Lucid: A Non-intrusive, Scalable and Interpretable Scheduler for Deep Learning Training Jobs

Qinghao Hu*
S-Lab, NTU
& Shanghai AI Laboratory
Singapore & China

Meng Zhang*
S-Lab,
Nanyang Technological University
Singapore

Peng Sun
SenseTime Research
& Shanghai AI Laboratory
China

Yonggang Wen
Nanyang Technological University
Singapore

Tianwei Zhang
Nanyang Technological University
Singapore

ABSTRACT

While recent deep learning workload schedulers exhibit excellent performance, it is arduous to deploy them in practice due to some substantial defects, including inflexible intrusive manner, exorbitant integration and maintenance cost, limited scalability, as well as opaque decision processes. Motivated by these issues, we design and implement Lucid, a non-intrusive deep learning workload scheduler based on interpretable models. It consists of three innovative modules. First, a two-dimensional optimized profiler is introduced for efficient job metric collection and timely debugging job feedback. Second, Lucid utilizes an indolent packing strategy to circumvent interference. Third, Lucid orchestrates resources based on estimated job priority values and sharing scores to achieve efficient scheduling. Additionally, Lucid promotes model performance maintenance and system transparent adjustment via a well-designed system optimizer. Our evaluation shows that Lucid reduces the average job completion time by up to 1.3× compared with state-of-the-art preemptive scheduler Tiresias. Furthermore, it provides explicit system interpretations and excellent scalability for practical deployment.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Computing methodologies** → **Planning and scheduling**.

KEYWORDS

Cluster Management, Workload Scheduling, Machine Learning

ACM Reference Format:
Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2023. Lucid: A Non-intrusive, Scalable and Interpretable Scheduler for Deep Learning Training Jobs. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '23), March 25–29, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3575693.3575705

*Equal Contribution.

ASPLOS '23, March 25–29, 2023, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9916-6/23/03.
https://doi.org/10.1145/3575693.3575705

1 INTRODUCTION

Over the past decades, Deep Learning (DL) presents incredible performance and rapid popularity across many applications, including image classification [54], recommendation [96], etc. To facilitate DL model development, IT companies and research institutes often build large-scale multi-tenant DL clusters [31, 42, 98]. 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 [11, 31, 42, 73, 76, 97, 98, 100], 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 [97], elasticity [44] and adaptive training [76]. 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 [44, 67, 73, 76, 97]. 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 [84], "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 [44, 84, 97] require the modification of the source code of the underlying DL frameworks (e.g., Pytorch [71]) or CUDA library [94]. 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 [11, 57, 76] 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 [51, 108]. In commercial applications, minor quality improvement drives a significant increase in customer engagement and company profits [43]. Therefore, developers are not prone to adopt this mechanism due to the degradation issue.

Second, plenty of schedulers adopt machine learning (ML) based methods [42, 60, 74, 92, 100] or optimization-based methods [30, 65–67, 107] 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 [19, 30, 66, 67, 74, 92, 93] meet the scalability bottleneck when deployed in production-level systems. For instance, Gavel [67] spends thousands of seconds solving a 2048-job allocation problem through linear programming, which takes too long to meet the real-time requirement [66]. Reinforcement Learning (RL) based schedulers also confront the same issue: Metis [93] 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) [32, 52], Gradient Boosting Decision Tree (GBDT) [42, 100] and RL [74, 92]. 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 [33, 55, 81]. 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 [41].

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 [42, 48, 95], existing DL schedulers pack jobs to increase the utilization through an intrusive manner [10, 67, 97, 100, 103]. 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 [42, 95], 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). 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.

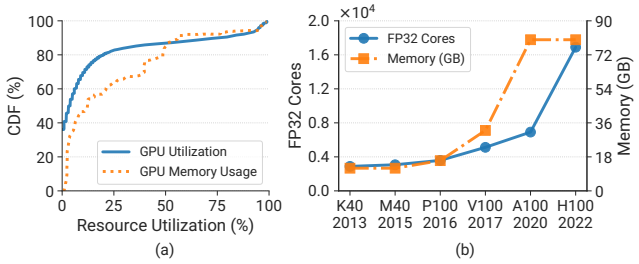

Figure 1: Background. (a) GPU utilization distribution in an Alibaba cluster [98]. (b) Exponential growth of NVIDIA datacenter GPU capability. x-axis: GPU name & release year.

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 [42] and Microsoft [48]. 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:

• **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 [101], Kubernetes [15]). 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~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.

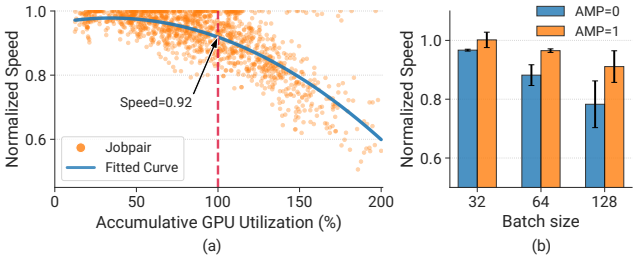

Figure 2: Motivation. (a) Accumulated GPU utilization of colocated jobpairs against average speeds. (b) Average effect of batch size and mixed-precision to packing performance.

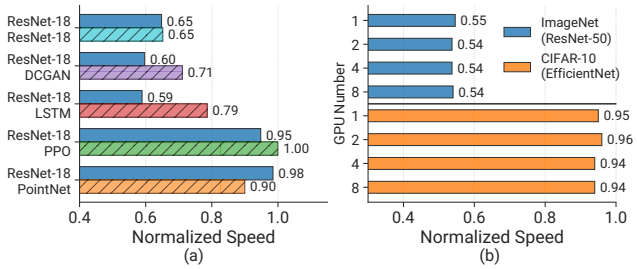

Figure 3: Packing Examples. (a) Colocate with ResNet-18. (b) Two same jobs packing with different GPU numbers.

2 BACKGROUND AND MOTIVATION

In this section, we first provide a brief introduction to the essential terminologies of DL training and cluster scheduling. Then we highlight the characteristics of DL clusters and job colocation that inspire the design of Lucid.

2.1 Background

DL Training. A DL model learns its parameters (i.e. weights) in an *iterative* process [58, 97]. In each iteration, it operates on a batch of labeled data to update model weights through gradient descent. The whole training process usually consists of numerous mini-batch iterations and can last for hours to days, which can be preempted and resumed via checkpoints [61, 72]. Based on the repetitive pattern, operators can profile a few iterations to obtain the resource utilization features of the job. Unlike prior profiling-based DL job schedulers [30, 31, 62] that rely on intrusive libraries to inspect job execution status, Lucid collects metrics non-intrusively.

DL Cluster Scheduling. It is a common practice for tech companies and research institutes to build multi-tenant DL clusters to facilitate DL model development. In many companies [42, 48, 95], the cluster is usually divided into several Virtual Clusters (VCs) dedicated to different product groups. Users submit DL training jobs into the cluster with related configurations (e.g. GPU demand, CPU demand, job name).

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 [11, 31, 73, 76, 97, 98, 100] 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 [5] or MIG [4] technologies. (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 [11, 76]. However, they have several significant drawbacks as mentioned in §1 (**G1~G3**).

2.2 Characteristics of DL Clusters

Low GPU Utilization. Recent works [97, 98, 100, 103] show a common phenomenon that most GPUs are underutilized in DL clusters. Figure 1 (a) shows the Cumulative Distribution Function (CDF) of one-week GPU usage statistics collected from an Alibaba datacenter [98]. 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 (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.

Table 1: Summary of models and datasets used in our experiments. *AMP*: Enable/Disable mixed precision training.

Task	Model	Dataset	Batch size	AMP
✻	ResNet-50 [37]	ImageNet [23]	32, 64, 128	+/-
✻	MobileNetV3 [40]	ImageNet [23]	32, 64, 128	+/-
✻	ResNet-18 [37]	CIFAR-10 [53]	32, 64, 128	+/-
✻	MobileNetV2 [82]	CIFAR-10 [53]	32, 64, 128	+/-
✻	EfficientNet [86]	CIFAR-10 [53]	32, 64, 128	+/-
✻	VGG-11 [85]	CIFAR-10 [53]	32, 64, 128	+/-
❋	DCGAN [77]	LSUN [102]	32, 64, 128	+/-
❊	PointNet [75]	ShapeNet [16]	32, 64, 128	+/-
♦	BERT [24]	SQuAD [78]	32	+/-
✦	LSTM [9]	Wikitext2 [64]	64, 128	+/-
◆	Transformer [88]	Multi30k [25]	32, 64	-
❖	PPO [83]	LunarLander	32, 64, 128	-
❖	TD3 [28]	BipedalWalker	32, 64, 128	-
★	NeuMF [38]	MovieLens [36]	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

High-skewed Workload Distribution. Real-world production DL clusters [42, 48, 95] present similar workload distributions: (1) *Small-scale*. Over 95% jobs are single-node jobs (within 4/8 GPUs) in Microsoft [48] and SenseTime [42]. (2) *Recurring*. Most jobs (~90%) are recurring hyperparameter searching jobs [95, 104]. (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.

2.3 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 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 (§4.1) equipped with NVIDIA RTX3090 GPUs and implemented with Pytorch 1.10 [71].

Figure 2 (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 3 (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 (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 3 (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 2 (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 [67].

Non-intrusive Interference-aware Job Packing. All of existing packing-enabled DL schedulers rely on the intrusive paradigm. Specifically, they modify DL frameworks [97, 98, 103] or require user-code adaptation [10, 67, 100] 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 2 (a)) and the packing strategy is applicable to all single-node jobs (Figure 3 (b)), covering over 95% workloads (§2.2). 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 [6, 100]. 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).

Lucid 由于需要考虑单 GPU 混布，因此考虑了单位时间内模型对 kernels / memory read 的占用比例，作为 GPU / GPU memory 的 utilization

Job Duration Estimation. Recent DL cluster analysis works from SenseTime [42] and Alibaba [95] 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.

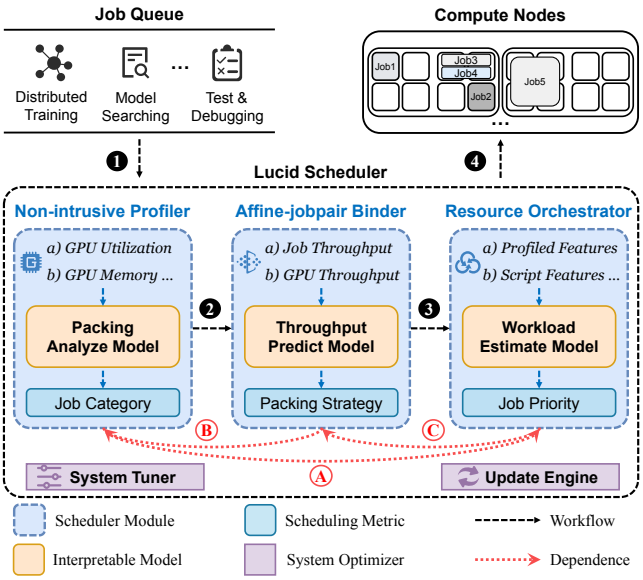

Figure 4: Overview of Lucid system architecture. Each module contains an interpretable model for key metric prediction. System optimizers are applicable to all components tuning. Scheduling workflow and module dependencies are represented by black and red arrows respectively.

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.

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

Algorithm 1 Space-aware Profiling

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

```
1:  procedure SPACE-AWARE PROFILE(J, Q)
2:      if J.gpu ≤ N_prof then                          ▷Job Scale limit
3:          Enqueue J to Q
4:      SortJobGPUNum(Q)                                ▷Sort by Least GPU First
5:      CheckRunningJobs(T_prof)                        ▷Evict Overtime Running Jobs
6:      for all Job ∈ Q do
7:          if Consolidate(Job) is True then
8:              ConsolidateAllocate(Job)                ▷Job Start Profiling
9:              Dequeue Job from Q
10:             Non-intrusiveProfile(Job)
11:         else
12:             break
```

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 (§4.5). We depict their interactions in Figure 4 with red arrows: Ⓐ *Orchestrator* adopts features from *Profiler* for better duration estimation. Lucid cannot precisely match previous recurrent jobs without profiled features. Ⓑ 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*. Ⓒ *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.

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 [6] or DCGM [3] in a non-intrusive way. Then the profiler sends these features to the *Packing Analyze Model* (§3.5.1), 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.

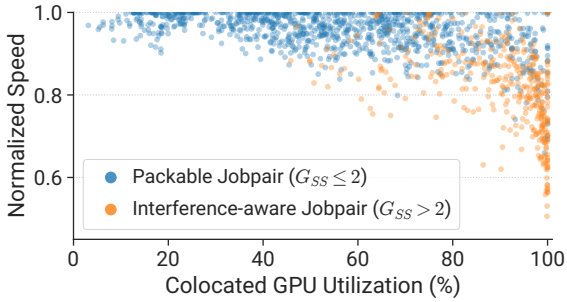

Figure 5: Indolent Packing. Lucid non-intrusively determines whether jobpairs are suitable for colocated execution (Blue Points) or should be exclusive execution (Orange Points).

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 [30, 31, 62]. By prioritizing jobs that request fewer resources, the head-of-line (HOL) blocking problem of small-scale profiling clusters can be efficiently solved. Algorithm 1 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 [42].

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 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 [100], our profiling mechanism possesses the following superiorities: (a) *Timely Feedback*. Plenty of short-term debugging jobs suffer from severe queuing delays (§2.2) due to the runtime-agnostic scheduling paradigm of currently deployed clusters [42, 48, 95]. Whilst Lucid's profiler can well resolve this issue and improve the job fairness. (b) *Effortless*. Lucid does not

Algorithm 2 Lucid Resource Orchestrator

Input: Job Queue: Q, Running Jobs: J

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

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.

3.3 Affine-Jobpair Binder

Different from previous packing-enabled schedulers [10, 67, 97, 98, 100] that apply user-code or DL framework intrusive approaches to identify jobpairs with interference, 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 [67]; (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 5 depicts the binder decisions of all possible jobpair combinations listed in Table 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 [10, 67, 97, 98, 100] usually keep a fixed strategy on job packing without cluster-wide awareness. However, most clusters [42, 79] present diurnal patterns on the job submission rate (throughput) and cluster utilization. When clusters are relatively idle, the ignorance of cluster throughput may cause unnecessary job packing and prolong the job training progress.

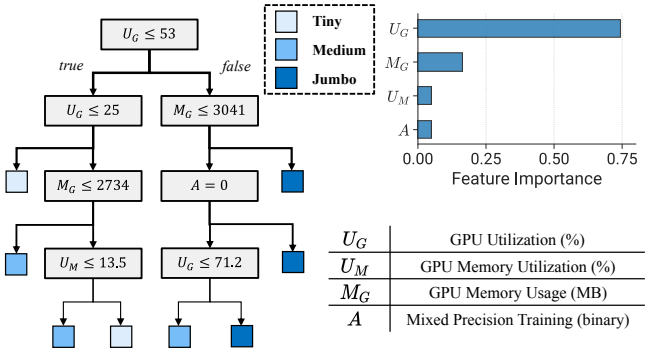

Figure 6: Packing Analyze Model. *Left*: Visualization and interpretation. *Right*: Feature importance and notation.

For this reason, we develop *Throughput Predict Model* (§3.5.2) 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.

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 [97]. The rule of thumb is to prioritize short-term jobs like the Shortest-Job-First (SJF) policy[31], whereas it is impossible to obtain perfect job duration information in reality. Besides, previous intrusive prediction paradigm [67, 73, 97] (i.e. iteration time measurement) can be misleading due to the high cancellation and failure rates of DL training jobs [42, 48]. However, as mentioned in §2.3, a majority of workloads are repetitive and we can leverage prior data to train *Workload Estimate Model* (§3.5.3) to provide job duration estimations for scheduling.

Resource Orchestrator comprehensively considers both temporal and spatial aspects of DL jobs. Algorithm 2 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 [31, 42]. 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 [42]. Further, after

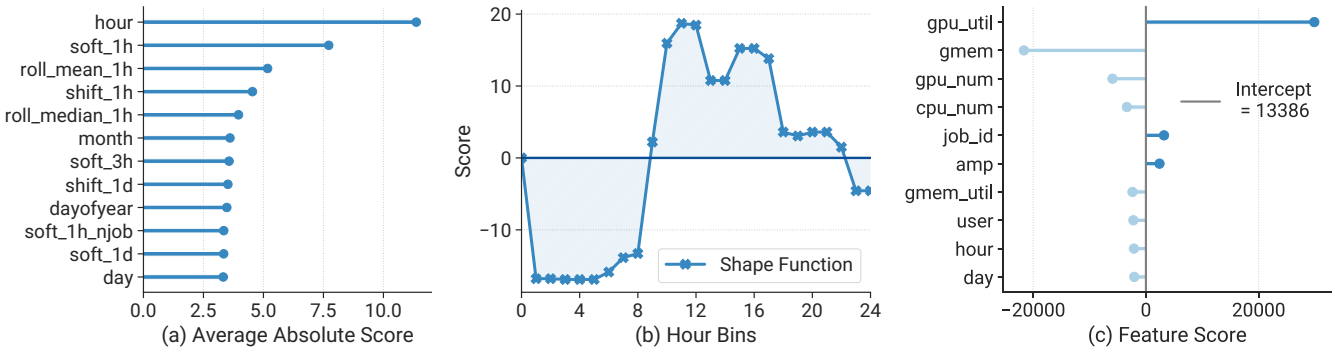

Figure 7: Throughput Predict Model (a & b): Global interpretation of overall feature importance and the learned shape function of *hour* (blue line). Workload Estimate Model (c): Local interpretation of features' contribution for one prediction.

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.

3.5 Interpretable Models

In order to provide accurate prediction and transparent interpretation for the cluster scheduling, Lucid employs Primo [41] interpretable models as the foundation for each scheduler module.

3.5.1 Packing Analyze Model. Inspired by LinnOS [35], 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 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 [41]. We leverage *minimal cost-complexity pruning* [14] to prune the learned tree to obtain a compact and accurate model.

Interpretation: Figure 6 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.

3.5.2 Throughput Predict Model. We adopt a novel additive model algorithm GA^{2}M [59, 69] for cluster throughput prediction. GA^{2}M contains a series of *shape functions* $f(\cdot)$ and has the form: $y = \mu + \sum f_i\left(\boldsymbol{x}^i\right) + \sum f_{ij}\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right)$, 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. 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 7 (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 7). From Figure 7 (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 7 (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.

3.5.3 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 [68] to convert them to relatively dense numerical values and leverage affinity propagation [27] 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 7 (c) presents the feature interpretation of one job prediction from the Venus cluster in SenseTime [42]. 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.

Table 2: Summary of traces in large-scale simulations.

Trace	Source	#GPUs	#Jobs	Avg. Duration
Saturn [42]	SenseTime (Sep. 2020)	2,080	101,254	13,006s
Venus [42]	SenseTime (Sep. 2020)	1,080	23,859	5,419s
Philly [48]	Microsoft (Oct. 2017)	864	12,389	25,533s

3.6 System Optimizer

3.6.1 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 optimize the performance of interpretable models, we adopt the Pool Adjacent Violators (PAV) [8] algorithm to pose a monotonic constraint [41] on the learned feature shape function based on the model interpretability.

3.6.2 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 EVALUATION

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

4.1 Experimental Setup

Implementation. We implement Lucid with approximate 4700 lines of Python code. It leverages the gRPC [1] 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 §4.2. All experiment results without explicit comments are derived from the simulation. Besides, we implement Lucid interpretable models based on Primo [41]. For experiment workloads, we implement all models listed in Table 1 with Pytorch [71].

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.

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 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 [95, 98] and use a higher utilization trace for evaluation, as shown in Figure 12 (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 1) and vice versa. We apply hierarchical sampling to randomly assign each workload a job type derived from Table 1.

Table 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**

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 [89], and Kubernetes [15]). 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)* [42]: 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.

(4) *Horus* [100]: a packing-enabled and data-driven policy that predicts job resource usage through model analysis. It is intrusive as it obtains ONNX [7] graph representation through user-code intrusion and relies on a black-box ML model.

(5) *Tiresias* [31]: 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 [76] and discuss its impact on model quality in §4.7. We do not evaluate its performance in large-scale traces (§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.

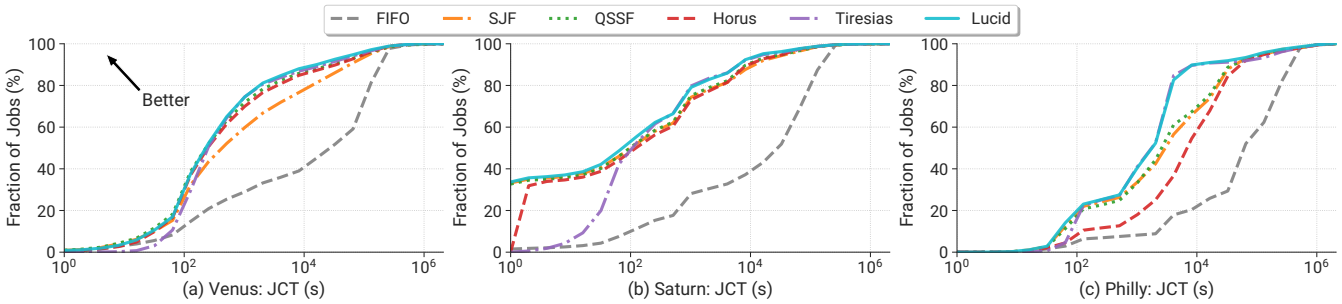

Figure 8: CDF of JCT using different scheduling approaches across three clusters: Venus, Saturn and Philly.

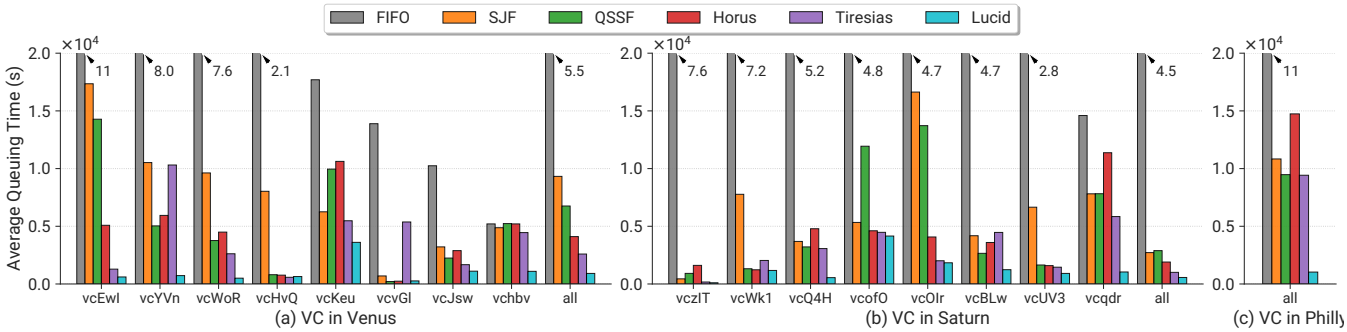

Figure 9: Average job queuing delay using different scheduling approaches across each VC, where *all* indicates the whole cluster.

Table 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**

4.2 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 [67]. 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 3). Lucid successfully improves the average JCT by 2.3× on the continuous trace and makespan by 1.4× on the static trace.

Table 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.

4.3 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 2).

Overall Performance. Figure 8 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, Lucid improves the average JCT by up to 1.3× compared with Tiresias, saving 2.2 hours for DL training jobs on average.

Figure 9 presents the VC-level analysis of average job queuing delay across three clusters. We select the top-8 VCs with the highest

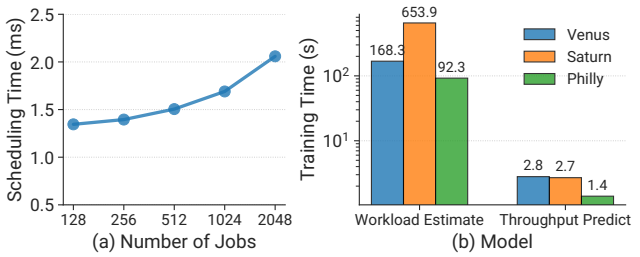

Figure 10: Scalability Analysis. (a) Scheduling latency (unit: ms) under various numbers of jobs. (b) Model training time (unit: s) across three clusters (y-axis in log scale).

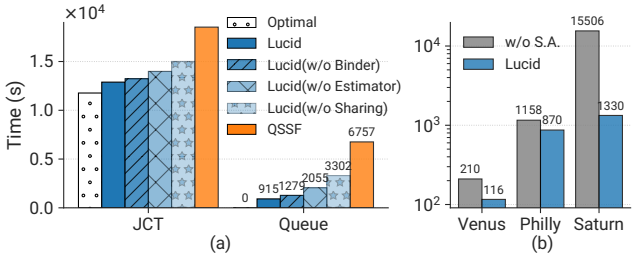

Figure 11: Ablation Study. Effect analysis of (a) binder and estimator; (b) space-aware profiling (S.A.), y-axis in log scale.

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 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 unfairness. Table 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 §2.2, 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 5. Lucid obviously outperforms Tiresias for both large and small jobs, which demonstrates large jobs will not experience starvation in Lucid scheduling.

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 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 10 (a), we measure the scheduling decision latency under more 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 [67] needs to take around 30 minutes to solve the linear programming problem [66]. Shockwave [108] and Muri [107] 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 retraining 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 10 (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.

4.5 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 §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 Figure 11 (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 11 (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). Moreover, we

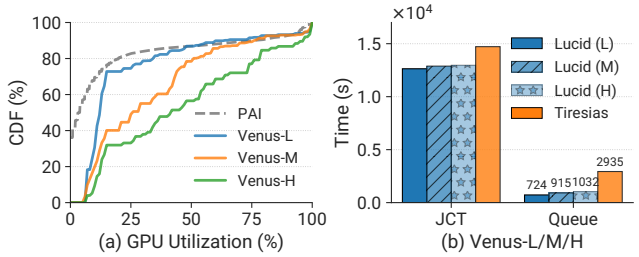

Figure 12: Sensitivity Analysis. (a) GPU utilization distributions of Alibaba PAI cluster [95, 98] and generated Venus traces with Low/Median/High utilization. (b) Lucid scheduling performance under various workload distributions.

Table 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

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 11. 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* (§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 11 (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 [30, 31, 62].

Sensitivity Analysis of Workload Distribution. One major concern of Lucid is whether it only applies to low cluster-wide GPU utilization scenarios. Figure 12 (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 (§4.3); Venus-L is designed to mimic the Alibaba cluster utilization scenario; Venus-H represents a high GPU utilization trace. As shown in Figure 12 (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 6 shows the scheduling performance under different T_{prof} settings (100s~600s) in Venus.

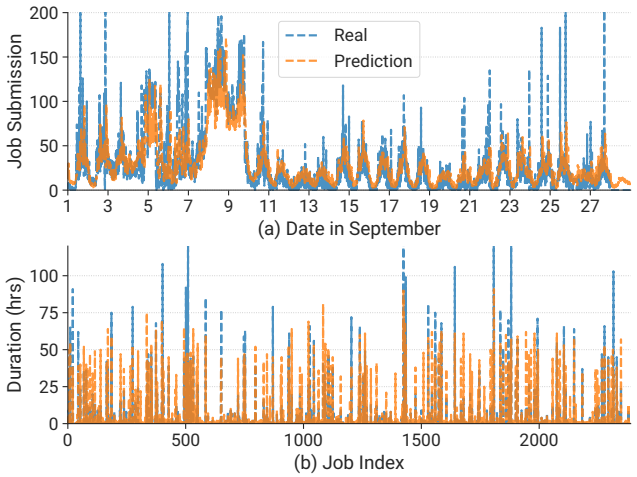

Figure 13: Prediction Visualization. (a) *Throughput Predict Model* for job submission prediction in Saturn. (b) *Workload Estimate Model* for job duration estimation in Venus.

Table 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**

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. 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.

4.6 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 (§3.5).

Model Performance. Figure 13 (a) presents cluster-wide job throughput prediction on Saturn September. We observe that our prediction

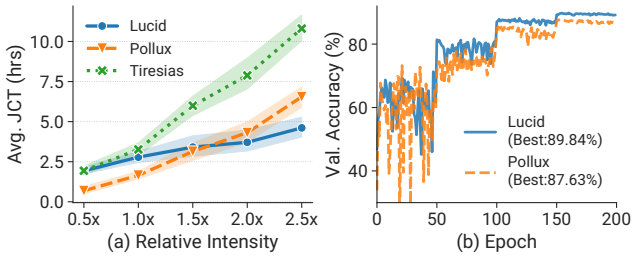

Figure 14: Comparison with Pollux. (a) Average JCT under various workload intensities. (b) Validation accuracy of an EfficientNet job with (Pollux) or without adaptive training.

can precisely reflect the actual trend with small estimation errors, which laid the foundation for dynamic system scaling and tuning. Figure 13 (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 [81]. We provide comprehensive evaluations of Lucid models with a series of popular ML algorithms: Random Forest (RF) [13], LightGBM [50], XGBoost [18] and DNN [56]. We use the default hyperparameters for baseline algorithms. Table 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.

4.7 Comparison with Elastic Scheduler

We further compare Lucid with the state-of-the-art elastic scheduler Pollux [76] 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 14 (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 14 (b). Pollux induces over 2% accuracy decrease in EfficientNet training which is often unacceptable in practice (**G3**) [108].

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 [30, 31, 62] 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 11). 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 [31]. According to our evaluation in Venus, preemption causes an additional 13% queuing overhead.

5 RELATED WORKS

DL Job Schedulers. Schedulers tailored for DL training workloads have been actively researched in recent years [11, 31, 42, 73, 76, 97, 98, 100] and many of them adopt job packing to improve resource utilization. Gandiva [97] leverages online-profiling to introspectively determine whether to co-locate jobs on an accelerator. AntMan [98] enables more fine-grained GPU sharing with dynamic scaling techniques. Salus [103] implements two primitives *fast job switching* and *memory sharing* for more efficient GPU sharing. Horus [100] converts user models into ONNX [7] 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 [67] and Gandiva$_{fair}$ [17] focus on leveraging the heterogeneity of GPU generations to improve resource utilization. CODA [106] designs a feedback-based adaptive CPU allocation algorithm for DL training jobs. Similarly, Synergy [65] allocates CPU and memory resources according to the workload sensitivity to these resources. Muri [107] 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 [15, 39, 89] 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 [21, 22, 47, 49], or job structure knowledge [12, 26, 46, 90, 91]. For more general cases, some schedulers [20, 70, 87] make the prediction from the history of relevant jobs. In DL clusters, Helios [42] characterizes SenseTime workloads and finds that using a LightGBM [50] model to predict job duration can improve scheduling performance. MLaaS [95] 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 [63], DeepAid [34] and Lemna [33] design toolkits to improve system transparency by interpreting black-box ML models. Furthermore, Unicorn [45] adopts causal inference to find effective repairs. In recent resource management research, Sinan [105] employs LIME [80] to identify important features of its hybrid model and Sage [29] dedicates to performance degradation reasoning of microservice. In contrast to them, Lucid adopts Primo [41] framework which directly builds interpretable models instead of putting effort to understand the black-box process.

6 CONCLUSION

In this paper, 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.

In the future, we plan to improve our work in two directions. (1) Supporting more scheduling objectives like fairness [62, 99, 108] and SLO-guarantee [30] to further improve user experience. (2) Adding 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).

ACKNOWLEDGMENTS

We sincerely thank our shepherd, Thaleia Dimitra Doudali, and the anonymous reviewers for their valuable comments on this paper. This study is supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, Shanghai AI Laboratory, as well as cash and in-kind contributions from the industry partner(s).

A ARTIFACT APPENDIX

A.1 Abstract

This artifact appendix describes how to reproduce main results in our Lucid paper. In our public repository, we provide the source code, related dataset and the instructions to perform artifact evaluation. Please refer to the `README.md` file for more details.

A.2 Artifact Check-List (Meta-information)

- **Program:** Python; Shell Script.
- **Model:** Lucid Model: Decision Tree and GA^{2}M; Workload Model: Listed in Table 1.
- **Data set:** Job Traces: SenseTime Helios and Microsoft Philly; Workload Dataset: Listed in Table 1.
- **Run-time environment:** Ubuntu 20.04 with Python 3.9, Pytorch 1.10, CUDA 11.3 and cuDNN 8.
- **Hardware:** Each server is equipped with dual-sockets Intel Xeon Gold 6326 CPUs (64 threads, 256GB memory) and 8 NVIDIA RTX 3090 GPUs (24GB memory).
- **Execution:** Refer to `README.md` file.
- **Metrics:** Average job completion time; Average job queuing delay.
- **Output:** Performance results and figures of baselines and Lucid.
- **Experiments:** Reproduction of cluster `Venus` results.
- **How much disk space required (approximately)?:** 10GB.
- **How much time is needed to prepare workflow (approximately)?:** 1 hour.
- **How much time is needed to complete experiments (approximately)?:** 2 hours.
- **Publicly available?:** Yes.
- **Code licenses?:** S-Lab License.
- **Data licenses?:** Creative Commons Attribution 4.0.

A.3 Description

A.3.1 How to Access. To reproduce the main results of this work, we provide code and detailed documentation of Lucid in the artifact repository as below [2].

Artifact Link

GitHub: https://github.com/S-Lab-System-Group/Lucid
DOI: https://doi.org/10.5281/zenodo.7275326

A.4 Installation

Please refer to `README.md` file for detailed instructions.

```
git clone https://github.com/S-Lab-System-Group/Lucid.git
conda create -n lucid python=3.9
conda activate lucid
cd Lucid/simulation
pip install -r requirements.txt
```

A.5 Evaluation and Expected Results

Scheduling Performance. The results generated in experiments of the artifact can be matched with the results in Table 4, Table 5, Figure 8 and Figure 9.

Model Evaluation. The interpretable model results can be matched with Table 7, Figure 7 and Figure 13.

REFERENCES

[1] 2023. gRPC: An RPC library and framework. https://grpc.io/.
[2] 2023. *Lucid: Lucid Artifact.* https://doi.org/10.5281/zenodo.7275326
[3] 2023. NVIDIA Data Center GPU Manager. https://developer.nvidia.com/dcgm.
[4] 2023. NVIDIA Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.
[5] 2023. NVIDIA Multi-Process Service. https://docs.nvidia.com/deploy/mps/index.html.
[6] 2023. NVIDIA-smi. https://developer.nvidia.com/nvidia-system-management-interface.
[7] 2023. ONNX: Open Neural Network Exchange. https://github.com/onnx/onnx.
[8] Miriam Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and Edward Silverman. 1955. An Empirical Distribution Function for Sampling with Incomplete Information. *The Annals of Mathematical Statistics* 26 (1955).
[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations (ICLR '15).*
[10] Yixin Bao, Yanghua Peng, and Chuan Wu. 2019. Deep Learning-based Job Placement in Distributed Machine Learning Clusters. In *IEEE Conference on Computer Communications (INFOCOM '19).*
[11] Zhengda Bian, Shenggui Li, Wei Wang, and Yang You. 2021. 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).*
[12] Eric Boutin, Jaliya Ekanayake, Wei Lin, Bing Shi, Jingren Zhou, Zhengping Qian, Ming Wu, and Lidong Zhou. 2014. Apollo: Scalable and Coordinated Scheduling for Cloud-Scale Computing. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14).*
[13] Leo Breiman. 2001. Random Forests. *Machine learning* (2001), 5–32.
[14] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. 1984. *Classification and regression trees.* Wadsworth.
[15] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade. *Queue* 14 (2016), 70–93.
[16] 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. 2015. ShapeNet: An Information-Rich 3D Model Repository. *CoRR* abs/1512.03012 (2015).
[17] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, and Srinidhi Viswanatha. 2020. Balancing Efficiency and Fairness in Heterogeneous GPU Clusters for Deep Learning. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys '20).*
[18] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16).*
[19] Zhaoyun Chen, Lei Luo, Wei Quan, Mei Wen, and Chunyuan Zhang. 2019. Poster Abstract: Deep Learning Workloads Scheduling with Reinforcement Learning on GPU Clusters. In *IEEE Conference on Computer Communications Workshops (INFOCOM '19).*
[20] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. 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).*
[21] Carlo Curino, Djellel E. Difallah, Chris Douglas, Subru Krishnan, Raghu Ramakrishnan, and Sriram Rao. 2014. Reservation-Based Scheduling: If You're Late Don't Blame Us!. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '14).*
[22] Pamela Delgado, Florin Dinu, Anne-Marie Kermarrec, and Willy Zwaenepoel. 2015. Hawk: Hybrid Datacenter Scheduling. In *2015 USENIX Annual Technical Conference (USENIX ATC '15).*
[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition.*
[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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).*
[25] Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. 2016. Multi30K: Multilingual English-German Image Descriptions. In *Proceedings of the 5th Workshop on Vision and Language.*
[26] Andrew D. Ferguson, Peter Bodik, Srikanth Kandula, Eric Boutin, and Rodrigo Fonseca. 2012. Jockey: Guaranteed Job Latency in Data Parallel Clusters. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12).*
[27] Brendan J. Frey and Delbert Dueck. 2007. Clustering by Passing Messages Between Data Points. *Science* (2007).
[28] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning (ICML '18).*
[29] Yu Gan, Mingyu Liang, Sundar Dev, David Lo, and Christina Delimitrou. 2021. 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).*
[30] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2021. Chronus: A Novel Deadline-aware Scheduler for Deep Learning Training Jobs. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '21).*
[31] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU Cluster Manager for Distributed Deep Learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19).*
[32] Rong Gu, Yuquan Chen, Shuai Liu, Haipeng Dai, Guihai Chen, Kai Zhang, Yang Che, and Yihua Huang. 2021. Liquid: Intelligent Resource Estimation and Network-Efficient Scheduling for Deep Learning Jobs on Distributed GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2021).
[33] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. 2018. LEMNA: Explaining Deep Learning Based Security Applications. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18).*
[34] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. 2021. 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).*
[35] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S. Gunawi. 2020. LinnOS: Predictability on Unpredictable Flash Storage with a Light Neural Network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20).*
[36] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* 5 (2015), 1–19.
[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16).*
[38] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17).*
[39] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI '11).*
[40] 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. 2019. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV '19).*
[41] Qinghao Hu, Harsha Nori, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2022. Primo: Practical Learning-Augmented Systems with Interpretable Models. In *2022 USENIX Annual Technical Conference (USENIX ATC '22).*
[42] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. 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).*
[43] Jensen Huang. 2023. NVIDIA GTC 2022 KEYNOTE. https://www.nvidia.com/gtc/keynote/.
[44] Changho Hwang, Taehyun Kim, Sunghyun Kim, Jinwoo Shin, and KyoungSoo Park. 2021. Elastic Resource Sharing for Distributed Deep Learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21).*
[45] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. 2022. Unicorn: reasoning about configurable system performance through the lens of causality. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22).*
[46] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys '07).*
[47] Virajith Jalaparti, Peter Bodik, Ishai Menache, Sriram Rao, Konstantin Makarychev, and Matthew Caesar. 2015. 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).*
[48] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC '19).*

[49] 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. 2016. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

[50] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems (NeurIPS '17)*.

[51] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2017. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *International Conference on Learning Representations (ICLR '17)*.

[52] Sejin Kim and Yoonhee Kim. 2020. 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)*.

[53] Alex Krizhevsky. 2023. The CIFAR10 Dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[54] H.T. Kung, Bradley McDanel, and Sai Qian Zhang. 2019. 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)*.

[55] Mikel Landajuela, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, Jacob F Pettit, and Daniel Faissol. 2021. Discovering symbolic policies with deep reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML '21)*.

[56] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521 (2015), 436–444.

[57] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. 2022. Aryl: An Elastic Cluster Scheduler for Deep Learning. *CoRR* abs/2202.07896 (2022).

[58] Rui Li, Yufan Xu, Aravind Sukumaran-Rajam, Atanas Rountev, and P. Sadayappan. 2021. 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)*.

[59] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. 2013. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '13)*.

[60] Yunteng Luan, Xukun Chen, Hanyu Zhao, Zhi Yang, and Yafei Dai. 2019. SCHED^2: Scheduling Deep Learning Training via Deep Reinforcement Learning. In *2019 IEEE Global Communications Conference (GLOBECOM '19)*.

[61] Qinyi Luo, Jiaao He, Youwei Zhuo, and Xuehai Qian. 2020. 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)*.

[62] Kshiteej Mahajan, Arjun Balasubramanian, Arjun Singhvi, Shivaram Venkataraman, Aditya Akella, Amar Phanishayee, and Shuchi Chawla. 2020. Themis: Fair and Efficient GPU Cluster Scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*.

[63] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. 2020. Interpreting Deep Learning-Based Networking Systems. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '20)*.

[64] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations (ICLR '17)*.

[65] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking Beyond GPUs for DNN Scheduling on Multi-Tenant Clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*.

[66] Deepak Narayanan, Fiodar Kazhamiaka, Firas Abuzaid, Peter Kraft, Akshay Agrawal, Srikanth Kandula, Stephen Boyd, and Matei Zaharia. 2021. Solving Large-Scale Granular Resource Allocation Problems Efficiently with POP. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*.

[67] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[68] Gonzalo Navarro. 2001. A Guided Tour to Approximate String Matching. *Comput. Surveys* 33 (2001), 31–88.

[69] Harsha Nori, Rich Caruana, Zhiqi Bu, Judy Hanwen Shen, and Janardhan Kulkarni. 2021. Accuracy, Interpretability, and Differential Privacy via Explainable Boosting. In *Proceedings of the 38th International Conference on Machine Learning (ICML '21)*.

[70] Jun Woo Park, Alexey Tumanov, Angela Jiang, Michael A. Kozuch, and Gregory R. Ganger. 2018. 3Sigma: Distribution-Based Cluster Scheduling for Runtime Uncertainty. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*.

[71] 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. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS '19)*.

[72] Xuan Peng, Xuanhua Shi, Hulin Dai, Hai Jin, Weiliang Ma, Qian Xiong, Fan Yang, and Xuehai Qian. 2020. 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)*.

[73] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*.

[74] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. 2021. DL2: A Deep Learning-Driven Scheduler for Deep Learning Clusters. *IEEE Transactions on Parallel and Distributed Systems* 32 (2021), 1947–1960.

[75] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. 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)*.

[76] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R. Ganger, and Eric P. Xing. 2021. Pollux: Co-adaptive Cluster Scheduling for Goodput-Optimized Deep Learning. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*.

[77] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In *International Conference on Learning Representations (ICLR '16)*.

[78] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *CoRR* abs/1606.05250 (2016).

[79] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '12)*.

[80] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "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)*.

[81] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* (2019), 206–215.

[82] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '18)*.

[83] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

[84] 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. 2022. Singularity: Planet-Scale, Preemptive and Elastic Scheduling of AI Workloads. *CoRR* abs/2202.07848 (2022).

[85] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR '15)*.

[86] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML '19)*.

[87] Alexey Tumanov, Angela Jiang, Jun Woo Park, Michael A. Kozuch, and Gregory R. Ganger. 2016. *JamaisVu: Robust Scheduling with Auto-Estimated Job Runtimes*. Technical Report. Carnegie Mellon University.

[88] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems (NeurIPS '17)*.

[89] 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. 2013. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing (SoCC '13)*.

[90] Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, and Ion Stoica. 2016. Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI '16)*.

[91] Abhishek Verma, Ludmila Cherkasova, and Roy H. Campbell. 2011. ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments. In *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC '11)*.

[92] Haoyu Wang, Zetian Liu, and Haiying Shen. 2020. Job scheduling for large-scale machine learning clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '20)*.

[93] Luping Wang, Qizhen Weng, Wei Wang, Chen Chen, and Bo Li. 2020. 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)*.

[94] Shaoqi Wang, Oscar J Gonzalez, Xiaobo Zhou, Thomas Williams, Brian D Friedman, Martin Havemann, and Thomas Woo. 2020. 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)*.

[95] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. 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)*.

[96] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. 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)*.

[97] 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. 2018. Gandiva: Introspective Cluster Scheduling for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*.

[98] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic Scaling on GPU Clusters for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*.

[99] Zhisheng Ye, Peng Sun, Wei Gao, Tianwei Zhang, Xiaolin Wang, Shengen Yan, and Yingwei Luo. 2021. ASTRAEA: A Fair Deep Learning Scheduler for Multi-tenant GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[100] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. 2022. Horus: Interference-Aware and Prediction-Based Scheduling in Deep Learning Systems. *IEEE Transactions on Parallel and Distributed Systems* 33 (2022), 88–100.

[101] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*.

[102] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. 2016. LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop. *CoRR* abs/1506.03365 (2016).

[103] Peifeng Yu and Mosharaf Chowdhury. 2020. Fine-Grained GPU Sharing Primitives for Deep Learning Applications. In *Proceedings of Machine Learning and Systems (MLSys '20)*.

[104] Peifeng Yu, Jiachen Liu, and Mosharaf Chowdhury. 2021. Fluid: Resource-aware Hyperparameter Tuning Engine. In *Proceedings of Machine Learning and Systems (MLSys '21)*.

[105] Yanqi Zhang, Weizhe Hua, Zhuangzhuang Zhou, G. Edward Suh, and Christina Delimitrou. 2021. 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)*.

[106] Han Zhao, Weihao Cui, Quan Chen, Jingwen Leng, Kai Yu, Deze Zeng, Chao Li, and Minyi Guo. 2020. 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)*.

[107] Yihao Zhao, Yuanqiang Liu, Yanghua Peng, Yibo Zhu, Xuanzhe Liu, and Xin Jin. 2022. Multi-resource interleaving for deep learning training. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '22)*.

[108] Pengfei Zheng, Rui Pan, Tarannum Khan, Shivaram Venkataraman, and Aditya Akella. 2023. Shockwave: Fair and Efficient Cluster Scheduling for Dynamic Adaptation in Machine Learning. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI '23)*.

Received 2022-07-07; accepted 2022-09-22