with DL Compiler Techniques**. My research so far focuses on the strategy-level decisions for DL job scheduling. Combining DL model operator-level optimization is a promising direction for both the training and inference workloads. It can significantly accelerate job throughout and improve resource utilization. Indeed, the two problems are highly correlated: scheduling determines the workload resource scale and an efficient DL code generator aims to find the optimal job parallel scheme as well as intra-model fusion opportunities. The collaboration could benefit both sides and improve workload scalability within the cluster. Studying them in a joint context is promising and I believe I can help organically bridge the two fields.

- **General and Practical System Deployment.** While learning-augmented systems have demonstrated their remarkable performance across many domains, their deployment in practice is a non-trivial problem. My research so far considers the interpretability, training and inference overhead issues, however, more extensive work is required for practical deployment. For instance, (1) supporting learning-augmented systems with unsupervised learning is needed; (2) comprehensive programming language support is needed because different systems

have their own coding requirements. A tool for converting Python-based models to other formats is necessary; (3) integrating with existing RL-based system development framework to facilitate more practical exploration-based system deployment. These pave the way for research work into reality.

- **Interpretable System Performance Tuning.** With workloads becoming more computationally expensive and data amounts becoming larger, modern computer systems become more complex and are typically composed of multiple components, where each component has a plethora of configuration options that can be deployed individually or in conjunction with other components on different hardware platforms. Hence, understanding and configuring the systems is challenging and error-prone. Developers and end-users are often overwhelmed with the complexity and hard to reach desired performance goals. My research aims to enable interpreting and reasoning about configurable system performance with causal inference and counterfactual reasoning. It can assist developers to identify and fix performance faults, as well as provide tuning guidance.

Bibliography

[1] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. Antman: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 533–548. USENIX Association, 2020. xvi, xvii, xviii, 4, 11, 12, 16, 22, 42, 47, 48, 74, 78, 80, 84, 85, 93, 99, 103

[2] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '22, pages 945–960. USENIX Association, 2022. xviii, 2, 4, 5, 59, 76, 80, 84, 93, 99, 103

[3] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D. Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, page 1487–1495. Association for Computing Machinery, 2017. xxi, 41, 44

[4] Xingyou Song, Sagi Perel, Chansoo Lee, Greg Kochanski, and Daniel Golovin. Open source vizier: Distributed infrastructure and api for reliable and flexible blackbox optimization. In *First Conference on Automated Machine Learning*, AutoML '22, 2022. xxi, 44

[5] Piali Das, Nikita Ivkin, Tanya Bansal, Laurence Rouesnel, Philip Gautier, Zohar Karnin, Leo Dirac, Lakshmi Ramakrishnan, Andre Perunicic, Iaroslav Shcherbatyi, Wilton Wu, Aida Zolic, Huibin Shen, Amr Ahmed, Fela Winkelmolen, Miroslav Miladinovic, Cedric Archembeau, Alex Tang, Bhaskar Dutt, Patricia Grao, and Kumar Venkateswar. Amazon sagemaker autopilot: a white box automl solution at scale. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, DEEM '20. Association for Computing Machinery, 2020. xxi, 44

[6] Valerio Perrone, Huibin Shen, Aida Zolic, Iaroslav Shcherbatyi, Amr Ahmed, Tanya Bansal, Michele Donini, Fela Winkelmolen, Rodolphe Jenatton, Jean Baptiste Faddoul, Barbara Pogorzelska, Miroslav Miladinovic, Krishnaram Kenthapadi, Matthias Seeger, and Cédric Archambeau. Amazon sagemaker automatic model tuning: Scalable gradient-free optimization. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21. Association for Computing Machinery, 2021. xxi, 41, 44

[7] Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, and Lidong Zhou. Retiarii: A deep learning Exploratory-Training framework. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 919–936. USENIX Association, 2020. xxi, 41, 44, 71

[8] Microsoft neural network intelligence. https://github.com/microsoft/nni, 2023. xxi, 44, 48

[9] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, pages 561–577. USENIX Association, 2018. xxi, 44, 51, 64, 65

[10] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *CoRR*, abs/1807.05118, 2018. xxi, 41, 44, 48, 50, 64, 65

[11] Chatgpt. https://openai.com/blog/chatgpt, 2023. 1, 41

[12] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR '22, pages 10684–10695, June 2022. 1

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. 1, 79

[14] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '16, pages 265–283. USENIX Association, 2016. 1, 139

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32 of *NeurIPS '19*. Curran Associates, Inc., 2019. 1, 41, 51, 75, 79, 90

[16] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, omega, and kubernetes: Lessons learned from three container-management systems over a decade. *Queue*, 14:70–93, 2016. 1, 29, 39, 77, 93, 103

[17] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer Berlin Heidelberg, 2003. 1, 15, 35, 77

[18] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, and Lidong Zhou. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '18, pages 595–610. USENIX Association, 2018. 2, 4, 12, 25, 39, 42, 74, 75, 76, 78, 80, 84, 85, 86, 103

[19] Rui Li, Yufan Xu, Aravind Sukumaran-Rajam, Atanas Rountev, and P. Sadayappan. Analytical characterization and design space exploration for optimization of cnns. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, 2021. 2

[20] Xuan Peng, Xuanhua Shi, Hulin Dai, Hai Jin, Weiliang Ma, Qian Xiong, Fan Yang, and Xuehai Qian. Capuchin: Tensor-based gpu memory management for deep learning. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20. Association for Computing Machinery, 2020. 2

[21] Qinyi Luo, Jiaao He, Youwei Zhuo, and Xuehai Qian. Prague: High-performance heterogeneity-aware asynchronous decentralized training. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20. Association for Computing Machinery, 2020. 2

[22] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33 of *NeurIPS '20*, 2020. 2, 11, 16, 41, 43, 59, 63, 67

[23] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, 2021. 2, 4, 7, 48, 59, 74, 75, 76, 84, 85, 86, 87, 90, 103, 107, 138, 139

[24] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference*, USENIX ATC '19, pages 947–960. USENIX Association, 2019. 2, 4, 12, 13, 16, 17, 18, 22, 25, 37, 38, 42, 59, 76, 84, 86, 90

[25] Nvidia multi-instance gpu. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/, 2023. 3, 12, 45, 58, 65, 78

[26] Nvidia multi-process service. https://docs.nvidia.com/deploy/mps/index.html, 2023. 3, 45, 58, 65, 72, 78

[27] Dror G. Feitelson. Packing schemes for gang scheduling. In *Job Scheduling Strategies for Parallel Processing*, pages 89–110. Springer Berlin Heidelberg, 1996. 3, 11

[28] Nvlink and nvswitch. https://www.nvidia.com/en-us/data-center/nvlink/, 2023. 4, 14, 64

[29] Infiniband networking. https://www.nvidia.com/en-us/networking/products/infiniband/, 2023. 4, 65

[30] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. Horus: Interference-aware and prediction-based scheduling in deep learning systems. *IEEE Transactions on Parallel and Distributed Systems*, 33:88–100, 2022. 4, 74, 75, 76, 78, 80, 84, 85, 94, 103

[31] Peifeng Yu and Mosharaf Chowdhury. Fine-grained gpu sharing primitives for deep learning applications. In *Proceedings of Machine Learning and Systems*, volume 2 of *MLSys '20*, pages 98–111, 2020. 4, 12, 42, 48, 58, 76, 80, 103

[32] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. Fluid: Resource-aware hyperparameter tuning engine. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021. 5, 41, 42, 44, 45, 48, 65, 71

[33] Chieh-Jan Mike Liang, Hui Xue, Mao Yang, Lidong Zhou, Lifei Zhu, Zhao Lucis Li, Zibo Wang, Qi Chen, Quanlu Zhang, Chuanjie Liu, and Wenjun Dai. Autosys: The design and operation of learning-augmented systems. In *2020 USENIX Annual Technical Conference*, USENIX ATC '20, pages 323–336. USENIX Association, 2020. 6, 105, 107, 108, 109, 136

[34] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Interpreting deep learning-based networking systems. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '20, page 154–171. Association for Computing Machinery, 2020. 6, 104, 105, 109, 131, 136

[35] Qinghao Hu, Zhisheng Ye, Meng Zhang, Qiaoling Chen, Peng Sun, Yonggang Wen, and Tianwei Zhang. Hydro: Surrogate-Based hyperparameter tuning service in datacenters. In *17th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '23. USENIX Association, 2023. 7, 138, 139

[36] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs. In *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '23. Association for Computing Machinery, 2023. 7, 42, 48, 138, 139

[37] Qinghao Hu, Harsha Nori, Peng Sun, Yonggang Wen, and Tianwei Zhang. Primo: Practical learning-augmented systems with interpretable models. In *2022 USENIX Annual Technical Conference*, USENIX ATC '22. USENIX Association, 2022. 8, 76, 88, 90, 104, 138, 139

[38] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, 2014. 11

[39] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NeurIPS '14, page 3104–3112. MIT Press, 2014. 11

[40] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198. Association for Computing Machinery, 2016. 11

[41] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *CoRR*, abs/2101.03961, 2021. 11, 16

[42] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '20, pages 289–304. USENIX Association, 2020. 11, 12, 16, 22, 42, 83, 98, 102, 139

[43] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18. Association for Computing Machinery, 2018. 11, 25, 39, 74, 78, 86, 103

[44] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous gpu clusters for deep learning. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20. Association for Computing Machinery, 2020. 11, 22, 103

[45] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '19, pages 485–500. USENIX Association, 2019. 12, 15, 16, 28, 29, 31, 32, 39, 42, 74, 78, 83, 86, 87, 94, 98, 102, 103

[46] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems*

Design and Implementation, OSDI '20, pages 481–498. USENIX Association, 2020. 42, 49, 62, 74, 75, 76, 80, 84, 85, 86, 94, 97, 103

[47] Hanyu Zhao, Zhenhua Han, Zhi Yang, Quanlu Zhang, Fan Yang, Lidong Zhou, Mao Yang, Francis C.M. Lau, Yuqi Wang, Yifan Xiong, and Bin Wang. Hived: Sharing a GPU cluster for deep learning with guarantees. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 515–532. USENIX Association, 2020. 12, 16, 22

[48] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *2018 USENIX Annual Technical Conference*, USENIX ATC '18, pages 533–546. USENIX Association, 2018. 12, 20, 24, 38

[49] Tirthak Patel, Zhengchun Liu, Raj Kettimuthu, Paul Rich, William Allcock, and Devesh Tiwari. Job characteristics on large-scale systems: Long-term analysis, quantification, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. 12, 24, 38

[50] Mengdi Wang, Chen Meng, Guoping Long, Chuan Wu, Jun Yang, Wei Lin, and Yangqing Jia. Characterizing deep learning training workloads on alibaba-pai. In *Proceedings of the 2019 IEEE International Symposium on Workload Characterization*, IISWC '19, pages 189–202, 2019. 13, 18, 22, 39

[51] Lustre. https://www.lustre.org/, 2023. 15

[52] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, page 307–320. USENIX Association, 2006. 15

[53] Memcached. https://memcached.org/, 2023. 15

[54] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI '14, page 583–598. USENIX Association, 2014. 15

[55] Nccl. https://developer.nvidia.com/nccl, 2023. 15

[56] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '21, pages 1–18. USENIX Association, 2021. 16, 42, 45, 74, 75, 78, 94, 101, 103

[57] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and Kyoung-Soo Park. Elastic resource sharing for distributed deep learning. In *18th*

USENIX Symposium on Networked Systems Design and Implementation, NSDI '21. USENIX Association, 2021. 16, 74, 75

[58] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing*, SoCC '13. Association for Computing Machinery, 2013. 17, 28, 29, 39, 93, 103

[59] Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: the next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20. Association for Computing Machinery, 2020. 18, 20, 38

[60] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '12. Association for Computing Machinery, 2012. 85

[61] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference*, USENIX ATC '20, pages 205–218. USENIX Association, 2020. 38

[62] Yanpei Chen, Sara Alspaugh, and Randy Katz. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *Proceedings of the VLDB Endowment*, 5:1802–1813, 2012. 18

[63] Richard L. Moore, Adam Jundt, Leonard K. Carson, Kenneth Yoshimoto, Amin Ghadersohi, and William S. Young. Analyzing throughput and utilization on trestles. In *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the EXtreme to the Campus and Beyond*, XSEDE '12. Association for Computing Machinery, 2012. 22

[64] Gonzalo P. Rodrigo, P.-O. Östberg, Erik Elmroth, Katie Antypas, Richard Gerber, and Lavanya Ramakrishnan. Towards understanding hpc users and systems: A nersc case study. *Journal of Parallel and Distributed Computing*, 111:206–221, 2018. 24, 38

[65] Ru Zhang, Wencong Xiao, Hongyu Zhang, Yu Liu, Haoxiang Lin, and Mao Yang. An empirical study on program failures of deep learning jobs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 1159–1170. Association for Computing Machinery, 2020. 25

[66] Nicole Wolter, Michael O Mccracken, Allan Snavely, Lorin Hochstein, Taiga Nakamura, and Victor Basili. What ' s working in HPC : Investigating HPC User Behavior and Productivity. *CTWatch Quarterly*, 2:1–14, 2006. 27, 38

[67] Sheng Zhang, Zhuzhong Qian, Zhaoyi Luo, Jie Wu, and Sanglu Lu. Burstiness-aware resource reservation for server consolidation in computing clouds. *IEEE Transactions on Parallel and Distributed Systems*, 27:964–977, 2016. 28

[68] Anas A. Youssef and Diwakar Krishnamurthy. Burstiness-aware service level planning for enterprise application clouds. *Journal of Cloud Computing*, 6:1–21, 2017. 28

[69] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. Network-aware scheduling for data-parallel jobs: Plan when you can. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '15, page 407–420. Association for Computing Machinery, 2015. 28, 29, 39, 103

[70] Marcel Blöcher, Lin Wang, Patrick Eugster, and Max Schmidt. Switches for hire: Resource scheduling for data center in-network computing. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 268–285. Association for Computing Machinery, 2021. 28

[71] Alexey Tumanov, Angela Jiang, Jun Woo Park, Michael A. Kozuch, and Gregory R. Ganger. Jamaisvu: Robust scheduling with auto-estimated job runtimes. Technical report, Carnegie Mellon University, 2016. 28, 30, 39, 103

[72] Robert Grandl, Mosharaf Chowdhury, Aditya Akella, and Ganesh Ananthanarayanan. Altruistic scheduling in multi-resource clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '16, pages 65–80. USENIX Association, 2016. 28

[73] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '16, pages 81–97. USENIX Association, 2016.

[74] Abeda Sultana, Li Chen, Fei Xu, and Xu Yuan. E-las: Design and analysis of completion-time agnostic scheduling for distributed deep learning cluster. In *49th International Conference on Parallel Processing*, ICPP '20. Association for Computing Machinery, 2020. 28, 32

[75] Lutz Prechelt. *Early Stopping - But When?*, pages 55–69. Springer Berlin Heidelberg, 1998. 29

[76] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *8th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '11. USENIX Association, 2011. 29, 39, 103

[77] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In

Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems, EuroSys '07, page 59–72. Association for Computing Machinery, 2007. 29, 39, 103

[78] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. Jockey: Guaranteed job latency in data parallel clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, page 99–112. Association for Computing Machinery, 2012.

[79] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. Aria: Automatic resource inference and allocation for mapreduce environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, page 235–244. Association for Computing Machinery, 2011.

[80] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '14, pages 285–300. USENIX Association, 2014.

[81] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '16, pages 363–378. USENIX Association, 2016. 29, 39, 103

[82] Carlo Curino, Djellel E. Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. Reservation-based scheduling: If you're late don't blame us! In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '14, page 1–14. Association for Computing Machinery, 2014. 29, 39, 103

[83] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. Hawk: Hybrid datacenter scheduling. In *2015 USENIX Annual Technical Conference*, USENIX ATC '15, pages 499–510. USENIX Association, 2015.

[84] Sangeetha Abdu Jyothi, Carlo Curino, Ishai Menache, Shravan Matthur Narayanamurthy, Alexey Tumanov, Jonathan Yaniv, Ruslan Mavlyutov, Inigo Goiri, Subru Krishnan, Janardhan Kulkarni, and Sriram Rao. Morpheus: Towards automated slos for enterprise clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '16, pages 117–134. USENIX Association, 2016. 29, 39, 103

[85] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 3sigma: Distribution-based cluster scheduling for runtime uncertainty. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18. Association for Computing Machinery, 2018. 30, 39, 103

[86] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30 of *NeurIPS '17*, 2017. 30, 35, 101, 103, 114

[87] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33:31–88, 2001. 30, 89

[88] Xinxin Mei, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. Energy efficient real-time task scheduling on cpu-gpu hybrid clusters. In *IEEE Conference on Computer Communications*, INFOCOM '17, pages 1–9, 2017. 34, 35, 39

[89] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '09, page 205–216. Association for Computing Machinery, 2009. 34, 39

[90] James Douglas Hamilton. *Time Series Analysis*. Princeton University Press, 2020. 35

[91] Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72:37–45, 2018. 35

[92] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*, ICLR '15, 2015. 35, 52, 79, 125

[93] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22:679–688, 2006. 35

[94] Nvidia-smi. https://developer.nvidia.com/nvidia-system-management-interface, 2023. 36, 80, 83

[95] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The impact of gpu dvfs on the energy and performance of deep learning: An empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, e-Energy '19, page 315–325. Association for Computing Machinery, 2019. 36, 39

[96] Dgx-1 bmc. https://docs.nvidia.com/dgx/dgx1-user-guide, 2023. 37

[97] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. Data center energy consumption modeling: A survey. *IEEE Communications Surveys and Tutorials*, 18:732–794, 2016. 37

[98] Norbert Attig, Paul Gibbon, and Thomas Lippert. Trends in supercomputing: The european path to exascale. *Computer Physics Communications*, 182:2041–2046, 2011. 38

[99] Wayne Joubert and Shi-Quan Su. An analysis of computational workloads for the ornl jaguar system. In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, page 247–256. Association for Computing Machinery, 2012.

[100] Nikolay A. Simakov, Joseph P. White, Robert L. DeLeon, Steven M. Gallo, Matthew D. Jones, Jeffrey T. Palmer, Benjamin Plessinger, and Thomas R. Furlani. A workload analysis of nsf's innovative HPC resources using xdmod. *CoRR*, abs/1801.04306, 2018. 38

[101] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 153–167. Association for Computing Machinery, 2017. 38, 39, 103

[102] Tan N. Le, Xiao Sun, Mosharaf Chowdhury, and Zhenhua Liu. Allox: Compute allocation in hybrid clusters. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20. Association for Computing Machinery, 2020. 39

[103] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3:41–es, 2007. 39

[104] Wenjie Liu, Zhihui Du, Yu Xiao, David A. Bader, and Chen Xu. A waterfall model to achieve energy efficient tasks mapping for large scale GPU clusters. In *25th IEEE International Symposium on Parallel and Distributed Processing, Workshop Proceedings*, IPDPS '11, pages 82–92, 2011. 39

[105] Yuki Abe, Hiroshi Sasaki, Shinpei Kato, Koji Inoue, Masato Edahiro, and Martin Peres. Power and performance characterization and modeling of gpu-accelerated systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, pages 113–122, 2014.

[106] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. Understanding gpu power: A survey of profiling, modeling, and simulation methods. *ACM Computing Surveys*, 49, 2016.

[107] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *42nd International Conference on Parallel Processing*, ICPP '13, pages 826–833, 2013.

[108] Xinxin Mei, Qiang Wang, and Xiaowen Chu. A survey and measurement study of gpu dvfs on energy conservation. *Digital Communications and Networks*, 3: 89–100, 2017.

[109] Qiang Wang and Xiaowen Chu. Gpgpu performance estimation with core and memory frequency scaling. *IEEE Transactions on Parallel and Distributed Systems*, 31:2865–2881, 2020. 39

[110] Vincent Chau, Xiaowen Chu, Hai Liu, and Yiu-Wing Leung. Energy efficient job scheduling with dvfs for cpu-gpu heterogeneous systems. In *Proceedings of the Eighth International Conference on Future Energy Systems*, e-Energy '17, page 1–11. Association for Computing Machinery, 2017. 39

[111] Xinxin Mei, Qiang Wang, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. Energy-aware task scheduling with deadline constraint in dvfs-enabled heterogeneous clusters. *CoRR*, abs/2104.00486, 2021. 39

[112] Richard Liaw, Romil Bhardwaj, Lisa Dunlap, Yitian Zou, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. Hypersched: Dynamic resource reallocation for model development on a deadline. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19. Association for Computing Machinery, 2019. 41, 42, 44, 45, 49, 62, 65, 71

[113] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, 2016. 41, 43, 52, 56, 63, 79

[114] Torchvision new training recipe. https://pytorch.org/blog/how-to-train-state-of-the-art-models-using-torchvision-latest-primitives/, 2023. 41, 56

[115] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *CoRR*, abs/1907.11692, 2019. 41, 42

[116] Alibaba machine learning platform for ai. https://www.alibabacloud.com/product/machine-learning, 2023. 41

[117] Microsoft azure automated machine learning. https://azure.microsoft.com/en-us/products/machine-learning/automatedml, 2023. 41

[118] Ujval Misra, Richard Liaw, Lisa Dunlap, Romil Bhardwaj, Kirthevasan Kandasamy, Joseph E. Gonzalez, Ion Stoica, and Alexey Tumanov. Rubberband: Cloud-based hyperparameter tuning. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 327–342. Association for Computing Machinery, 2021. 41, 42, 44, 45, 49, 62, 71, 139

[119] Lisa Dunlap, Kirthevasan Kandasamy, Ujval Misra, Richard Liaw, Michael Jordan, Ion Stoica, and Joseph E. Gonzalez. Elastic hyperparameter tuning on the cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21. Association for Computing Machinery, 2021. 41, 42, 44, 45, 49, 62, 71

[120] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '19, 2019. 41, 56, 79

[121] Github copilot. https://github.com/features/copilot/, 2023. 41

[122] Duolingo. https://www.duolingo.com/, 2023. 41

[123] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua

Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022. 42

[124] Sameer Kumar, Yu Wang, Cliff Young, James Bradbury, Naveen Kumar, Dehao Chen, and Andy Swing. Exploring the limits of concurrency in ml training on google tpus. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021. 42

[125] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics — Workshop on Challenges & Perspectives in Creating Large Language Models*, ACL-BigScience '22, 2022. 42, 65

[126] John Thorpe, Pengzhan Zhao, Jonathan Eyolfson, Yifan Qiao, Zhihao Jia, Minjia Zhang, Ravi Netravali, and Guoqing Harry Xu. Bamboo: Making preemptible instances resilient for affordable training of large dnns. In *20th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '23. USENIX Association, 2023. 42, 43, 60, 62, 72, 139, 141

[127] Shang Wang, Peiming Yang, Yuxuan Zheng, Xin Li, and Gennady Pekhimenko. Horizontally fused training array: An effective hardware utilization squeezer for training novel deep learning models. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021. 42, 44, 57, 64, 65, 70, 72

[128] Zhisheng Ye, Peng Sun, Wei Gao, Tianwei Zhang, Xiaolin Wang, Shengen Yan, and Yingwei Luo. Astraea: A fair deep learning scheduler for multi-tenant gpu clusters. *IEEE Transactions on Parallel and Distributed Systems*, 2021. 42, 139

[129] Wei Gao, Qinghao Hu, Zhisheng Ye, Peng Sun, Xiaolin Wang, Yingwei Luo, Tianwei Zhang, and Yonggang Wen. Deep learning workload scheduling in gpu datacenters: Taxonomy, challenges and vision. *CoRR*, abs/2205.11913, 2022. 42

[130] Greg Yang and Edward J. Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, pages 11727–11737, 2021. 42, 45, 46, 52, 53, 54

[131] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. In *Advances in Neural Information Processing Systems*, NeurIPS '21, 2021. 42, 45, 46, 52, 53, 55, 56, 64

[132] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems*, MLSys '20, pages 230–246, 2020. 44, 65, 68

[133] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *ICML '18*, pages 1437–1446, 2018. 44, 106, 116

[134] Yimin Huang, Yujun Li, Hanrong Ye, Zhenguo Li, and Zhihua Zhang. Improving model training with multi-fidelity hyperparameter evaluation. In *Proceedings of Machine Learning and Systems*, MLSys '22, 2022.

[135] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 2018. 65

[136] Yang Li, Yu Shen, Jiawei Jiang, Jinyang Gao, Ce Zhang, and Bin Cui. Mfes-hb: Efficient hyperband with multi-fidelity quality measurements. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI '21, 2021.

[137] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NeurIPS '12, page 2951–2959. Curran Associates Inc., 2012. 44, 106, 116

[138] Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research*, 15:3735–3739, 2014.

[139] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Zhi Yang, Ce Zhang, and Bin Cui. Transbo: Hyperparameter optimization via two-phase transfer learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22. Association for Computing Machinery, 2022.

[140] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, and Bin Cui. Volcanoml: speeding up end-to-end automl via scalable search space decomposition. *Proceedings of the VLDB Endowment*, 14:2167–2176, 2021. 44

[141] Fan Lai, Yinwei Dai, Harsha V. Madhyastha, and Mosharaf Chowdhury. ModelKeeper: Accelerating DNN training via automated training warmup. In *20th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '23, pages 769–785. USENIX Association, 2023. 44, 139

[142] Yang Li, Yu Shen, Wentao Zhang, Yuanwei Chen, Huaijun Jiang, Mingchao Liu, Jiawei Jiang, Jinyang Gao, Wentao Wu, Zhi Yang, Ce Zhang, and Bin Cui. Openbox: A generalized black-box optimization service. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21. Association for Computing Machinery, 2021. 44

[143] David Salinas, Matthias Seeger, Aaron Klein, Valerio Perrone, Martin Wistuba, and Cedric Archambeau. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *First Conference on Automated Machine Learning*, AutoML '22, 2022. 44

[144] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. 44, 65, 116

[145] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, ICML '13, 2013. 44

[146] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the International Joint Conference on Artificial Intelligence*, IJCAI '15, 2015. 44

[147] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. 45, 46

[148] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling. *CoRR*, abs/2010.14701, 2020.

[149] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack William Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*, NeurIPS '22, 2022. 45, 46

[150] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, NeurIPS '18, 2018. 45, 52

[151] Sho Yaida. Meta-principled family of hyperparameter scaling strategies. *CoRR*, abs/2210.04909, 2022. 45

[152] Radford M. Neal. *Priors for Infinite Networks*. Springer New York, New York, NY, 1996. 45

[153] Lénaïc Chizat and Francis Bach. On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in Neural Information Processing Systems*, NeurIPS '18, 2018. 46

[154] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 2018. 46

[155] Daniel A. Roberts, Sho Yaida, and Boris Hanin. *The Principles of Deep Learning Theory*. Cambridge University Press, 2022. 46

[156] Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. In *Advances in Neural Information Processing Systems*, NeurIPS '19, 2019. 52

[157] Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. *CoRR*, abs/2006.14548, 2020.

[158] Greg Yang and Etai Littwin. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, pages 11762–11772, 2021.

[159] Greg Yang. Tensor programs iii: Neural matrix laws. *CoRR*, abs/2009.10685, 2021. 46, 52

[160] Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *Proceedings of Machine Learning and Systems*, MLSys '18, 2018. 48, 56, 139

[161] William F. Whitney. Parallelizing neural networks on one gpu with jax, 2023. URL `http://willwhitney.com/parallel-training-jax.html`. 48, 56

[162] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 52

[163] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, ICLR '17, 2017. 52

[164] Meng Zhang, Qinghao Hu, Peng Sun, Yonggang Wen, and Tianwei Zhang. Boosting distributed full-graph gnn training with asynchronous one-bit communication. *CoRR*, abs/2303.01277, 2023. 52

[165] Sergey Ioffe and Christian Szegedy. Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, ICML '15, 2015. 52

[166] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. 52

[167] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30 of *NeurIPS '17*, 2017. 52, 54, 63, 79

[168] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ICML '10, pages 249–256, 2010. 53

[169] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, ICLR '17, 2017. 54

[170] James Reed, Zachary DeVito, Horace He, Ansley Ussery, and Jason Ansel. torch.fx: Practical program capture and transformation for deep learning in python. In *Proceedings of Machine Learning and Systems*, MLSys '22, 2022. 55, 64

[171] Zhiquan Lai, Shengwei Li, Xudong Tang, Keshi Ge, Weijie Liu, Yabo Duan, Linbo Qiao, and Dongsheng Li. Merak: An efficient distributed dnn training framework with automated 3d parallelism for giant foundation models. *CoRR*, abs/2206.04959, 2022. 56

[172] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, ICCV '19, 2019. 56, 63, 79

[173] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, ICLR '15, 2015. 56, 63, 79

[174] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, EMNLP '20, 2020. 56

[175] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. 56

[176] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE International Conference on Computer Vision*, ICCV '21, 2021. 56

[177] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20, 2020. 58, 61, 64, 71

[178] Nvfuser. https://github.com/pytorch/pytorch/projects/30, 2023. 58, 64

[179] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. Taso: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19. Association for Computing Machinery, 2019. 58

[180] Colin Unger, Zhihao Jia, Wei Wu, Sina Lin, Mandeep Baines, Carlos Efrain Quintero Narvaez, Vinay Ramakrishnaiah, Nirmal Prajapati, Pat McCormick, Jamaludin Mohd-Yusof, Xi Luo, Dheevatsa Mudigere, Jongsoo Park, Misha Smelyanskiy, and Alex Aiken. Unity: Accelerating DNN training through joint optimization of algebraic transformations and parallelization. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, pages 267–284. USENIX Association, 2022.

[181] Lianmin Zheng, Chengfan Jia, Minmin Sun, Zhao Wu, Cody Hao Yu, Ameer Haj-Ali, Yida Wang, Jun Yang, Danyang Zhuo, Koushik Sen, Joseph E. Gonzalez, and Ion Stoica. Ansor: Generating high-performance tensor programs for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 863–879. USENIX Association, 2020. 58, 105

[182] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19. Association for Computing Machinery, 2019. 60

[183] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21. Association for Computing Machinery, 2021. 60, 71

[184] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. Dapple: a pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '21. Association for Computing Machinery, 2021. 60

[185] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In *Proceedings of the*

38th International Conference on Machine Learning, ICML '21, pages 7937–7947, 2021. 60

[186] Shigang Li and Torsten Hoefler. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21. Association for Computing Machinery, 2021. 60

[187] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, pages 559–578. USENIX Association, 2022. 61

[188] Nvidia data center gpu manager. https://developer.nvidia.com/dcgm, 2023. 62, 70, 83

[189] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. https://skylion007.github.io/OpenWebTextCorpus/, 2023. 63

[190] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, ICLR '17, 2017. 63, 79

[191] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference*, BMVC '16, 2016. 63

[192] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. 63, 79

[193] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, 2008. 63

[194] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 63

[195] Pytorch fullyshardeddataparallel. https://pytorch.org/docs/stable/fsdp, 2023. 64

[196] Amd epyc 7742 cpu. https://www.amd.com/en/products/cpu/amd-epyc-7742, 2023. 64

[197] Eleutherai gpt-neo 1.3b. https://huggingface.co/EleutherAI/gpt-neo-1.3B, 2023. 65

[198] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2017. 65

[199] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, ICLR '15, 2015. 65

[200] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, ICLR '19, 2019. 65

[201] Pytorch examples. https://github.com/pytorch/examples, 2023. 67

[202] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Jixiang Li, Ji Liu, Ce Zhang, and Bin Cui. Hyper-tune: towards efficient hyper-parameter tuning at scale. *Proceedings of the VLDB Endowment*, 15:1256–1265, 2022. 68

[203] Pytorch automatic mixed precision training. https://pytorch.org/docs/stable/amp, 2023. 69

[204] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2020. 71

[205] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *CoRR*, abs/2205.05198, 2022. 71

[206] Yunfeng Lin, Guilin Li, Xing Zhang, Weinan Zhang, Bo Chen, Ruiming Tang, Zhenguo Li, Jiashi Feng, and Yong Yu. Modularnas: Towards modularized and reusable neural architecture search. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021. 71

[207] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, NeurIPS '12, 2012. 72

[208] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '17, 2017. 72, 79

[209] Sangjin Choi, Inhoe Koo, Jeongseob Ahn, Myeongjae Jeon, and Youngjin Kwon. Envpipe: Performance-preserving dnn training framework for saving energy. In *2023 USENIX Annual Technical Conference*, USENIX ATC '23, 2023. 72

[210] Guanhua Wang, Kehan Wang, Kenan Jiang, XIANGJUN LI, and Ion Stoica. Wavelet: Efficient dnn training with tick-tock scheduling. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021. 72

[211] Gangmuk Lim, Jeongseob Ahn, Wencong Xiao, Youngjin Kwon, and Myeongjae Jeon. Zico: Efficient GPU memory sharing for concurrent DNN training. In *2021 USENIX Annual Technical Conference*, USENIX ATC '21, pages 161–175. USENIX Association, 2021. 72

[212] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Multi-resource interleaving for deep learning training. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '22. Association for Computing Machinery, 2022. 72, 75, 97, 103, 139

[213] H.T. Kung, Bradley McDanel, and Sai Qian Zhang. Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19. Association for Computing Machinery, 2019. 74

[214] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. Recssd: near data processing for solid state drive based recommendation inference. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21. Association for Computing Machinery, 2021. 74

[215] Zhengda Bian, Shenggui Li, Wei Wang, and Yang You. Online evolutionary batch size orchestration for scheduling deep learning workloads in gpu clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21. Association for Computing Machinery, 2021. 74, 75, 78, 103

[216] Dharma Shukla, Muthian Sivathanu, Srinidhi Viswanatha, Bhargav Gulavani, Rimma Nehme, Amey Agrawal, Chen Chen, Nipun Kwatra, Ramachandran Ramjee, Pankaj Sharma, Atul Katiyar, Vipul Modi, Vaibhav Sharma, Abhishek Singh, Shreshth Singhal, Kaustubh Welankar, Lu Xun, Ravi Anupindi, Karthik Elangovan, Hasibur Rahman, Zhou Lin, Rahul Seetharaman, Cheng Xu, Eddie Ailijiang, Suresh Krishnappa, and Mark Russinovich. Singularity: Planet-scale, preemptive and elastic scheduling of ai workloads. *CoRR*, abs/2202.07848, 2022. 75

[217] Shaoqi Wang, Oscar J Gonzalez, Xiaobo Zhou, Thomas Williams, Brian D Friedman, Martin Havemann, and Thomas Woo. An efficient and non-intrusive gpu scheduling framework for deep learning training systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. 75

[218] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. Aryl: An elastic cluster scheduler for deep learning. *CoRR*, abs/2202.07896, 2022. 75

[219] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, ICLR '17, 2017. 75

[220] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In *20th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '23. USENIX Association, 2023. 75, 97, 102, 139

[221] Jensen Huang. Nvidia gtc 2022 keynote. https://www.nvidia.com/gtc/keynote/, 2023. 75

[222] Haoyu Wang, Zetian Liu, and Haiying Shen. Job scheduling for large-scale machine learning clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, CoNEXT '20. Association for Computing Machinery, 2020. 75

[223] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. Dl2: A deep learning-driven scheduler for deep learning clusters. *IEEE Transactions on Parallel and Distributed Systems*, 32:1947–1960, 2021. 75

[224] Yunteng Luan, Xukun Chen, Hanyu Zhao, Zhi Yang, and Yafei Dai. Sched^2: Scheduling deep learning training via deep reinforcement learning. In *2019 IEEE Global Communications Conference*, GLOBECOM '19, pages 1–7. IEEE, 2019. 75

[225] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. Solving large-scale granular resource allocation problems efficiently with pop. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21. Association for Computing Machinery, 2021. 75, 97

[226] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. Chronus: A novel deadline-aware scheduler for deep learning training jobs. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21. Association for Computing Machinery, 2021. 75, 83, 98, 102, 139

[227] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. Looking beyond GPUs for DNN scheduling on Multi-Tenant clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, pages 579–596. USENIX Association, 2022. 75, 103, 139

[228] Luping Wang, Qizhen Weng, Wei Wang, Chen Chen, and Bo Li. Metis: Learning to schedule long-running applications in shared container clusters at scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20, 2020. 75

[229] Zhaoyun Chen, Lei Luo, Wei Quan, Mei Wen, and Chunyuan Zhang. Poster abstract: Deep learning workloads scheduling with reinforcement learning on gpu clusters. In *IEEE Conference on Computer Communications Workshops*, INFOCOM '19, pages 1023–1024, 2019. 75

[230] Rong Gu, Yuquan Chen, Shuai Liu, Haipeng Dai, Guihai Chen, Kai Zhang, Yang Che, and Yihua Huang. Liquid: Intelligent resource estimation and network-efficient scheduling for deep learning jobs on distributed gpu clusters. *IEEE Transactions on Parallel and Distributed Systems*, 2021. 75

[231] Sejin Kim and Yoonhee Kim. Co-scheml: Interference-aware container co-scheduling scheme using machine learning application profiles for gpu clusters. In *2020 IEEE International Conference on Cluster Computing*, CLUSTER '20, 2020. 75

[232] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. Lemna: Explaining deep learning based security applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 364–379. Association for Computing Machinery, 2018. 75, 104, 105, 107, 109, 111, 136

[233] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, pages 206–215, 2019. 101, 106, 110, 112

[234] Mikel Landajuela, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, Jacob F Pettit, and Daniel Faissol. Discovering symbolic policies with deep reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, pages 5979–5989, 2021. 75, 109

[235] Yixin Bao, Yanghua Peng, and Chuan Wu. Deep learning-based job placement in distributed machine learning clusters. In *IEEE Conference on Computer Communications*, INFOCOM '19, pages 505–513, 2019. 76, 80, 84, 85

[236] Alex Krizhevsky. The cifar10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html, 2023. 79

[237] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '18, 2018. 79

[238] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *ICML '19*, pages 6105–6114, 2019. 79

[239] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*, ICLR '16, 2016. 79

[240] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2016. 79

[241] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015. 79

[242] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. 79

[243] Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, 2016. 79

[244] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, ICML '18, 2018. 79

[245] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, 2017. 79

[246] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5:1–19, 2015. 79

[247] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. Linnos: Predictability on unpredictable flash storage with a light neural network. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 173–190. USENIX Association, 2020. 88, 105, 106, 107, 109, 114, 117, 118, 119

[248] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Wadsworth, 1984. 88, 115, 139

[249] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '13. Association for Computing Machinery, 2013. 88, 113, 114

[250] Harsha Nori, Rich Caruana, Zhiqi Bu, Judy Hanwen Shen, and Janardhan Kulkarni. Accuracy, interpretability, and differential privacy via explainable boosting. In *Proceedings of the 38th International Conference on Machine Learning*, ICML '21, pages 8227–8237, 2021. 88, 106, 114, 118

[251] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 2007. 89

[252] Miriam Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and Edward Silverman. An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics*, 26, 1955. 90, 118

[253] grpc: An rpc library and framework. https://grpc.io/, 2023. 90

[254] Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2023. 94, 103, 139

[255] Leo Breiman. Random forests. *Machine learning*, pages 5–32, 2001. 101

[256] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. Association for Computing Machinery, 2016. 101, 125, 127

[257] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521: 436–444, 2015. 101

[258] Han Zhao, Weihao Cui, Quan Chen, Jingwen Leng, Kai Yu, Deze Zeng, Chao Li, and Minyi Guo. Coda: Improving resource utilization by slimming and co-locating dnn and cpu jobs. In *2020 IEEE 40th International Conference on Distributed Computing Systems*, ICDCS '20, pages 853–863. IEEE, 2020. 103

[259] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21. Association for Computing Machinery, 2021. 104, 105, 136

[260] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. Unicorn: reasoning about configurable system performance through the lens of causality. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22. Association for Computing Machinery, 2022. 104

[261] Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G. Edward Suh, and Christina Delimitrou. Sinan: Ml-based and qos-aware resource management for cloud microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21. Association for Computing Machinery, 2021. 104, 105, 107

[262] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. Association for Computing Machinery, 2016. 104, 107, 109, 111, 135

[263] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. Sage: practical and scalable ml-driven performance debugging in microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21. Association for Computing Machinery, 2021. 104

[264] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, Murali Chintalapati, and Dongmei Zhang. Improving service availability of cloud systems by predicting disk error. In *2018 USENIX Annual Technical Conference*, USENIX ATC '18, pages 481–494. USENIX Association, 2018. 105

[265] Xingda Wei, Rong Chen, and Haibo Chen. Fast rdma-based ordered key-value store using remote learned cache. In *14th USENIX Symposium on Operating*

Systems Design and Implementation, OSDI '20, pages 117–135. USENIX Association, 2020. 105, 107, 109

[266] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. Automated smartnic offloading insights for network functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21. Association for Computing Machinery, 2021. 105, 106, 107, 108, 109, 118, 124, 127, 128

[267] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. Network planning with deep reinforcement learning. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '21. Association for Computing Machinery, 2021.

[268] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation*, NSDI '20, 2020. 105

[269] Liangyi Gong, Zhenhua Li, Feng Qian, Zifan Zhang, Qi Alfred Chen, Zhiyun Qian, Hao Lin, and Yunhao Liu. Experiences of landing machine learning onto market-scale mobile malware detection. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20. Association for Computing Machinery, 2020. 105

[270] Xueyuan Han, Xiao Yu, Thomas Pasquier, Ding Li, Junghwan Rhee, James Mickens, Margo Seltzer, and Haifeng Chen. SIGL: Securing software installations through deep graph learning. In *30th USENIX Security Symposium*, USENIX Security '21, pages 2345–2362. USENIX Association, 2021.

[271] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing functions in binaries with neural networks. In *24th USENIX Security Symposium*, USENIX Security '15. USENIX Association, 2015. 105

[272] Size Zheng, Yun Liang, Shuo Wang, Renze Chen, and Kaiwen Sheng. Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20. Association for Computing Machinery, 2020. 105

[273] Riyadh Baghdadi, Massinissa Merouani, Mohamed-Hicham Leghettas, Kamel Abdous, Taha Arbaoui, Karima Benatchba, and Saman Amarasinghe. A deep learning based cost model for automatic code optimization. In *Proceedings of Machine Learning and Systems*, MLSys '21, 2021. 105, 108

[274] Haoran Qiu, Subho S. Banerjee, Saurabh Jha, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. FIRM: An intelligent fine-grained resource management framework for slo-oriented microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 805–825. USENIX Association, 2020. 105, 107

[275] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. Rlscheduler: An automated hpc batch job scheduler using reinforcement learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. 105

[276] Shane Bergsma, Timothy Zeyl, Arik Senderovich, and J. Christopher Beck. Generating complex, realistic cloud workloads using recurrent neural networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21. Association for Computing Machinery, 2021. 105, 108

[277] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. Generating data series query workloads. *The VLDB Journal*, 27: 823–846, 2018.

[278] Vsevolod Livinskii, Dmitry Babokin, and John Regehr. Random testing for c and c++ compilers with yarpgen. *Proceedings of the ACM on Programming Languages*, page 1–25, 2020. 105, 108, 124, 128

[279] Erick Carvajal Barboza, Sara Jacob, Mahesh Ketkar, Michael Kishinevsky, Paul Gratz, and Jiang Hu. Automatic microprocessor performance bug detection. In *International Symposium on High Performance Computer Architecture*, HPCA '21, 2021. 105, 108

[280] Jiachen Wang, Ding Ding, Huan Wang, Conrad Christensen, Zhaoguo Wang, Haibo Chen, and Jinyang Li. Polyjuice: High-performance transactions via learned concurrency control. In *15th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '21, pages 198–216. USENIX Association, 2021. 105, 109

[281] Shengwen Liang, Ying Wang, Youyou Lu, Zhe Yang, Huawei Li, and Xiaowei Li. Cognitive SSD: A deep learning engine for in-storage data retrieval. In *2019 USENIX Annual Technical Conference*, USENIX ATC '19, pages 395–410. USENIX Association, 2019. 105, 107, 109

[282] Xinyang Zhang, Ningfei Wang, Hua Shen, Shouling Ji, Xiapu Luo, and Ting Wang. Interpretable deep learning under fire. In *29th USENIX Security Symposium*, USENIX Security '20, 2020. 105

[283] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, 2021. 107, 109, 136

[284] Tomer Eliyahu, Yafim Kazak, Guy Katz, and Michael Schapira. Verifying learning-augmented systems. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '21. Association for Computing Machinery, 2021. 105, 109, 112, 136

[285] Christoph Molnar. *Interpretable Machine Learning*. 2019. 106, 110, 112

[286] José Jiménez-Luna, Francesca Grisoni, and Gisbert Schneider. Drug discovery with explainable artificial intelligence. *Nature Machine Intelligence*, pages 573–584, 2020. 106

[287] Cynthia Rudin and Berk Ustun. Optimized scoring systems: Toward trust in machine learning for healthcare and criminal justice. *INFORMS Journal on Applied Analytics*, 48:449–466, 2018. 106

[288] Shanka Subhra Mondal, Nikhil Sheoran, and Subrata Mitra. Scheduling of time-varying workloads using reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI '21, 2021. 106

[289] Yuanxiang Gao, Li Chen, and Baochun Li. Spotlight: Optimizing device placement for training deep neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, ICML '18, pages 1676–1684, 2018.

[290] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, ICML'17, 2017.

[291] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of deep reinforcement learning in communications and networking: A survey. *IEEE Communications Surveys Tutorials*, 2019. 106

[292] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, page 197–210. Association for Computing Machinery, 2017. 106, 107, 109, 118, 130

[293] Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, page 452–465. Association for Computing Machinery, 2017. 107

[294] Pradeep Ambati, Inigo Goiri, Felipe Frujeri, Alper Gun, Ke Wang, Brian Dolan, Brian Corell, Sekhar Pasupuleti, Thomas Moscibroda, Sameh Elnikety, Marcus Fontoura, and Ricardo Bianchini. Providing slos for resource-harvesting vms in cloud platforms. In *14th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '20, pages 735–751. USENIX Association, 2020. 107

[295] Ji Zhang, Ping Huang, Ke Zhou, Ming Xie, and Sebastian Schelter. Hddse: Enabling high-dimensional disk state embedding for generic failure detection system of heterogeneous disks in large data centers. In *2020 USENIX Annual Technical Conference*, USENIX ATC '20, pages 111–126. USENIX Association, 2020. 107

[296] Xu Li, Feilong Tang, Jiacheng Liu, Laurence T. Yang, Luoyi Fu, and Long Chen. AUTO: Adaptive congestion control based on multi-objective reinforcement learning for the satellite-ground integrated network. In *2021 USENIX Annual Technical Conference*, USENIX ATC '21, pages 611–624. USENIX Association, 2021. 107

[297] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 270–288. Association for Computing Machinery, 2019. 108

[298] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A runtime-based computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference*, USENIX ATC '21, pages 503–521. USENIX Association, 2021. 108, 135

[299] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch. *CoRR*, abs/2009.07896, 2020. 109, 135

[300] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, volume 30 of *NeurIPS '17*. Curran Associates, Inc., 2017. 109, 135

[301] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. In *Proceedings of 1993 International Conference on Neural Networks*, IJCNN '93, 1993. 109

[302] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, ICML '15, pages 2048–2057, 2015. 110

[303] Sarthak Jain and Byron C. Wallace. Attention is not explanation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '19, 2019. 110

[304] Sarah Wiegreffe and Yuval Pinter. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, EMNLP '19, 2019. 110

[305] Awesome-ml-for-system. `https://github.com/S-Lab-System-Group/Awesome-ML-for-System`, 2022. 112, 135

[306] Trevor Hastie and Robert Tibshirani. Generalized additive models: Some applications. *Journal of the American Statistical Association*, 82:371–386, 1987. 113

[307] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *CoRR*, abs/1909.09223, 2019. 114

[308] Yin Lou, Rich Caruana, and Johannes Gehrke. Intelligible models for classification and regression. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12. Association for Computing Machinery, 2012. 114

[309] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *CoRR*, abs/2103.11251, 2021. 114

[310] Djob Mvondo, Mathieu Bacou, Kevin Nguetchouang, Lucien Ngale, Stéphane Pouget, Josiane Kouam, Renaud Lachaize, Jinho Hwang, Tim Wood, Daniel Hagimont, Noël De Palma, Bernabé Batchakui, and Alain Tchana. Ofc: An opportunistic caching system for faas platforms. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 228–244. Association for Computing Machinery, 2021. 114

[311] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, NeurIPS '14, 2014. 117

[312] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. 117

[313] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, volume 31 of *NeurIPS '18*, 2018. 117

[314] Aaron M. Roth, Nicholay Topin, Pooyan Jamshidi, and Manuela Veloso. Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy. *CoRR*, abs/1907.01180, 2019.

[315] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, AISTATS '20, pages 1855–1865, 2020. 117

[316] Nilotpal Chakravarti. Isotonic median regression: A linear programming approach. *Mathematics of Operations Research*, 14:303–308, 1989. 118

[317] Arnaud Van Looveren and Janis Klaise. Interpretable counterfactual explanations guided by prototypes. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ECML-PKDD '21, 2021. 118

[318] Keinosuke Fukunaga and Patrenahalli M. Narendra. A branch and bound algorithms for computing k-nearest neighbors. *IEEE Transactions on Computers*, 24:750–753, 1975. 118

[319] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbah, Alex Rocha, and Joe Stubbs. Lessons learned from the chameleon testbed. In *2020 USENIX Annual Technical Conference*, USENIX ATC '20, pages 219–233. USENIX Association, 2020. 119

[320] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of cloudlab. In *2019 USENIX Annual Technical Conference*, USENIX ATC '19, pages 1–14. USENIX Association, 2019. 119

[321] Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New Support Vector Algorithms. *Neural Computation*, 12:1207–1245, 2000. 125

[322] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '16, 2016. 125

[323] Yujie Fan, Yanfang Ye, and Lifei Chen. Malicious sequential pattern mining for automatic malware detection. *Expert Systems with Applications*, 52:16–25, 2016. 129

[324] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *ICML '16*, pages 1928–1937. PMLR, 2016. 130

[325] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13. Association for Computing Machinery, 2013. 131

[326] Federal communications commission. https://www.fcc.gov/reports-research/reports/measuring-broadband-america, 2023. 131

[327] Zili Meng, Yaning Guo, Yixin Shen, Jing Chen, Chao Zhou, Minhu Wang, Jia Zhang, Mingwei Xu, Chen Sun, and Hongxin Hu. Practically deploying heavyweight adaptive bitrate algorithms with teacher-student learning. *IEEE/ACM Transactions on Networking*, 29:723–736, 2021. 131

[328] Dash.js: Javascript player. https://github.com/Dash-Industry-Forum/dash.js, 2023. 131

[329] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: evidence from a large

video streaming service. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '14, 2014. 131

[330] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE Conference on Computer Communications*, INFOCOM '16. 131

[331] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '15. Association for Computing Machinery, 2015. 131

[332] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41:1–58, 2009. 135

[333] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, AAAI '16, 2016. 136

[334] Yingtian Tang, Han Lu, Xijun Li, Lei Chen, Mingxuan Yuan, and Jia Zeng. Learning-aided heuristics design for storage system. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21. Association for Computing Machinery, 2021. 136

[335] Maximilian Grüner, Melissa Licciardello, and Ankit Singla. Reconstructing proprietary video streaming algorithms. In *2020 USENIX Annual Technical Conference*, USENIX ATC '20. USENIX Association, 2020. 136

[336] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *Proceedings of the 37th International Conference on Machine Learning*, ICML '20, pages 6150–6160, 2020. 139

[337] Hongzi Mao, Parimarjan Negi, Akshay Narayan, Hanrui Wang, Jiacheng Yang, Haonan Wang, Ryan Marcus, Ravichandra Addanki, Mehrdad Khani Shirkoohi, Songtao He, Vikram Nathan, Frank Cangialosi, Shaileshh Bojja Venkatakrishnan, Wei-Hung Weng, Song Han, Tim Kraska, and Mohammad Alizadeh. Park: An open platform for learning-augmented computer systems. In *Advances in Neural Information Processing Systems*, NeurIPS '19, 2019. 140

[338] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23. Association for Computing Machinery, 2023. 140

[339] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation*, OSDI '22, pages 521–538. USENIX Association, 2022. 140

[340] Insu Jang, Zhenning Yang, Zhen Zhang, Xin Jin, and Mosharaf Chowdhury. Oobleck: Resilient distributed training of large models using pipeline templates. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23. Association for Computing Machinery, 2023. 141

[341] Zhuang Wang, Zhen Jia, Shuai Zheng, Zhen Zhang, Xinwei Fu, T. S. Eugene Ng, and Yida Wang. Gemini: Fast failure recovery in distributed training with in-memory checkpoints. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23. Association for Computing Machinery, 2023. 141