Castor: Optimizing Deep Learning Job Scheduling in Multi-Tenant GPU Clusters via Intelligent Colocation

Yizhou Luo, Jiaxin Lai, Shaohuai Shi, Chen Chen, Shuhan Qi, Jiajia Zhang, Qiang Wang*

Abstract—Deep learning (DL) has achieved significant success across a wide range of domains, prompting the widespread deployment of GPU clusters equipped with specialized accelerators to support high-performance training workloads. To minimize operational costs while maximizing resource utilization, efficient job scheduling in these clusters is essential. Although recent schedulers have improved cluster efficiency through periodic reallocation or selection of GPU resources, they still face challenges such as preemption and migration overheads, along with the risk of degrading model accuracy. Despite these limitations, the potential of GPU sharing remains largely underexplored. Few existing studies have systematically examined GPU sharing as a strategy to enhance resource utilization and reduce job queuing delays in multi-tenant DL clusters. Motivated by these insights, we propose a job scheduling model that enables multiple jobs to share the same set of GPUs without modifying their original training configurations. We introduce Castor, a simple yet efficient scheduling system, to achieve intelligent GPU colocation for multiple DL jobs. Castor intelligently selects job pairs for GPU sharing and determines runtime parameters (sub-batch size and scheduling time point) to optimize overall system performance while preserving the accuracy of DL convergence through gradient accumulation. Through a combination of physical DL workloads and trace-driven simulations across various configurations, we demonstrate that Castor reduces average job completion time by 26–52% compared to state-of-the-art preemptive DL schedulers, despite operating under a preemption-free policy. Furthermore, Castor effectively identifies optimal resource-sharing configurations, outperforming the baseline first-fit sharing policy (SJF-FFS) by up to 20% on large-scale workload traces.

Index Terms—Distributed Deep Learning, Job Scheduling, Resource Sharing

1 Introduction

The widespread adoption of Deep Neural Network (DNN) [1]–[3] has surged across both academic research and industrial applications, owing to its remarkable effectiveness in domains like image recognition and language understanding. However, as datasets expand and models become more complex—particularly with the rapid advancements of Large Language Models (LLMs) [4]—the process of model training becomes increasingly time-consuming and resource-intensive. To mitigate this challenge, distributed training techniques [5] have been extensively employed, where the learning workload is partitioned across multiple computing nodes, enabling parallel execution and significantly reducing overall training time.

In modern data centers, where computational infrastructure is shared among multiple tenants, numerous deep learning training jobs often run concurrently. Without careful coordination, contention for limited resources can cause significant slowdowns in training performance [6]. In this dynamic setting—where the volume of incoming jobs continues to grow—it is critical to employ adaptive resource management and intelligent orchestration to sustain high system throughput. Although various conventional schedulers have been developed to manage generic computing workloads [7]–[11], they are not tailored for distributed deep learning jobs and fail to exploit unique characteristics such as iterative execution and convergence behavior to accelerate training and optimize resource usage.

Several existing systems for managing and orchestrating deep learning workloads [12]–[17] adopt exclusive GPU allocation and preemption-based policies to improve system throughput and reduce training latency. Among them, the heuristic-driven scheduler Tiresias [16] demonstrates that the shortest-remaining-service-first (SRSF) approach performs optimally when job durations are known in advance. Despite this, lightweight training jobs often face delays, waiting for GPU resources held by long-running jobs that dominate cluster occupancy. Pollux [18], a state-of-the-art solution in this space, aims to boost overall cluster efficiency by dynamically reallocating resources while maintaining fairness and adaptively optimizing resource consumption efficiency per workload. However, Pollux also takes control over GPU selection and hyperparameter tuning, which—while beneficial in some cases—has been shown to degrade final model quality [19]. Generally, preemption-based and exclusive-execution approaches are vulnerable to head-of-line (HOL) blocking, where large jobs delay the execution of smaller ones, leading to increased job completion times (JCT). This can result in starvation for low-resource, short-duration jobs, while larger jobs may incur

- *Yizhou Luo, Jiaxin Lai, Shaohuai Shi, Shuhan Qi, Jiajia Zhang, and Qiang Wang are with Harbin Institute of Technology, Shenzhen 518055, China.*
- *Chen Chen is with the John Hopcroft Center for Computer Science, Shanghai Jiao Tong University.*
- *Qiang Wang is the corresponding author.*
  E-mail: qiang.wang@hit.edu.cn

significant overhead from frequent migration and context switching.

Recent advances in job scheduling, such as Gandiva [20], Zico [21], Salus [22], and Lucid [23], highlight increasing efforts toward shared usage of resources, especially GPUs and interconnect bandwidth. This approach aims to boost overall system efficiency while mitigating queueing delays and preventing job starvation. Gandiva [20] pioneered a time-slicing mechanism for GPUs, leveraging predicted training patterns of distributed jobs to guide GPU multiplexing. However, its strategy was conservative, limiting GPU multiplexing to jobs that utilize only a single GPU. Alternatively, Lucid [23] introduced a lazy packing heuristic to minimize contention among colocated jobs. Nevertheless, its effectiveness was constrained by a fixed batch size assumption, which restricted scheduling flexibility and job compatibility. To overcome the limitations imposed by the fixed batch size assumption, gradient accumulation has been widely adopted in modern deep learning frameworks [24]. The core idea is to aggregate gradients over multiple smaller micro-batches before performing a parameter update. This approach is particularly beneficial for training extremely large models [25], [26], where GPU memory constraints restrict each node to processing only one micro-batch at a time. From an optimization standpoint, the technique is mathematically equivalent to using a larger mini-batch size, as it effectively averages gradients over the same number of training examples prior to weight updates.

Building on the above insights, we propose a scheduling system, called Castor, that supports the concurrent execution of multiple training jobs on shared GPU resources. This article extends our previous conference paper [27], which introduced a scheduling algorithm for judicious GPU sharing among multiple deep learning jobs within a cluster using a temporal paradigm. Similar to [27], Castor comprehensively investigates the advantages of GPU-level colocation to improve overall system efficiency. To mitigate memory constraints and maintain convergence stability, Castor integrates gradient accumulation. Besides, Castor advances the job scheduling method in [27] in three folds. First, we propose an IR prediction model that requires only one-time profiling per job. The accuracy and practical applicability of this model are thoroughly validated through extensive experiments (see Section 9.5). Second, we replace the default temporal sharing mechanism with NVIDIA's Multi-Process Service (MPS) to incorporate spatial GPU sharing. This system-level enhancement is shown to be more effective, as evidenced by the ablation studies in Section 9.5. Third, in the experiments, we introduce a new metric, "sharing benefit", to statically quantify the advantages of intelligent GPU sharing. We also include a more comprehensive evaluation to highlight Castor's superior performance, as detailed in Section 9. The main contributions of this work are outlined as follows:

- We introduce a novel DDL job scheduling model that enables multiple jobs to fully or partially share the same set of GPUs while ensuring model convergence through gradient accumulation. Our model emphasizes intelligent GPU resource sharing to mitigate starvation issues and alleviate GPU memory constraints, addressing limitations in prior methods that risk accuracy degradation.
- We propose Castor, a simple yet effective scheduling system tailored to this problem. Initially, we derive the optimal scheduling strategy for a job pair (an ongoing job and a newly arrived one) to determine both GPU sharing feasibility and launch timing. Subsequently, a greedy strategy is employed to decide batch sizes and GPU allocation, aiming to minimize interference with existing jobs while reducing queuing delays.
- We introduce a practical and interpretable prediction model as a predictor to estimate the value of the interference ratio for unprofiled cases, overcoming the challenges of speculation on a large scale due to the exponential growth in combinations. We further demonstrate the model's accuracy and applicability through extensive experiments.
- Through both physical and simulated evaluations of two widely adopted real-world DL workloads, Venus [28] and Philly [29], we assess Castor across various scales of job traces. When compared to state-of-the-art preemptive DL schedulers such as Pollux [18] and non-preemptive schedulers that allow GPU sharing, like Lucid [23], Castor reduces the average job completion time by 26-52%. Furthermore, in comparison to the first-fit GPU sharing approach for newly arriving jobs, Castor effectively avoids sharing decisions that could potentially degrade overall performance, outperforming this method by up to 20%.

2 Related Work

Scheduling DL training workloads has attracted growing attention in recent years. Prior research in this area primarily aims to maximize resource utilization and optimize resource allocation in multi-tenant GPU environments. Broadly, existing approaches can be classified into two categories: preemptive and exclusive schedulers, and non-preemptive schedulers.

Preemptive and Exclusive Schedulers. These schedulers are designed with the ability to interrupt or preempt active jobs, reallocating dedicated resources to those with higher priority. Such exclusivity ensures that once a job acquires resources, they remain unavailable to others during its execution, thereby promoting predictable performance and reducing interference among running jobs. Early systems like Optimus [15] and Cynthia [30] adopted training time prediction based on simplified assumptions of convergence behavior. Tiresias [16] tackled the issue of severe resource contention by introducing adaptive scheduling mechanisms combined with effective job migration strategies. On top of that, Harmony [31] and Spear [32] employed deep reinforcement learning techniques to minimize metrics like average job completion time and overall makespan, demonstrating improved scheduling efficiency. More recent approaches, such as AFS [33] and Heet [34], have employed resource fragmentation to heuristically minimize an overall metric. Another line of research in DDL job scheduling algorithms relies on theoretical formulation and optimization, treating DDL job scheduling as constrained optimization problems.

The recent state-of-the-art Pollux [18] and Sia [35] dynamically reallocates resources to enhance cluster-wide throughput while ensuring fairness and continually optimizing each DL job to maximize resource utilization. However, these methods cannot guarantee no accuracy degradation for all models. Moreover, they may encounter performance degradation due to migration [16], [18], [35] and GPU underutilization [36].

Non-preemptive Schedulers. Early non-preemptive schedulers predominantly relied on heuristic algorithms based on job characterization and hardware performance modeling. In recent studies, attention has shifted towards resource sharing, encompassing GPU and network resources, which hold significant potential for improving computing resource utilization and alleviating starvation. Gandiva [20] introduced GPU time-slicing and job scheduling by predicting DDL training job characteristics. However, it adopted a conservative approach, limiting GPU sharing to single-GPU jobs. Wang et al. [37] explored contention under various all-reduce architectures, including RAR. Nonetheless, they relied on a system-dependent online-fitting model for execution time prediction without explicitly formulating any scheduling optimization problem. Zico [21] focused on system-wide memory consumption for concurrent training and devised a feasible memory management solution to ensure that concurrent jobs do not exceed the allocated memory budget. Yu et al. [38], [39] addressed network resource sharing in multiple RAR-based DDL job training, alleviating communication contention overhead. Lucid [23] employed an indolent packing strategy to mitigate interference. However, few of these approaches offer a general and flexible solution for sharing GPUs among DL jobs.

Furthermore, as workload heterogeneity increases, the complexity and computational cost of scheduling rise exponentially. To address this challenge, recent studies have increasingly adopted model-based prediction methods as an alternative to traditional analytical or rule-based strategies.

Prediction model. In order to predict job performance in online services and conduct trace-driven simulations for offline training, it is essential to derive relevant information, such as iteration time, for effective performance modeling. However, collecting data for every possible scenario is impractical due to the prohibitive time costs involved. Consequently, a conventional approach is to profile only a select number of cases, while the remaining cases are estimated through mathematical fitting. But, this method still suffers from severe time costs when the workload heterogeneity becomes significant. Predictive modeling has been widely adopted as a practical solution to address this inherent intractability. Heet [34] integrated an SVD-based PQ-decomposition model to reduce the complexity of profiling cases from $O(M \times N)$ to $O(\min\{M, N\})$, where M and N represent the maximum number of GPUs and host types, respectively. Lucid [23] also employed a model based on the GA^{2}M algorithm to predict cluster throughput, in addition to a similar model for job duration prediction.

To address the challenges identified in the existing studies, we aim to explore the new opportunities presented by gradient accumulation and GPU sharing to enhance overall performance. Additionally, we propose a prediction model based on a regression decision tree to effectively mitigate the significant profiling overhead introduced by the surging complexity of heterogeneous workloads.

3 System Modeling and Problem Formulation

In this section, we first present our model of DL job training time, grounded in both empirical observations and theoretical formulation. Building upon this model, we introduce a representative resource-sharing scenario involving two running DL jobs. Finally, we generalize this scenario into a comprehensive scheduling problem and formally define the optimization objective we seek to address.

For ease of reference, we summarize some frequently used notations throughout this paper in Table 1.

Table 1
Frequently used notations

Name	Descriptions
$\mathcal{S}/\mathcal{N}$	Set of servers/GPUs in the cluster
S_i	the i-th server of the cluster
$g_{i,j}$	the j-th GPU on the i-th server
$\mathcal{J}$	The job set
G_j	# of GPUs requested by job j
J_k	the k-th job
$\mathcal{G}(J_k)$	the set of GPUs used by J_k
$\mathcal{S}(J_k)$	the set of servers used by J_k
a_k	the arrival time of J_k
B_k	the mini-batch size per-GPU used by J_k
I_k	# of iterations that J_k needs to run
t_k	The execution time of one iteration of J_k
L_k	The total execution time of J_k running solely
T_k	The completion time of job j
E_k	the timestamp when J_k is finished
$\mathcal{F}$	The set of feasible scheduling solution
f_j^k	The schedule of job j in the k-th solution

We consider a multi-tenant GPU cluster comprising $|\mathcal{S}|$ servers equipped with $|\mathcal{N}|$ GPUs evenly distributed. These servers are interconnected with a network switch that has sufficient bandwidth. All GPUs within the cluster share the same specifications and theoretical peak performance. At the onset of a scheduling horizon $|\mathcal{T}|$ spanning time-slots, a set of DDL jobs $\mathcal{J}$ awaits scheduling for training over the duration of $|\mathcal{T}|$. Each job $J_k \in \mathcal{J}$ is characterized by the number of GPUs it requires, denoted as $\mathcal{G}(J_k)$, and the total number of training iterations I_k requested by its users.

3.1 DL Job Training Time Modeling

We begin by collecting and visualizing the training throughput variations of individual DL jobs under different training configurations. Representative examples are shown in Figure 2. Based on the patterns observed in these figures and the training process of DL jobs, we construct a mathematical model that captures both GPU computation time and the network communication overhead associated with the all-reduce operation.

3.1.1 Modeling GPU Computation

The DL model is trained using back-propagation. The computation time on GPU scales linearly with the per-GPU batch size B, which can be calculated as follows.

$$t_{comp}(B) = \alpha_{comp} + \beta_{comp} \times B \tag{1}$$

3.1.2 Modeling Network Communication

The gradient aggregation overhead depends on the topology as well as the network communication algorithm. We simply define the communication part as follows.

$$t_{comm} = \alpha_{comm} + \beta_{comm} \times M \tag{2}$$

where M is the message size, and $\alpha_{comm}, \beta_{comm}$ are the all-reduce time model parameters that are not related to M [40]. It is important to note that the values of α_{comm} and β_{comm} are influenced by the algorithms used for the All-Reduce operation, which vary based on the number of processes and message sizes [40]–[42].

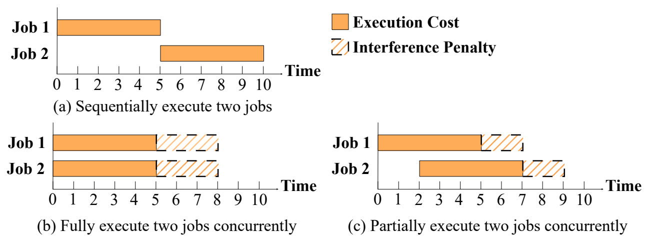


Figure 1. Three job schedules for two DL jobs.

3.1.3 Modeling Gradient Accumulation

Given that GPU memory constraints may limit the per-GPU batch size, some schedulers tackle this limitation through memory offloading [43] (which may introduce additional system overhead) or by adjusting batch sizes and other training hyper-parameters [18] (which may compromise model accuracy). As our model incorporates GPU sharing, the memory footprint frequently imposes constraints on feasibility. Thus, we focus on gradient accumulation, which can dynamically reduce the sub-batch size while preserving the original model accuracy as per the user's requested batch size. It is also easily implemented using popular DL frameworks. It is important to note that one can utilize gradient accumulation algorithms to manage the computational aspect, thereby reducing batch size to mitigate memory consumption. We subsequently define the overall iteration time as follows.

$$t_{iter}^{j} = (s-1) \times t_{comp}^{j}(\frac{B}{s}) + ((t_{comp}(\frac{B}{s}))^{\delta} + t_{comm}{}^{\delta})^{1/\delta} \tag{3}$$

where s represents the accumulation step required to attain the original batch size, and δ denotes the degree of overlap between GPU computation and all-reduce communication, as initially proposed in [18]. It is important to acknowledge that δ may vary when different batch sizes are applied.

3.2 Inteference Ratio (IR) for Colocation

Existing schedulers that facilitate GPU sharing, such as Gandiva [20], Gavel [17], and Lucid [23], often adopt conservative and limited approaches or require additional application information to generate schedules. In contrast, we apply a simple interference model to describe the overhead of GPU sharing. We illustrate three possible job schedules for two jobs sharing the same set of GPUs in Figure 1. Schedule (a) sequentially executes two DL jobs. Schedules (b) and (c) involve invoking two DL jobs simultaneously or with partial overlap, resulting in varying degrees of interference penalty. To optimize the average job completion time, one must balance the tradeoff between job queuing/waiting time (Job 2 waits for Job 1 to finish in (a)) and interference penalty (complete overlap of two jobs leads to severe penalty in (b)). In practice, the job iteration time under GPU sharing can be measured and modeled by equations (1) and (2), as they occupy partial GPU and network resources with similar trends. To simplify the model, if a new job shares GPUs occupied by an existing job (Job A and Job B), we multiply their job iteration time with an interference ratio (IR) as follows,

$$\hat{t}_A = t_A \xi_A \tag{4}$$
$$\hat{t}_B = t_B \xi_B, \tag{5}$$

where ξ_A and ξ_B denote the IRs of two jobs, reflecting the performance degradation resulting from GPU sharing. The solution to determining the optimal scheduling point under this scenario will be discussed in Section 5.1.

3.3 Scheduling Modeling

In this paper, we adopt the "gang-scheduling" discipline widely prevalent in practical large-scale GPU clusters [16], [39], [44]. Under gang scheduling, all workers (i.e., GPUs) of a DDL job must be allocated simultaneously. Furthermore, once a job commences its scheduled run, all allocated GPUs must remain dedicated to the job until its completion, with no allowances for preemption or migration. (It is worth noting that frequent job preemption and migration can significantly degrade performance [16]). Upon job completion, the occupied resources are simultaneously released. Differing from conventional GPU-exclusive scheduling policies, we permit GPUs to be occupied by multiple workers from various jobs concurrently. These workers can be allocated within a single server or across multiple servers, provided there exists a network path connecting them.

Assume $y_{jg}[\tau]$ denotes that the job j uses GPU g in the time slot t. Job j requires G_j number of GPUs.

$$\sum_{g \in \mathcal{G}} y_{jg}[\tau] = G_j, \tag{6}$$

To ensure that one GPU at most holds C jobs, we have

$$\sum_{j \in J[\tau]} y_{jg}[\tau] \le C, \forall j \in \mathcal{J}[t], \tau \in \mathcal{T}, g \in \mathcal{G} \tag{7}$$

In practice, we observe that interference degradation can be severe, rarely improving performance when more than two jobs share the same set of GPUs. Therefore, we set $C = 2$ in our context.

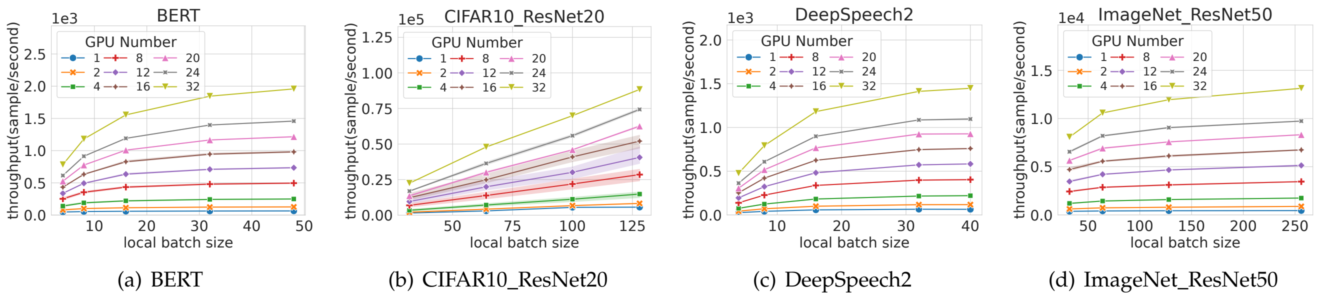


Figure 2. System throughput for four DL models in our experiments, as measured using a 4-server cluster each with 8 NVIDIA A6000 GPU. Each sub-figure shows the values of different resource and training batch size settings for each model.

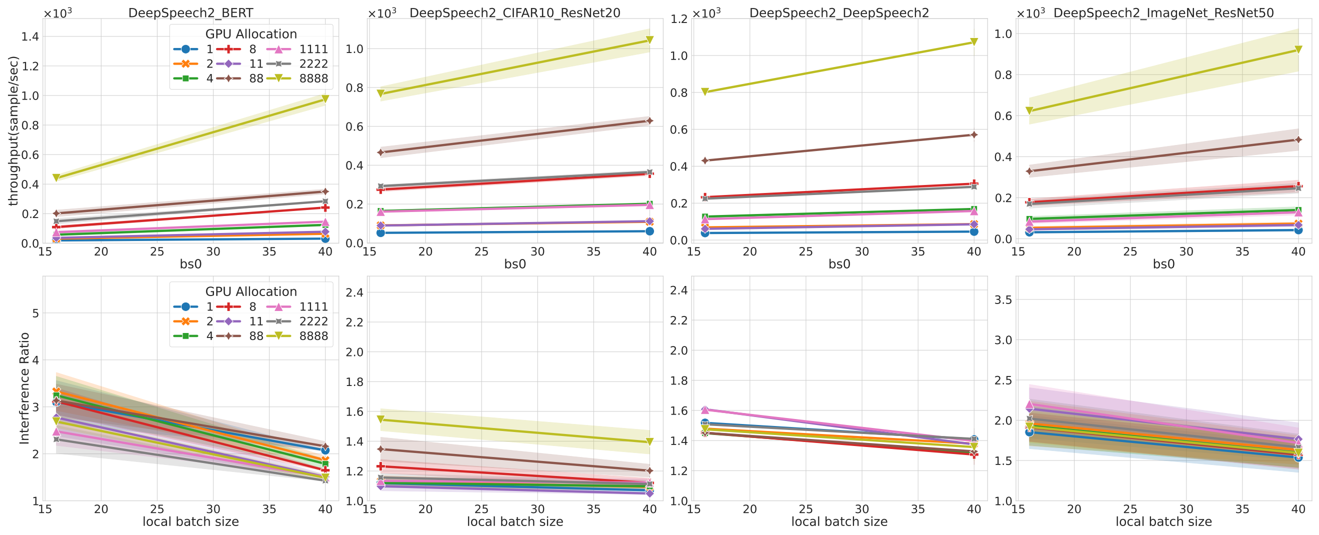


Figure 3. TOP: System throughput of difference DL models paired with DeepSpeech2 to share the same set of GPUs. BOTTOM: the interference ratio ξ for different DL models and resource and training settings.

Also, since we consider gang scheduling, we have

$$y_{jg}[\tau] = y_{gs}[\tau-1], \forall s \in S, j \in \mathcal{J}[\tau], a_j < \tau \leq T_j \quad (8)$$

$$y_{jg}[\tau] = 0, \forall g \in \mathcal{G}, j \notin \mathcal{J}[\tau], \tau \in \mathcal{T} \quad (9)$$

$$y_{jg}[\tau] \in \mathbb{Z}^+, \forall g \in \mathcal{G}, j \in \mathcal{J}[\tau], \tau \in \mathcal{T} \quad (10)$$

The completion time of job j can be calculated as

$$T_j = a_j + \arg\min_{\tau} \sum_{\tau \in \mathcal{T}} \frac{1}{t^j_{iter}} \geq I_k, \forall j \in \mathcal{J}[\tau], \tau \geq a_j \quad (11)$$

$$\phi_j[t] = \frac{B_k}{t^j_{iter}} \quad (12)$$

where $\phi_j[\tau]$ denotes the system throughput of the job.

In practice, it is more common to monitor and collect DL training throughput using popular DL frameworks. The throughput can be readily converted to iteration time given the training batch size. By measuring DL job throughput under both sole execution and concurrent execution with other jobs, we can fit the time model (Equation (3)) for both cases and naturally infer the interference ratio ξ. Figure 2 illustrates the throughputs of all DL models in our experiments across a range of resource allocations and batch sizes. Overall, our model closely represents the observed data. We also notice that different jobs exhibit varying sensitivities to network communication and GPU workloads. For instance, the ResNet20 model applied to the CIFAR-10 dataset demonstrates a linear increase in performance with batch size across all GPU configurations within the experimental range. This observation indicates that the bottleneck is related to GPU computation and is limited by GPU memory. On the contrary, the ResNet50 model, when applied to the ImageNet dataset, primarily achieves peak throughput with a local batch size of 128. This is largely due to the bottleneck present in the data loading process. We also measure the system throughput of different job pairs and training configurations, as depicted in Figure 3. We find that throughput can be fitted by Equation (12), albeit with different parameters from the solely running mode. Moreover, the interference ratios of different cases exhibit a wide range of up to 6 in our experiments, emphasizing that avoiding unfavorable cases is crucial for improving overall performance.

3.4 Problem Formulation

In this paper, our goal is to determine the scheduling decisions $y_{jg}[t]$ to minimize the average JCT, which is commonly used to evaluate the efficiency of DL job schedulers [16], [18]. This optimization problem can be formulated as follows:

$$\min_{y_{jg}[t],\forall j,g,t} \sum_{j\in J[t]} T_j \tag{13}$$

We note that Problem (13) presents an integer non-convex program with packing and covering constraints, which is NP-hard. Given these challenges, we first provide an overview of our entire Castor design in Section 4. Next, we will explore a heuristic approach that provides a provable local optimum guarantee for a job pair that intelligently shares GPUs in Section 5.

4 Castor Architectural Overview

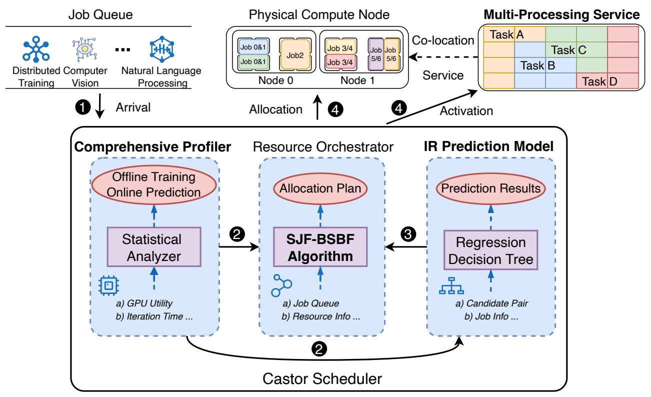


Figure 4. The overview of Castor's solution, which is primarily composed of four main components (displayed in bold).

In this section, we present our system that is capable of solving the problem outlined in Section 3. Our Castor comprises four essential components (displayed in bold) and the workflow of which is illustrated in Figure 4. Jobs must be profiled prior to the training process (❶), and we utilize a *Comprehensive Profiler* to record and analyze job attributes such as iteration time and resource utilization. After profiling, the *Profiler* simultaneously sends the relevant information (❷) to both the *Resource Orchestrator* and *IR Prediction Model*. When decisions regarding colocation are necessary, the *IR Prediction Model* generates predictions for the specified pair of jobs and returns the target value of the IR to the scheduling algorithm (❸). At the conclusion of a scheduling round, the *Resource Orchestrator* transmits the allocation plans to the physical hosts, initiating jobs on the designated devices (❹) and activating the Multi-processing Service to facilitate resource sharing if required.

Building Castor necessitates both algorithm design and systems support. In the following sections, we will detail the core optimization algorithm (Section 5) and highlight the implementation aspects of Castor (Sections 6, 7, and 8).

5 Castor Algorithm Design

In this section, we will elaborate on the core concept of the scheduling algorithm employed in the resource orchestrator. It is important to note that concurrent execution on a GPU can degrade overall performance if interference is significant, underscoring the relevance of **Theorem 1**. Building on **Theorem 1**, we introduce our scheduling algorithm, SJF-BSBF, which incorporates the shortest job first strategy. By carefully selecting job pairs that benefit from GPU sharing, even in a non-preemptive manner, Castor minimizes job queuing time while avoiding scenarios where sharing may adversely impact overall performance.

5.1 Scheduling One Job Pair

We assume that all the tasks of a DL job are assigned to a fixed set of GPUs during its execution. Before we design the scheduling algorithm, each new-arriving DL job should be placed in a certain set of intra-node or inter-node processors, which is called job placement.

Consider that if there is a new job sharing the GPUs occupied by the existing job. Job A and Job B

$$\hat{t}_A = t_A \xi_A \tag{14}$$

$$\hat{t}_B = t_B \xi_B \tag{15}$$

Assume κ is the insertion time. We have the following theorem.

Theorem 1 The shortest JCT of the above job pair is achieved by either sequentially executing them ($\kappa = t_A i_A$) or simultaneously invoking them concurrently $\kappa = 0$.

Proof. Case 1: If $\hat{t}_A i_A \geq \hat{t}_B i_B$, then

$$T_A = \hat{t}_B i_B + t_A \times (i_A - \frac{\hat{t}_B i_B}{\hat{t}_A}) \tag{16}$$

$$T_B = \hat{t}_B i_B \tag{17}$$

The average time is

$$\overline{T} = (T_A + T_B)/2 \tag{18}$$

$$= \hat{t}_B i_B + \frac{t_A i_A}{2} - \frac{\hat{t_B} i_B}{2\xi_A} \tag{19}$$

Case 2: If $\hat{t}_A i_A < \hat{t}_B i_B$, then

$$T_A = \kappa + \hat{t}_A \times (i_A - \frac{\kappa}{t_A}) \tag{20}$$

$$T_B = \kappa + \hat{t}_A \times (i_A - \frac{\kappa}{t_A}) + t_B \times (i_B - \frac{\hat{t_A} \times (i_A - \frac{\kappa}{t_A})}{\hat{t_B}}) \tag{21}$$

The average time is

$$\overline{T} = (T_A + T_B)/2 \tag{22}$$

$$= (\frac{2\xi_B + \xi_A - 2\xi_A\xi_B}{2\xi_B})\kappa + (1 - \frac{1}{2\xi_B})\hat{t_A} i_A + \frac{1}{2} t_B i_B \tag{23}$$

If $2\xi_B + \xi_A - 2\xi_A\xi_B > 0$, the function is monotonically increasing with respect to κ. The minimum value occurs at $\kappa = 0$, indicating that one should commence the new job immediately. Otherwise, the function is monotonically decreasing with respect to κ. The minimum value is achieved at $\kappa = t_A i_A$, which indicates that overlapping two jobs will harm the overall performance. □

In practice, evaluating the conditions for the best solution may be close to directly comparing the fully overlapped time and the fully non-overlapped time in terms of time cost.

5.2 Scheduling Multiple DL Jobs

One critical challenge in addressing Problem (13) is the allocation of GPUs when all GPUs in the cluster are occupied by existing jobs, and a new job arrives. This issue involves two key steps: determining which job to share resources with the new arrival, and deciding when to initiate the new job. In light of these challenges, the objective of our paper is to develop a straightforward yet effective online scheduling algorithm using a heuristic approach.

5.2.1 *Basic Idea*

We propose an online scheduling algorithm called SJF-BSBF (smallest job first with best sharing benefit first). Algorithm 1 describes the steps of SJF-BSBF. The intuition behind Algorithm 1 has three points. 1) As for job priority, the overall framework is based on the shortest job first (SJF) strategy, as tackled by Lines 1-2. This size-based heuristic strategy determines the job priority according to their used GPU numbers. We apply SJF since it performs well most of the time for online scheduling problems [16]. 2) As for GPU allocation, since the case of scheduling two jobs concurrently running on the same GPUs (fully or partially) is considered in our paper, when the free GPU number is not enough to execute the job, we try to look for those already occupied by the running jobs to schedule the new one. This is the core logic of SJF-BSBF and handled by Lines 3-19. 3) In the point of 2), for each job pair, we should also decide the batch size of the new job for gradient accumulation that not only exceeds the GPU memory size but also achieves the shortest JCT of scheduling the job pair. This corresponds to Line 11.

Algorithm 1 Shortest Job First with Best Sharing Benefit First

```
Input: The pending job list $\mathcal{J}_{pending}$ to allocate GPUs at
    the current scheduling time point. $L_k$ is the expected
    remaining time of $J_k$, calculated by $t^j_{iter} \times I_k$. $\mathcal{G}_{OJ}$
    denotes the GPU set occupied by at least one job. $\mathcal{J}_{share}$
    denotes the candidate job set for concurrent execution.
Output: $\mathcal{G}(J_k)$, The GPU sets of each job $J_k$ in $\mathcal{J}_{pending}$.
1: Sort $\mathcal{J}_{pending}$ by $L_k$ in ascending order.
2: for $J_k$ in $\mathcal{J}_{pending}$ do
3:     $\mathcal{G}(J_k) \leftarrow \emptyset$.
4:     $\mathcal{G}_{free} \leftarrow$ GPUs that hold no job.
5:     $\mathcal{G}_{OJ} \leftarrow$ GPUs that hold one job.
6:     if $|\mathcal{G}_{free}| \geq G_{J_k}$ then
7:         $\mathcal{G}(J_k) \leftarrow$ the top-$G_{J_k}$ GPUs in $\mathcal{G}_{free}$
8:     else
9:         if $|\mathcal{G}_{free}| + |\mathcal{G}_{OJ}| \geq G_{J_k}$ then
10:            for $g$ in $\mathcal{G}_{OJ}$ do
11:                Get $SF, b, t$ using Algorithm 2.
12:                Add $\mathcal{J}_g$ to $\mathcal{J}_{share}$ if SF is True.
13:            end for
14:            Sort $\mathcal{J}_{share}$ by $t$ in ascending order.
15:            for $J$ in $\mathcal{J}_{share}$ do
16:                Add $\mathcal{G}_J$ to $\mathcal{G}_{J_k}$ until $|\mathcal{G}_{J_k}| \geq G_{J_k}$.
17:            end for
18:        end if
19:    end if
20: end for
```

5.2.2 *GPU Allocation*

Given a job J_k that needs G_{J_k} GPUs, we should decide a set of GPUs to schedule it. The classical heuristic algorithms include First-Fit (FF) [45] and List-Scheduling (LS) [45]. In Algorithm 1, Lines 3-19 present the step of choosing the GPUs. First, if there are enough GPUs to execute J_k, we select the top-k GPU in $\mathcal{G}_{free}$ to make them as colidated on the nodes as possible (Lines 6-7). Second, notice that we allow at most two jobs to concurrently run on the same GPUs. Once the free GPU number is smaller that the request of J_k, we attempt to seek those GPUs that are already occupied by one job (Lines 10-17). We scan G_{OJ} and determine the best concurrent running setting of the running job and J_k, including the batch size of J_k and whether to let them share the GPUs, using Algorithm 2 introduced later. We add those pairs that can benefit from the sharing strategy (Lines 10-13). Then we sort $\mathcal{J}_{share}$ by the JCT of the job pair in ascending order (Line 14). Finally, we pick up the GPUs from those candidate jobs until the total number can fulfill the request of J_k (Lines 15-17). Notice that we do not pick the free GPUs at first for this case to save resources because the completion time of J_k is determined by those shared GPUs.

5.2.3 *Batch Size Scaling*

In Algorithm 2, given a running job J_{run} and a new job J_k ready to be scheduled, we present how to adaptively adjust the batch size for J_k to achieve the shortest average JCT of these two jobs. Notice that we do not adjust the batch size of the running job to reduce the complexity of the scheduling system. We search the batch size in the range [1, B_{J_k}] with a step of power two (Lines 5 and 12). For each candidate batch size, we use **Theorem 1** to obtain the best configuration of scheduling the job pair, including the flag of whether to let them share GPUs (SF) and the JCT (Line 6). Then we record that configuration if better (Lines 7-11). Notice that it is possible that the new job J_k may not be scheduled immediately and put back to the pending job pool if the final $\overline{SF}$ is $False$, indicating that running the job pair concurrently is not optimal.

Algorithm 2 Batch Size Scaling with Best Sharing Benefit

```
Input: A given job $J_k$ to allocate GPUs. The user's re-
    quested batch size $B_k$. $t_{share}$ represents the JCT of the
    best sharing configuration of two jobs.
Output: The sharing configuration denoted by $\overline{SF}, \overline{b}, \overline{t}$. $\overline{SF}$
    indicates whether to share the GPU for the new job.
1: $b \leftarrow B_{J_k}$
2: $\overline{b} \leftarrow B_{J_k}$
3: $\overline{t} \leftarrow$ a large number
4: $\overline{SF} \leftarrow False$
5: while $\lceil b \rceil \neq 1$ do
6:     Get $SF$ and $t_{share}$ according to Theorem 1.
7:     if $t_{share} \leq \overline{t}$ then
8:         $\overline{t} \leftarrow t$
9:         $\overline{b} \leftarrow b$
10:        $\overline{SF} \leftarrow SF$
11:    end if
12:    $b \leftarrow b/2$
13:    return $\overline{SF}, \overline{b}, \overline{t}$.
14: end while
```

5.2.4 *Time complexity of Castor*

The time consumption of Castor is primarily attributed to searching the GPU set for a pending job to be shared when there is insufficient resource (Lines 9 to Line 18 in Algorithm 1). Initially, a for loop (Line 10) scans all GPUs with one job, iterating $|\mathbb{G}(OJ)|$ times. Subsequently, each iteration executes Algorithm 2 to determine the time point to initiate GPU sharing as well as the appropriate batch size, with a time complexity of $\theta(\log 2(BJ_k))$. Lastly, after collecting candidate jobs for sharing GPUs, sorting the list in ascending order to select those with the shortest Job Completion Time (JCT) requires $\theta(|\mathcal{J}share|\log 2(|\mathcal{J}share|))$. Consequently, the time complexity for scheduling a job is $\theta(|\mathbb{G}(OJ)|\log 2(BJ_k) + |\mathcal{J}share|\log 2(|\mathcal{J}share|))$. In our system implementation on a 16-GPU cluster, the overhead of periodically scheduling those waiting jobs is negligible, averaging below 0.02 seconds for each operation.

6 IR Prediction Model

The value of ξ can be determined through mathematical interpolation of the ξ curve shown in Figure 3. However, this approach requires extensive case profiling and is limited by the upper and lower bounds of the available data. To address these limitations, we adopt a prediction model based on the Regression Decision Tree (RDT) [46], which achieves both intelligibility and accuracy while sparing us from the prohibitive costs associated with exploring an explosive combination space of job configurations.

6.1 Hardware Indicator Analysis

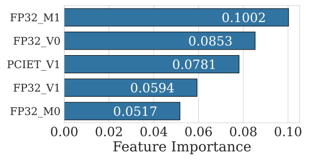
Since the interference ratio represents the intensity of resource contention during DL job execution caused by GPU sharing, its impact is naturally reflected in various hardware-level metrics, such as FLOPs and SM (Streaming Multiprocessor) utilization. Thus, it is reasonable to hypothesize that these hardware indicators can be analyzed to predict the value of ξ. To this end, we adopt a Regression Decision Tree (RDT) as the core of our prediction model, aiming to capture the underlying relationship between resource contention intensity and the hardware characteristics of DL jobs. Decision trees offer high interpretability and transparency in decision-making while maintaining strong performance in accuracy. Specifically, we employ the widely used XGBoost framework to construct a compact and accurate prediction model. In order to uncover the hidden correlations between job-level hardware features and resource contention, we collect detailed runtime metrics using the DCGM profiler. These metrics include SM activation rates, tensor core throughput, and communication bandwidth utilization, representing real-time GPU behavior. The complete set of DCGM indicators used for model training is summarized in Table 2; detailed descriptions of each metric are provided in [47]. Additionally, we augment the dataset with additional runtime metrics (CPU utilization and cross-node network communication bandwidth) to further enhance the model's expressiveness. These metrics are also presented in Table 2, alongside the DCGM indicators, to create a comprehensive input feature set for our prediction model.

Table 2
DCGM and other indicators collected as input for decision tree to train the prediction model.

Indicator	Resource Type	Notation
SMACT	Computation	Average rate of at least one warp active on SMs
SMOCC	Computation	Average rate of resident warps on SMs
TENSO	Computation	Average rate of active tensor pipe
DRAMA	Memory	Fraction of cycles with device memory exchange
FP32A	Computation	Average rate of active FP32 pipe
FP16A	Computation	Average rate of active FP16 pipe
PCITX	Communication	Rate of data transmitted over PCIe within node
PCIRX	Communication	Rate of data received over PCIe within node
GRACT	Computation	Fraction of time graphics engine active
NVLTX	Communication	Rate of data transmitted over NVLink within node
NVLRX	Communication	Rate of data received over NVLink within node
CPU	Comprehensive	Average rate of active CPU cores
NETTX	Communication	Rate of bits transmitted across nodes via IB
NETRX	Communication	Rate of bits received across nodes via IB

6.2 Training Decision Tree

To construct the training dataset for the regression decision tree, we first enumerate all possible combinations of job pairs and assign the corresponding value of ξ as the prediction label. The dataset is then randomly partitioned into training and evaluation sets based on job pairs, following an approximate 80:20 split. The training set is used to train the decision tree model, and its performance is subsequently evaluated on the held-out evaluation set. The top five features ranked by importance in the final regression decision tree are shown in Figure 5. Notably, indicators related to FP32 operations have the most significant impact on the model's predictions. Other hardware metrics also contribute meaningfully to the accuracy of the model, supporting precise and reliable prediction of the ξ value.

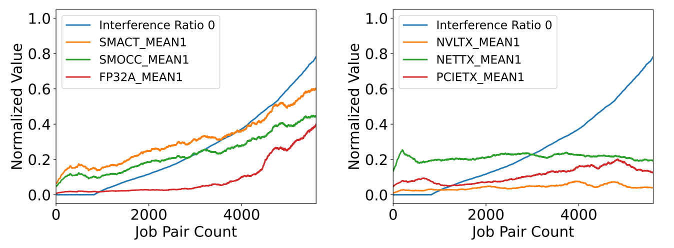


FP32_M1	Mean of FP32A on job1
FP32_V0	Variance of FP32A on job0
PCIET_V1	Variance of PCIETX on job1
FP32_V1	Variance of FP32A on job1
FP32_M0	Mean of FP32A on job0

Figure 5. Decision Tree Feature. Left: Importance of top 5 indicators. Right: corresponding notation (job0 represents the first job in the job pair, while job1 represents the second).

6.3 Revealing Correlation Between IR and Indicators

Based on the feature importance results shown in Figure 5, we further investigate the correlation between the interference ratio and the hardware indicators listed in Table 2. Given that NVIDIA MPS provides a software-based mechanism for GPU sharing without enforcing strict memory usage limits or performance isolation between MPS processes, resource contention can arise in three primary domains: computation, communication, and memory access (i.e., data I/O). Our goal is to identify which of these domains predominantly influences the value of ξ. Notably, four out of the five most important features identified by the regression model are associated with the FP32A indicator. According to NVIDIA's documentation [47], FP32A represents the fraction of cycles during which the FP32 and integer fused-multiply-add (FMA) pipeline is active. A higher



(a) Correlation among IR and computational indicators

(b) Correlation among IR and communicational indicators

Figure 6. Correlation among IR of job0 and computational indicators of job1 in each job pair.

FP32A value implies greater utilization of the FP32 compute cores. For instance, an FP32A value of 1.0 (100%) indicates continuous activity of the FP32 pipeline, whereas a value of 0.2 (20%) might imply either 20% of the SMs are fully utilized throughout the time period, or 100% of the SMs are active for only 20% of the time, or some combination thereof. In essence, FP32A is a strong indicator of computational intensity on the GPU. This observation supports a preliminary conclusion that computation is the dominant factor influencing resource contention and, consequently, the value of ξ. It is noteworthy that no cross-node network-related indicators appear in this top-five list. This is mainly attributable to the high-speed IB network in our testbed for the communication workload of distributed deep learning training, which generally provides sufficient cross-node bandwidth for our tested jobs. As a result, computational resources or intra-node network bandwidth become the bottleneck preventing further throughput improvements, rather than the cross-node bandwidth. To validate this hypothesis, we analyze the correlation between the interference ratio and various resource-related indicators. As illustrated in Figures 6(a) and 6(b) , the indicators associated with computational resources (Figure 6(a)) show a significantly higher correlation with the interference ratio, whereas those related to communication resources (Figure 6(b)) exhibit only weak correlations. These findings further confirm that computation-related metrics are the primary contributors to predicting the interference ratio experienced by a colocated job in a shared GPU environment.

6.4 Accuracy of Prediction Model

The Root Mean Square Error (RMSE) of the decision tree model for each job in the pair is 0.06 and 0.07, respectively. Notably, significant prediction error occurs only when the actual ξ value falls within the range $[1.4, 1.6]$; outside this range, the prediction error has minimal impact on the decision-making process discussed in Section 5.1. To further assess the accuracy of the decision tree-based prediction model, we conducted both physical and simulation-based experiments, as detailed in Section 9. The results demonstrate that although the prediction model occasionally produces estimations that deviate from those obtained using the ground truth (i.e., the mathematical interpolation method), the average deviation in sharing benefit across all jobs remains below 10%.

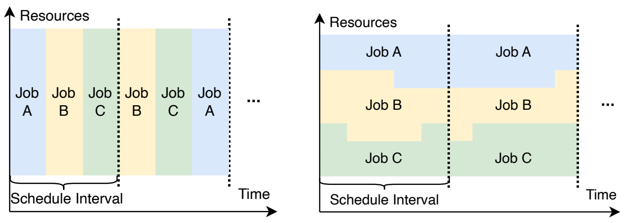
7 Comprehensive Job Profiler

To ensure the scheduling algorithm operates effectively, we develop a comprehensive profiler that models the performance of potential incoming jobs and relays the predictive results for both online service and offline training. Castor employs this profiling mechanism to enhance the scheduling policy outlined in Section 5.2. The *Comprehensive Job Profiler* establishes a short-term runtime limit, denoted as T_{prof}, for each job configuration and gathers performance data specific to the job. This data collection can capture the running features of DL jobs, alongside hardware metrics pertinent to predicting the interference ratio, as discussed in Section 6.1. Then, the profiler sends these features to the resource orchestrator and the prediction model (denoted as ❷ in Figure 4), which utilize the various data groups to proactively model performance and interference ratios, respectively. To facilitate a comprehensive evaluation on both physical and simulation platforms, we further design two distinct functions to enable experimental validation.

Online Prediction: In this scenario, the scheduler has no knowledge of the upcoming job sequence, nor does it possess any performance data regarding jobs under any configuration. Therefore, the profiling procedure will be forcibly conducted for every job within a series of GPU allocations during the time period T_{prof} before the job is admitted to the pending queue, as indicated by ❶ in Figure 4. It is noteworthy that the profiling series for GPU allocation must achieve a balance between data sufficiency and time efficiency. Specifically, when examining GPU allocation within a single node, we follow the principle of powers of two, namely $1, 2, 4, 8, \ldots$. In contrast, when investigating distributed allocation, we restrict our expansion to two nodes to minimize queueing times for short-term jobs while obtaining the cross-node communication impact on jobs' performance.

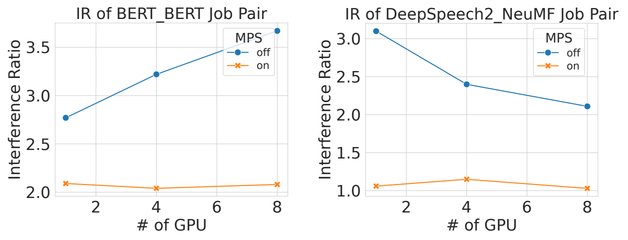
Offline Training: Unlike the online schema, the offline training procedure necessitates comprehensive information regarding job performance to maintain its integrity and credibility. Therefore, we conduct an extensive profiling process to collect detailed trace data for exploration in large-scale simulations that emulate the training step in the physical testbed. An elaborate example of the profiling cases for a specific DL job under various configurations is demonstrated in Figure 2. In particular, we decompose the training process of any DL job into the components of computation and communication. Our objective is to gather various GPU allocations to analyze computation, intra-node communication, and inter-node communication, thereby accurately modeling performance to attain high-level validity. By considering every potential DL job within a workload, the simulation can achieve both effectiveness and authenticity.

In summary, our Castor profiling mechanism possesses the following superiorities: (a) Comprehensiveness. The profiling results not only contribute to the performance modeling of DL jobs in online scenarios, but they also serve as a critical component in providing trace data for offline training simulations. (b) User-Friendly. Castor requires no code modifications in jobs submitted by users, allowing them to obtain results using the hyper-parameters they



(a) Temporal Sharing Paradigm (b) Spatial Sharing Paradigm

Figure 7. Illustration of the differences between temporal and spatial sharing paradigms.



(a) The IR of a BERT job when shares with another BERT job (b) The IR of a DeepSpeech2 job when shares with a NeuMF job

Figure 8. Differences between temporal and spatial sharing performance across various job pairs (displayed with a fixed global batch size).

specify [19]. (c) Enhancement of System Performance. By predicting job performance, the profiler provides the scheduler with an initial assumption about the submitted jobs in online schema. This allows the scheduler to generate an allocation plan based on the attributes of the jobs, thereby expediting the scheduling process and improving overall system efficiency. Furthermore, by utilizing our meticulously profiled trace data in the offline scenario, the validity and reliability of the simulator are ensured.

8 Efficient Sharing Through MPS

To illustrate the potential benefits of GPU sharing when conducting orthogonal studies aimed at enhancing parallel execution, we integrate the Multi-Process Service (MPS) [48] into our Castor solution as a technical support feature. The default temporal sharing paradigm of NVIDIA GPUs slices the flow of GPU execution into tiny shards, and jobs are arranged in a certain order to be served one after another, as is demonstrated in Figure 7(a).

Although it appears that the shared resources are providing service concurrently to all jobs running on them, these jobs are still accessing resources in a sequential approach. Furthermore, the overhead associated with context switching, which is introduced by this sequential approach, would be unacceptable when a cluster is under heavy workload. This could result in significant degradation of both resource utilization and system throughput. In order to minimize the detrimental performance degradation caused by temporal sharing, we adopt the Multi-Process Service (MPS) [48] to implement spatial sharing instead. A simple illustration of this sharing method is depicted in Figure 7(b).

The superiority of spatial sharing over temporal sharing can be directly reflected in the value of ξ (for ξ represents the extent of performance interference that one job imposes on the other job when sharing resources concurrently.). The ξ value of the BERT job under the spatial sharing paradigm (with MPS) is not only significantly more stable than that of the temporal sharing paradigm, but it is also remarkably lower by up to 43%, as illustrated in Figure 8(a). This result aligns with another tested ξ value for the job pair demonstrated in Figure 8(b), where the ξ is more volatile under the temporal sharing paradigm, with a value approximately 1.92 times higher than that of the spatial sharing paradigm.

9 Performance Evaluation

9.1 Experimental Setup

Cluster configurations: We first conduct physical experiments on a cluster of four servers. Each server is equipped with 128 Intel Xeon Platinum 8358 CPUs and 8 Nvidia A6000 GPUs. All nodes are inter-connected with 200-Gbps InfiniBand (IB). All experiments are performed in the environment of Ubuntu 20.04, PyTorch 1.12.1, CUDA 11.3, and OpenMPI 4.1.4. Based on the data measured in the physical environment, we then conduct simulation experiments to resemble the physical cluster configuration and test a large scale of clusters and job traces. To evaluate the performance of Castor in a large-scale cluster (16 servers, each with 8 GPUs) with long-term traces, we also implement a simulator to record job events and resource usage. All experimental results without explicit comments are derived from the simulation.

Baselines: We examine the following baselines and present a comparison among them across multiple dimensions in Table 3. Specifically, the 'Heuristic' attribute indicates whether a policy employs an explicit greedy mechanism during its scheduling process.

1) First-In-First-Out (FIFO): a traditional but popular policy adopted by several well-known cluster management systems, such as Yarn and Kubernetes. However, it usually performs poor due to its runtime-agnostic scheduling paradigm. Picks the top-G_j GPU with least execution time first.
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) Shortest Service First (SSF): similar to SJF, but SSF prioritizes jobs based on the required GPU service time rather than their completion time. This approach is more effective in alleviating congestion issues in heavy workload scenarios compared to SJF.
4) Tiresias [16]: a preemptive policy that prioritizes least attained service jobs (i.e., consumed GPU numbers and training iterations). It helps short-term jobs escape from resource starvation and finish earlier without any prior information.
5) Pollux [18]: the state-of-the-art elastic scheduler that adaptively adjusts the GPU resources for each job to optimize the overall job performance as well as

resource utilization. As explained in [23], Pollux cannot guarantee no accuracy degradation for all models as it allows the scheduler to tune the training batch size.

6) Lucid [23]: the state-of-the-art scheduler that dynamically enables job packing and colocation mechanisms based on prediction of submission traffic across the cluster. Lucid neglects GPU sharing for DDL jobs in its rules due to concerns about network contention. In contrast, our Castor considers GPU sharing for DDL jobs when colocation can enhance the overall JCT within the cluster.
7) SJF-FFS: we design SJF-FFS to establish a comparative baseline. This scheduler greedily resorts to colocation whenever idle resources are insufficient. It accentuates Castor's key improvement: the judicious activation of GPU sharing only when colocation contributes to enhanced overall JCT.

For physical experiments, we compare our Castor with FIFO, SJF, and Tiresias to demonstrate the advantages of resource sharing over those exclusive-mode policies. For simulation experiments, we also add Pollux, one of the state-of-the-art elasticity-based schedulers, to compare the sharing-based and the elasticity-based policies. We also compare Castor with SJF-FFS and Lucid in both physical and simulation experiments to illustrate the effectiveness of sensible GPU sharing and enabling colocation for DDL jobs over blindly-sharing and stale-restricted-sharing policies.

Table 3
Comparison among scheduling policies across multiple dimensions.

Policy	**Preemptive**	**Scalable**	**Heuristic**	**GPU Sharing**
FIFO	✗	✗	✗	✗
SJF	✗	✗	✔	✗
SSF	✗	✗	✔	✗
Tiresias	✔	✗	✔	✗
Pollux	✔	✔	✗	✗
Lucid	✗	✗	✔	✔
Castor (ours)	✗	✗	✔	✔

Workload Settings: We generate two different workloads; one is similar to the job trace of the Philly cluster at Microsoft [29], while the other resembles the job trace of the Venus cluster at SenseTime [28]. More details about the Philly and the Venus trace can be found in [6], [28] and the appendix of [16], [23]. For the physical experiments, considering that our testbed only has 4 nodes with 32 GPUs in total, we generated 30 jobs for the Philly workload and 60 jobs for the Venus workload by randomly sampling and scaling down the original trace. In terms of job characteristics, including the number of GPUs utilized and the number of training iterations, we predominantly adhere to the distributions observed in actual trace data. Specifically, 66% of jobs employ no more than 8 GPUs, while the rest of them utilize either 16 or 32 GPUs. Additionally, the training iterations for these jobs range from 200 to 2000. For the simulation experiments, we mainly follow the settings of Pollux and Lucid. We randomly sample 240 jobs from the busiest period in the DL cluster traces published by Microsoft and SenseTime. The settings of GPU numbers and training iterations also follow those of Pollux and Lucid. In particular, we leave Pollux out of the testing policies in the Venus workload for the corresponding gradient information of jobs is unavailable.

9.2 Evaluation Metrics

Job Completion Time (JCT): The average completion time of jobs in a DL workload not only represents the efficiency of a scheduling strategy but also reflects the overall user experience. We collect the completion time for each DL job once the model training is complete or the target iteration number is reached. We then calculate the arithmetic mean of all values to determine the overall JCT for the DL workload.

Makespan: The makespan represents the total time taken by the scheduler to complete the current workload, as well as the fairness of the scheduling strategy and the overall scheduling efficiency. We determine the makespan by calculating the difference between the timestamp of the first job submission and the timestamp of the last job completion.

Job Queuing Delay: The queuing delay of jobs refers to the average time that all jobs spend in the queue, which serves as an indicator of the average user experience. A low average queuing time reflects the overall scheduling efficiency of the scheduler and indicates that there are fewer starvation issues within the cluster. Specifically, it is the sum of the time periods spent waiting from the moment of submission until the completion of the job.

GPU Utilization: GPU utilization reflects the utility rate of cluster resources, with a higher utilization rate indicating greater economic efficiency of the cluster. We collect the SMACT metric from all GPUs using the NVIDIA DCGM tool (see Table 2) as an indicator of GPU utilization, as it offers both fine-grained visibility and broad insight into warp-level activities. In contrast, *nvidia-smi dmon* reports the Streaming Multiprocessor (SM) active rate with a minimum interval of one second, which fails to fully capture transient variations introduced by colocation.

Sharing Benefit: We introduce a metric called *sharing benefit* to evaluate the benefits of decisions on whether to perform GPU sharing, which takes the following forms:

$$\text{sharing\_benefit} = \frac{t_{\text{solo-location}}}{t_{\text{colocation}}} \tag{24}$$

where $t_{\text{solo-location}}$ represents the JCT of the current job, assuming it remains pending until sufficient resources are released. Conversely, $t_{\text{colocation}}$ denotes the JCT of the current job when executed in a GPU-sharing manner. It is important to note that the calculation above is based on the assumption that no new jobs will be submitted to the cluster. Specifically, a higher value indicates a greater benefit derived from the GPU-sharing decision.

9.3 Overall Evaluation on a Physical Cluster

JCT and Makespan: Figure 9(a) and 9(b) demonstrate the JCT distributions in the Philly and the Venus workloads under different scheduling policies. Over 90% of jobs require no more than 5 minutes of JCT when utilizing our Castor in the Venus workload, whereas other algorithms achieve this efficiency in less than 75% of cases. Castor generally achieves the best performance. In Table 4, it is reported that Castor demonstrates significant performance improvements compared to other policies by facilitating the colocation of jobs on each GPU. In particular, Castor achieves an average JCT that is 34.2% lower than that of Tiresias and 8.0% lower than that of Lucid in the Philly workload. These rates increase to 59.5% and 34.2% in the Venus workload, which has a larger workload size and is rich in short-duration jobs.

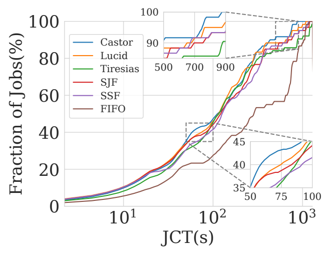
GPU Utilization: Figure 9(c) and 9(d) illustrate the activated SMs distribution in all GPUs of the cluster in the Philly and the Venus workloads using different scheduling policies, respectively. The distribution of time with 0% of activated SMs under the Castor policy is less than 15% and 10% for the Philly and Venus workloads, respectively. This is at least 50% lower than that of other policies, including the state-of-the-art GPU-sharing policy, Lucid. Specifically, since Lucid allows only restricted GPU sharing, the benefits of colocation within Lucid are relatively minor. Furthermore, Castor is the only policy capable of achieving 99% activation in SMs across the cluster. This advantage in resource utilization can be attributed to Castor's comprehensive exploration of the colocation space, overcoming the constraints imposed by sharing restrictions that hinder Lucid.

Job Queuing Delay: Figure 10(a) and 11(a) present the average queuing time of different scheduling policies in the Philly and Venus workloads. Firstly, the queuing time values of policies that allow colocation (Castor and Lucid) are generally lower than those of policies with the exclusive GPU mode. For the model Bert, Castor reduces the queuing time by over 63% compared to Tiresias in the Philly workload and nearly 66% in the Venus workload. Secondly, as illustrated in Figures 9(a) and 9(b), the queuing times for Castor and Lucid on jobs executed with ImageNet datasets (characterized by moderate GPU service time requirements) are generally shorter than those of other policies. This can primarily be attributed to the additional virtual resources generated by the sharing paradigm. Thirdly, both Castor and Lucid achieve nearly the same average queuing time on jobs executed with CIFAR10 datasets (characterized by lowest GPU service time requirements) as the heuristic but exclusive GPU mode policies (SJF and SSF). This indicates that the sharing paradigm does not hamper small and short jobs from gaining early access to resources.

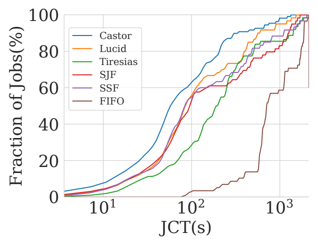
Table 4
Makespan and average job completion time (JCT) across different scheduling policies under the **Philly** and **Venus** workloads (measured via physical experiments).

Workload	Policy	Makespan (s)	Average JCT (s)
Philly	FIFO	1404	448.3
	SJF	1306	234.2
	SSF	1224	237.3
	Tiresias	1393	281.1
	Lucid	<u>*1207*</u>	<u>*201.1*</u>
	Castor	**928.1**	**185.1**
Venus	FIFO	2493	1005.7
	SJF	2448	408.2
	SSF	2326	337.8
	Tiresias	2312	390.4
	Lucid	<u>*2113*</u>	<u>*240.1*</u>
	Castor	**1618**	**158.6**

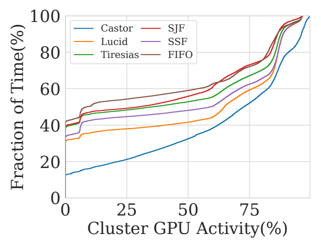
***Bold** indicates the best, and <u>*underline*</u> indicates the second best.



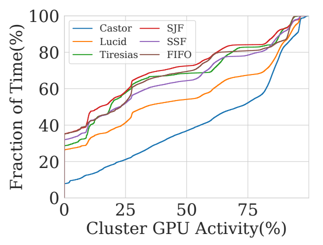
(a) JCT distribution of different scheduling policies in Philly workload



(b) JCT distribution of different scheduling policies in Venus workload



(c) GPU utilization distributions of different scheduling policies in Philly workload



(d) GPU utilization distributions of different scheduling policies in Venus workload

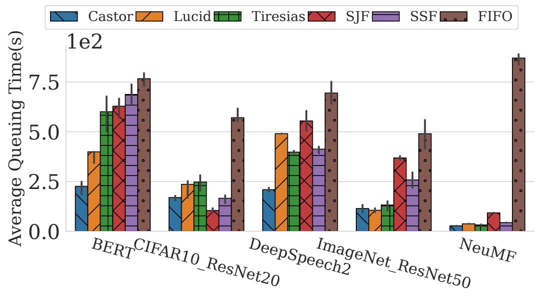
Figure 9. JCT & GPU utilization distributions of different scheduling policies in different workloads in physical experiments.

9.4 Overall Evaluation on Large-Scale Simulations

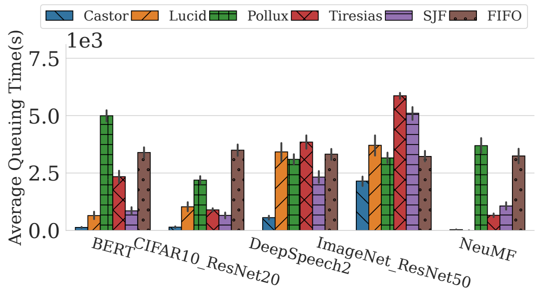
To verify the fidelity of our simulator, we also compare the results of physical experiments with simulations. We observe that the simulator can achieve the realistic experimental performance within 5% relative percentage errors on both makespan and average JCT. This confirms the high fidelity of our simulator.

JCT and Makespan: We first compare the JCTs of different scheduling policies in the standard simulation workload of both Philly and Venus. In Figures 12(a) and 12(b), it is evident that Castor outperforms other policies. Nearly 40% of jobs scheduled by Castor achieve JCTs of less than 900 seconds, resulting in a 55% reduction in the average JCT of the shortest 40% of jobs compared to Pollux. This demonstrates that the preemption-free policy can achieve better performance than preemptive policies, such as Tiresias and Pollux.

To investigate the impact of various scheduling policies on different types of jobs in finer granularity, we categorize jobs based on their GPU requirements. Jobs that require more than 8 GPUs are classified as large, while those that require 8 or fewer GPUs are classified as small. Table 5 presents the performance of various scheduling policies for a set of 240 jobs sampled from the Philly workload. In this scenario, Castor demonstrates superior performance compared to the advanced preemptive policy, Pollux. Although small, long-term jobs under Castor may experience longer JCTs than those under Pollux due to priority issues, large, short-term jobs benefit significantly from sharing GPUs with other jobs. This results in markedly shorter queuing times compared to other preemption-free policies. Similarly, we present the



(a) Results on a physical cluster



(b) Results on large-scale simulations

Figure 10. Average queueing time of different scheduling policies in Philly workload.

experimental results under the Venus workload in Table 6 employing the same categorization method as in the Philly workload scenario. In this circumstance, Castor outperforms the advanced preemption-free and colocation policy, Lucid, by 13.7% in average JCT and 55% in queuing time. This improvement can be attributed to the relaxation of Lucid's conservative colocation restrictions; Castor's decisions regarding GPU sharing are influenced solely by whether colocation will enhance the overall JCTs within the cluster.

GPU Utilization: Figures 12(c) and 12(d) illustrate the activated SMs distribution in all GPUs of the cluster using different scheduling policies in the Philly and Venus workloads, respectively. Notably, Pollux produces a significantly higher proportion of activated SMs at the mid-level (30%-50%) under the Philly workload, as demonstrated in Figure 12(c). This advantage is attributed to its ability to dynamically scale the number of GPUs. However, Pollux appears to be inferior in high-level activation of SMs when compared to the policies that facilitate GPU sharing, specifically Castor and Lucid. The superiorities of GPU sharing-allowed policies are particularly pronounced in the Venus workload, especially in the proportion of high-level (¿50%) SM activation, as illustrated in Figure 12(d). However, Lucid results in only a modest increase in GPU utilization across both workloads, with most of these improvements stemming from the low-level activation of the SM, which is partially attributed to its restricted sharing approach. The results indicate that policies allowing GPU sharing contribute to a higher proportion of high-level SM activation more efficiently than other policies. This advantage becomes even more pronounced under heavier workloads and more aggressive sharing decisions.

Job Queuing Delay: Figure 10(b) and 11(b) present comparisons of the average queuing time among different scheduling policies for various DL jobs. Notably, the policies that enable GPU sharing, namely Castor and Lucid, all yield lower queuing times compared to heuristic policies operating in exclusive GPU mode in most cases. Moreover, since Castor adopts a more aggressive GPU sharing approach than Lucid, it consistently leads to even lower queuing time compared to the results of Lucid. Furthermore, the reduction in queuing times is particularly evident in jobs that run for shorter periods of time. This can be partly attributed to the sharing paradigm, which generates additional virtual resources, along with the SJF-based sorting method that prioritizes shorter jobs, allowing them to access resources easier than longer jobs. As a result, most of the short jobs can be processed immediately after submission, effectively alleviating starvation issues more effectively than scaling-enabled policies, namely Pollux. Additionally, preemptive policies such as Tiresias and Pollux often exhibit longer queuing times attributable to job migration.

Table 5
Performance of large-scale and small-scale jobs by simulation under the **Philly** workload.

Metrics (hrs)	Policy	All Jobs	Large Jobs	Small Jobs
Average JCT	FIFO	2.64	1.04	1.60
	SJF	<u>1.88</u>	**0.36**	1.51
	Tiresias	2.57	1.31	1.26
	Pollux	2.17	1.09	1.08
	Lucid	2.10	1.32	**0.78**
	Castor	**1.54**	<u>0.45</u>	<u>1.09</u>
Average Queuing Time	FIFO	1.86	0.94	0.92
	SJF	1.09	**0.26**	0.83
	Tiresias	1.75	1.19	0.55
	Pollux	<u>1.00</u>	1.21	0.97
	Lucid	1.26	1.21	**0.05**
	Castor	**0.41**	<u>0.28</u>	<u>0.12</u>

*__Bold__ indicates the best, and <u>underline</u> indicates the second best.

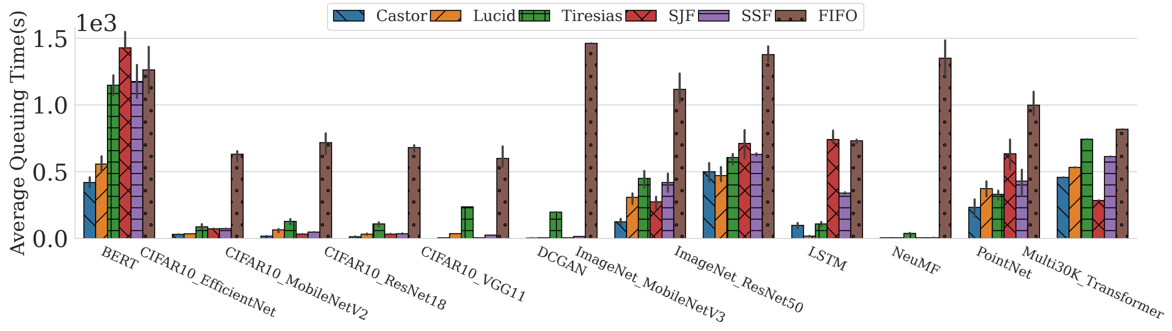
Table 6
Performance of large-scale and small-scale jobs by simulation under the **Venus** workload.

Metrics (hrs)	Policy	All Jobs	Large Jobs	Small Jobs
Average JCT	FIFO	4.73	1.94	2.79
	SJF	<u>3.35</u>	**0.72**	2.64
	Tiresias	3.94	1.76	<u>2.18</u>
	Lucid	<u>3.35</u>	<u>1.14</u>	2.21
	Castor	**2.89**	<u>1.13</u>	**1.76**
Average Queuing Time	FIFO	3.51	1.77	1.74
	SJF	<u>2.14</u>	**0.55**	1.59
	Tiresias	2.68	<u>1.58</u>	<u>1.10</u>
	Lucid	**2.07**	2.05	**0.02**
	Castor	**0.93**	<u>0.76</u>	<u>0.17</u>

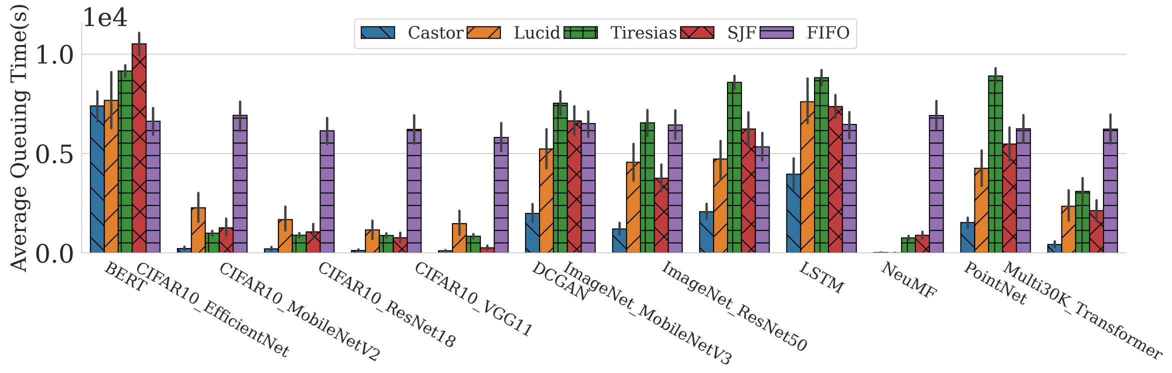
*__Bold__ indicates the best, and <u>underline</u> indicates the second best.

9.5 Ablation Study

We explore the impact of each component in Castor through ablation studies and conduct a sensitivity analysis of different workloads. The experiments in the ablation study are derived from the simulation evaluation if without any explicit notations.

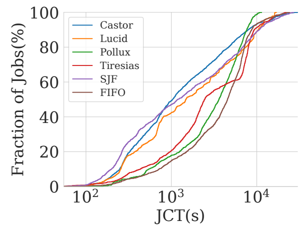


(a) Results on a physical cluster

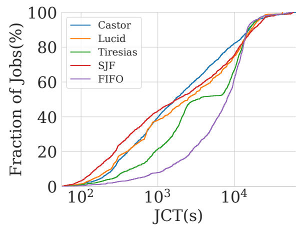


(b) Results on large-scale simulations

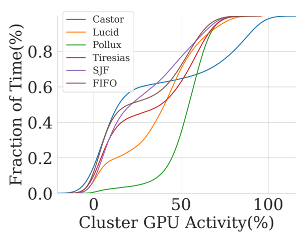
Figure 11. Average queueing time of different scheduling policies in Venus workloads.



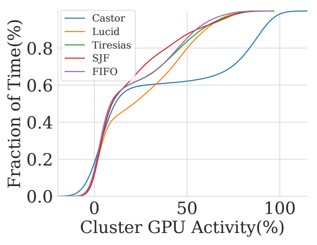
(a) JCT distributions of different scheduling policies in Philly workload



(b) JCT distributions of different scheduling policies in Venus workload



(c) GPU utilization distributions of different scheduling policies in Philly workload



(d) GPU utilization distributions of different scheduling policies in Venus workload

Figure 12. JCT & GPU utilization distributions of different scheduling policies in different workload on simulation evaluation.

Impact of IR Prediction Model: To assess the effectiveness of using decision trees as a prediction method for colocation decisions, we introduce a new metric called *sharing benefit*. This metric is defined in Section 9.2 and validated through extensive simulation experiments conducted in the baseline workload intensities of the Philly and Venus workloads. The results for each workload are presented in Figures 13(a) and 13(b), respectively. These results demonstrate that the GPU sharing decisions made by Castor closely align with those of the mathematical fitting-based approach across both workloads, as the average values of both JCT and sharing benefit in Castor and the fitting method are nearly identical. Furthermore, these findings validate the effectiveness of the decision tree-based prediction model in guiding GPU sharing decisions and demonstrate its practicality for use in real-world DL cluster scheduling, given that an insignificant deviation persists. It also provides strong evidence for the reliability of the prediction model discussed in Section 6.4.

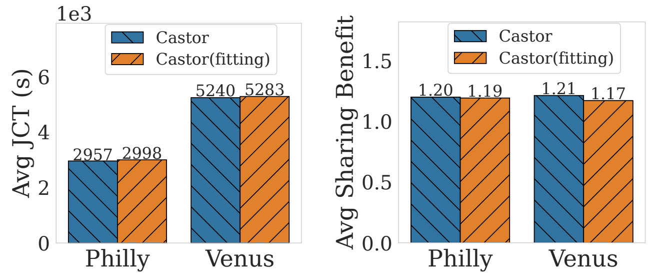
Impact of Sharing Strategies: To elaborate on the benefits of prudent sharing compared to a reckless approach, we conduct experiments under similar preconditions and analyze the same metrics for the Castor and SJF-FFS policies. Figures 14(a) and 14(b) illustrate the advantages of wise GPU colocation for the Philly and Venus workloads. It is noteworthy that Castor demonstrates a significant improvement over the SJF-FFS in the Venus workload, whereas the enhancement observed with the Philly workload is relatively modest. This can be attributed to two main factors. First, the base workload intensity is relatively low, which results in minimal long-term queuing and restricts the potential for improvement through GPU sharing. Second, because the trace data in our simulation environment is derived from a testbed integrated with the MPS technique, the reduced queuing times facilitated by GPU sharing often outweigh the performance penalties associated with resource contention resulting from colocation. However, this does not apply to the Venus workload, which is more congested with jobs and results in a greater number of traps when SJF-FFS attempts to perform GPU sharing, regardless of the overall consequences. Conversely, Castor can skillfully navigate some of these pitfalls due to its capacity to conduct a careful assessment of whether implementing colocation will prolong the overall JCTs.

Impact of Sharing Techniques: The differences between the temporal and spatial sharing paradigms, along with the theoretical and intuitive advantages of spatial sharing, are roughly outlined in Section 8. To conduct a more comprehensive and nuanced study, we perform a series of comparisons on our physical testbed across multiple dimensions and various metrics to assess the superiority of the spatial sharing paradigm (with MPS) compared to the default temporal sharing (without MPS). As illustrated in Figures 15(a) and 15(b), Castor achieves relatively low JCT, makespan, and queuing delay when employing the MPS technique, compared to scenarios without it in both workloads. Specifically, Castor demonstrates a significantly lower makespan in the Philly workload and remarkably reduced JCT and queuing time in the Venus workload when using MPS. Consistently, Castor also exhibits an extraordinarily higher average GPU utilization rate and sharing benefit compared to the cases without MPS.

Sensitivity to Intensity of Workload: We evaluate the performance of Castor against existing scheduling policies under varying workload intensities. Specifically, we scale the baseline workload of 240 jobs by factors ranging from $0.5\times$ to $2\times$, resulting in job submission counts from 120 to 480. The results are illustrated in Figure 16. An interesting observation is that Pollux outperforms other policies under low workload intensities. This suggests that Pollux is better suited for lightly loaded clusters, as its adaptive job batch sizing and resource scaling mechanisms become less effective when the cluster is overloaded. This observation is consistent with findings in prior studies [19], [23]. However, as workload intensity increases and GPU resources become scarce, Pollux's performance deteriorates due to its inability to leverage its adaptive strategies effectively. In contrast, across all workload levels, Castor consistently maintains a relatively low JCT growth rate compared to baseline policies. This advantage stems from Castor's ability to intelligently share GPU resources among jobs, thereby significantly reducing queuing time. Lucid, which supports limited GPU sharing, achieves moderate improvements in average JCT. However, its constrained solution space limits its ability to adapt to high-intensity workloads, especially when compared to Castor's more flexible design. This highlights the effectiveness of Castor's judicious GPU sharing strategy, which avoids harmful job placements and enhances overall system efficiency.

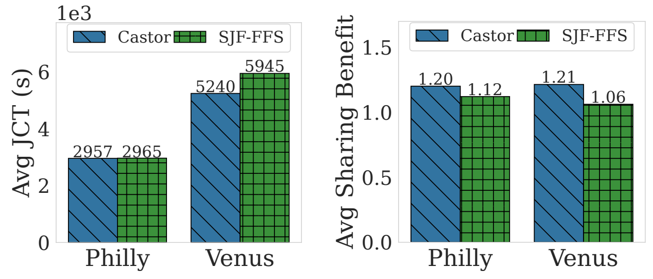
10 Conclusion

In this paper, we delve into the problem of resource scheduling for DL jobs in a multi-tenant GPU cluster, leveraging GPU sharing together with the MPS technique to reduce job queuing times and improve overall system performance. We begin by formulating a DL scheduling model that enables multiple jobs to share the same set of GPUs while employing gradient accumulation to address memory limitations and preserve training accuracy. Building upon this model, we derive the optimal scheduling strategy for a pair of jobs colocated on the same GPUs. We then propose a DL job scheduling system, Castor, integrating the heuristic algorithm SJF-BSBF and a precise IR prediction model to harness the benefits of GPU sharing while minimizing interference with running jobs. To validate the



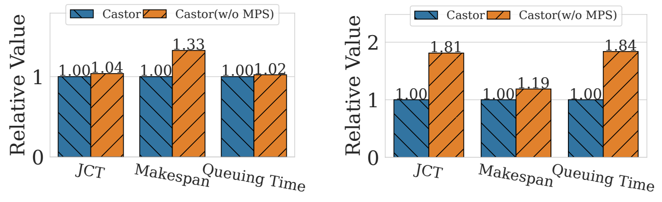
(a) Philly workload (b) Venus workload

Figure 13. Average JCT and Sharing Benefit of different IR derivation approaches.

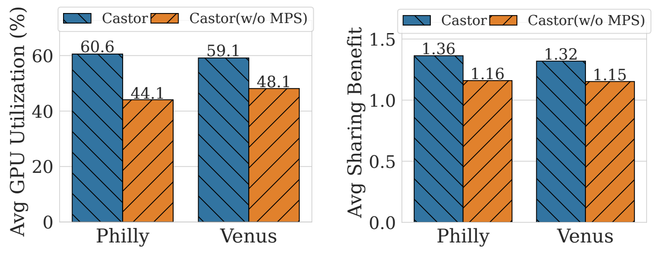


(a) Philly workload (b) Venus workload

Figure 14. Average JCT and Sharing Benefit of different sharing strategies.

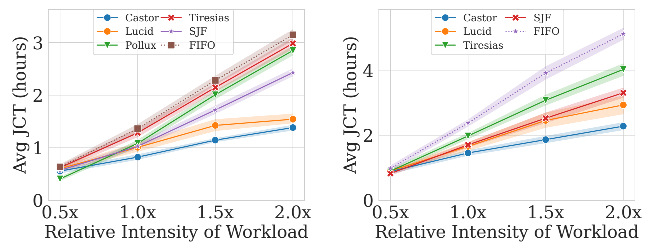


(a) Relative values in Philly workload (b) Relative values in Venus workload



(c) Average GPU utilization in different workloads (d) Average sharing benefit in different workloads

Figure 15. Comparisons among multiple metrics of different sharing techniques.



(a) Varying the intensity in Philly workloads. (b) Varying the intensity in Venus workloads.

Figure 16. Effects of varying the intensity in different workload on simulation evaluation.

effectiveness of our approach, we conduct extensive evaluations through both physical deployment and large-scale simulation experiments. The results demonstrate that the non-preemptive Castor significantly outperforms advanced preemptive schedulers such as Tiresias and Pollux, primarily by exploiting GPU sharing opportunities. Furthermore, Castor outperforms the state-of-the-art colocation-enabled scheduler Lucid by judiciously selecting sharing job pairs based on the predicted interference ratio, rather than relying solely on job type. This fine-grained approach allows Castor to make more informed and adaptive scheduling decisions. Moreover, our findings highlight the critical importance of configuring appropriate GPU sharing settings, as improper configurations can lead to significant performance degradation due to intensified resource contention. In particular, we illustrate and validate the advantages of the spatial sharing paradigm over temporal sharing.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (No. 62302126, No. 62302123), the Shenzhen Science and Technology Program (No. RCBS20221008093125065, No. KJZD20240903104103005, No. KJZD20230923114213027 and No. KJZD20230923115113026), Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies (2022B1212010005).

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] R. Zeng, C. Zeng, X. Wang, B. Li, and X. Chu, "Incentive mechanisms in federated learning and a game-theoretical approach," *IEEE Network*, vol. 36, no. 6, pp. 229–235, 2022.

[3] Y. Yao, Y. Hu, Y. Dang, W. Tao, K. Hu, Q. Huang, Z. Peng, G. Yang, and X. Zhou, "Workload-aware performance model based soft preemptive real-time scheduling for neural processing units," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 6, pp. 1058–1070, 2025.

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *NeurIPS*, 2020.

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng, "Large Scale Distributed Deep Networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1223–1231.

[6] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant gpu clusters for dnn training workloads," *arXiv preprint arXiv:1901.05758*, 2019.

[7] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. ACM, 2015, pp. 407–420.

[8] F. Giroire, N. Huin, A. Tomassilli, and S. Pérennes, "When Network Matters: Data Center Scheduling with Network Tasks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 2278–2286.

[9] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, "Tetrisched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters," in *Proceedings of the Eleventh European Conference on Computer Systems*, ser. EuroSys '16. ACM, 2016.

[10] Y. Liu, H. Xu, and W. C. Lau, "Online job scheduling with resource packing on a cluster of heterogeneous servers," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 1441–1449.

[11] C. Chen, X. Ke, T. Zeyl *et al.*, "Minimum Makespan Workflow Scheduling for Malleable Jobs with Precedence Constraints and Lifetime Resource Demands," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, July 2019.

[12] X. Mei, X. Chu, H. Liu, Y. Leung, and Z. Li, "Energy efficient real-time task scheduling on cpu-gpu hybrid clusters," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[13] H. Yabuuchi, D. Taniwaki, and S. Omura, "Low-latency job scheduling with preemption for the development of deep learning," in *USENIX Conference on Operational Machine Learning (OpML)*, 2019.

[14] A. Hsu, K. Hu, J. Hung, A. Suresh, and Z. Zhang, "Tony: An orchestrator for distributed machine learning jobs," in *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, 2019, pp. 39–41.

[15] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An Efficient Dynamic Resource Scheduler for Deep Learning Clusters," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18, 2018.

[16] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A GPU Cluster Manager for Distributed Deep Learning," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 485–500.

[17] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-Aware cluster scheduling policies for deep learning workloads," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, Nov. 2020, pp. 481–498.

[18] A. Qiao, S. K. Choe, S. J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G. R. Ganger, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021, pp. 1–18.

[19] S. Agarwal, A. Phanishayee, and S. Venkataraman, "Blox: A modular toolkit for deep learning schedulers," in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys '24, 2024.

[20] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective Cluster Scheduling for Deep Learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 595–610.

[21] G. Lim, J. Ahn, W. Xiao, Y. Kwon, and M. Jeon, "Zico: Efficient GPU memory sharing for concurrent DNN training," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, Jul. 2021, pp. 161–175.

[22] P. Yu and C. Mosharaf, "Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications," in *Proceedings of the third MLSys Conference*, 2019.

[23] Q. Hu, M. Zhang, P. Sun, Y. Wen, and T. Zhang, "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*, ser. ASPLOS 2023, 2023, p. 457–472.

[24] Z. Huang, B. Jiang, T. Guo, and Y. Liu, "Measuring the impact of gradient accumulation on cloud-based distributed training," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2023, pp. 344–354.

[25] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, *GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism*, 2019.

[26] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for DNN training," in *SOSP*. ACM, 2019, pp. 1–15.

[27] Y. Luo, Q. Wang, S. Shi, J. Lai, S. Qi, J. Zhang, and X. Wang, "Scheduling deep learning jobs in multi-tenant GPU clusters via wise resource sharing," in *IWQoS*. IEEE, 2024, pp. 1–10.

[28] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale GPU datacenters," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, B. R. de Supinski, M. W. Hall, and T. Gamblin, Eds. ACM, 2021, p. 104. [Online]. Available: https://doi.org/10.1145/3458817.3476223

[29] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of Large-Scale Multi-Tenant GPU clusters for DNN training workloads," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, Jul. 2019, pp. 947–960.

[30] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-Efficient Cloud Resource Provisioning for Predictable Distributed Deep Neural Network Training," in *2019 48th International Conference on Parallel Processing (ICPP)*, Aug 2019.

[31] Y. Bao, Y. Peng, and C. Wu, "Deep Learning-based Job Placement in Distributed Machine Learning Clusters," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019.

[32] Z. Hu, J. Tu, and B. Li, "Spear: Optimized Dependency-Aware Task Scheduling with Deep Reinforcement Learning," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, July 2019.

[33] C. Hwang, T. Kim, S. Kim, J. Shin, and K. Park, "Elastic resource sharing for distributed deep learning," in *NSDI*. USENIX Association, 2021, pp. 721–739.

[34] Z. Mo, H. Xu, and C. Xu, "Heet: Accelerating elastic training in heterogeneous deep learning clusters," in *ASPLOS (2)*. ACM, 2024, pp. 499–513.

[35] S. J. Subramanya, D. Arfeen, S. Lin, A. Qiao, Z. Jia, and G. R. Ganger, "Sia: Heterogeneity-aware, goodput-optimized ml-cluster scheduling," in *SOSP*. ACM, 2023, pp. 642–657.

[36] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, Renton, WA, Apr. 2022, pp. 945–960.

[37] Q. Wang, S. Shi, C. Wang, and X. Chu, "Communication contention aware scheduling of multiple deep learning training jobs," *arXiv preprint arXiv:2002.10105*, 2020.

[38] M. Yu, Y. Tian, B. Ji, C. Wu, H. Rajan, and J. Liu, "Gadget: Online resource optimization for scheduling ring-all-reduce learning jobs," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, p. 1569–1578.

[39] M. Yu, B. Ji, H. Rajan, and J. Liu, "On scheduling ring-all-reduce learning jobs in multi-tenant gpu clusters with communication contention," in *Proceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, ser. MobiHoc '22, 2022, p. 21–30.

[40] R. Rabenseifner, "Optimization of Collective Reduction Operations," in *Computational Science - ICCS 2004*, M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds., 2004, pp. 1–9.

[41] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, p. 49–66, Feb. 2005.

[42] T. Hoefler, W. Gropp, R. Thakur, and J. L. Träff, "Toward performance models of mpi implementations for understanding application scaling issues," in *Recent Advances in the Message Passing Interface*, 2010.

[43] C.-C. Huang, G. Jin, and J. Li, "Swapadvisor: Pushing deep learning beyond the gpu memory limit via smart swapping," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, 2020, p. 1341–1355.

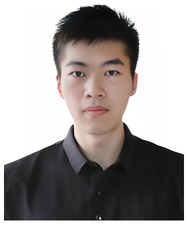
[44] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient GPU cluster scheduling," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 289–304.

[45] G. L. Stavrinides and H. D. Karatza, "Scheduling multiple task graphs in heterogeneous distributed real-time systems by exploiting schedule holes with bin packing techniques," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 540 – 552, 2011.

[46] K. S. Fong and M. Motani, "Symbolic regression enhanced decision trees for classification tasks," in *AAAI*. AAAI Press, 2024, pp. 12 033–12 042.

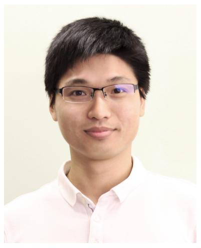
[47] NVIDIA, "Nvidia dcgm," https://docs.nvidia.com/datacenter/dcgm/3.1/user-guide/index.html, 2023.

[48] ——, "Nvidia multi-process service," https://docs.nvidia.com/deploy/mps/index.html, 2023.



Yizhou Luo is currently a master's degree candidate at Harbin Institute of Technology (HIT), Shenzhen China. He received his bachelor's degree from Harbin Institute of Technology, Shenzhen China. His research interests include efficient optimization in service systems and job deployment in distributed environments, with a particular interest in resource and job scheduling.

Jiaxin Lai is currently a master's degree candidate at Harbin Institute of Technology(HIT), Shenzhen China. He received his bachelor's degree from Harbin Institute of Technology, Shenzhen China. His research now focuses on optimizing Large Language Model inference services, with special emphasis on energy-efficient deployment and scheduling strategies.



Shaohuai Shi received a B.E. degree in software engineering from South China University of Technology, P.R. China, in 2010, an MS degree in computer science from Harbin Institute of Technology, P.R. China in 2013, and a Ph.D. degree in computer science from Hong Kong Baptist University in 2020. He is currently a professor in the School of Computer Science and Technology at Harbin Institute of Technology, Shenzhen. His research interests include GPU computing and machine learning systems. He was an awardee of the best paper award of IEEE INFOCOM 2021 and a member of the IEEE.

Chen chen (Member, IEEE) received the BEng degree from Tsinghua University, in 2014, and the PhD degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, in 2018. He is an associate professor with John Hopcroft Center for Computer Science, Shanghai Jiao Tong University. His recent research interests include distributed deep learning, big data systems and networking. He previously worked as a researcher with the Theory Lab, Huawei Hong Kong Research Center.

Shuhan Qi (IEEE Member) received his Ph.D. from Harbin Institute of Technology in 2017. He is currently a professor in the Department of Computer Science at Harbin Institute of Technology (Shenzhen). His research interests include game theory, multi-agent, and machine learning.

Jiajia Zhang Jiajia Zhang received his M.S. and Ph.D. degrees in Computer Sciences from Harbin Institute of Technology in 2009 and 2015 respectively. He worked as a post doctor in Peking University from 2015 to 2018. Now, he is an associate research fellow of Harbin Institute of Technology (Shenzhen). His main research interests include artificial intelligence, computer game, intelligence strategy decision and cyberspace security. Email: zhangjiajia@hit.edu.cn

Qiang Wang is currently an Associate Professor in Harbin Institute of Technology (HIT), Shenzhen China. He was a Research Assistant Professor in the Department of Computer Science, Hong Kong Baptist University before he joined HIT. Dr. Wang received his B.Eng. degree from South China University of Technology in 2014, and the Ph.D. degree in Computer Science from Hong Kong Baptist University in 2020. His research interests include general-purpose GPU computing and power-efficient computing. He received the best paper award of EuroSys 2025 and is a member of the IEEE.