Future Generation Computer Systems 183 (2026) 108596

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Concord: A GPU cluster scheduler with enhanced interference profiling and asymmetric job packing

Xinhua Wang [a], Weiwei Lin [a,b,*], Haijie Wu [a], Jidong Zhai [c], Keqin Li [d]

[a] Department of Computer Science and Engineering, South China University of Technology, GuangZhou, 510000, China
[b] Pengcheng Laboratory, Shenzhen, 518066, China
[c] Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Keywords:
DL workload
GPU scheduling
GPU utilization
GPU-sharing
Interference
Cloud computing

ABSTRACT

Due to the complex patterns of workload interference, consolidating multi-GPU (mGPU) and single-GPU (sGPU) jobs presents significant challenges in GPU sharing-enabled clusters. Existing workload schedulers struggle to effectively pack workloads with varied GPU requirements, which hinders overall system performance and efficiency. To address the challenges arising from the co-location of mGPU and sGPU workloads, we analyze job behaviors under various interference settings and propose Concord, a novel GPU sharing-enabled workload scheduler. First, we analyze the interference patterns resulting from the co-location of sGPU and mGPU jobs and propose lightweight interference prediction models. Subsequently, we derive a job placement policy with sharing incentives to ensure that average job completion time (JCT) is not degraded after packing. Unlike existing schedulers, Concord efficiently handles unpredictable DL workloads without relying on job duration estimation. Extensive evaluations using real-world cluster traces demonstrate that Concord outperforms state-of-the-art GPU sharing-enabled cluster schedulers across multiple metrics. Specifically, Concord outperforms state-of-the-art schedulers, achieving a 1.68× reduction in JCT and a 29% improvement in GPU utilization in high-load scenarios.

1. Introduction

The rapid advancements in deep learning (DL) have spurred its widespread adoption across a variety of domains, including image classification [1], control systems [2], and language translation [3]. As the computational demands of DL workloads continue to grow, both industry and academia have made substantial investments in large-scale GPU infrastructure to meet these needs [4]. However, empirical studies show that GPU utilization remains suboptimal, leading to inefficiencies in resource allocation and increased operational costs [5,6]. This is due to existing resource managers such as Kubernetes [7] and YARN [8] prohibiting the explicit use of GPU sharing (i.e., only allowing a single DL job to be assigned to each GPU).

Job packing – the co-location of multiple jobs on a GPU – has emerged as an effective technique to mitigate underutilization. While increasing the number of colocated jobs improves resource utilization and reduces queue delay, it also introduces interference that can significantly degrade performance [9]. Consequently, interference management has become a central focus of prior work [10–13].

We decompose job packing into three stages: S1, defining which packing patterns the scheduler supports; S2, predicting their performance under packing; and S3, deciding whether candidate packed jobs qualify for sharing incentives. Existing GPU sharing schedulers largely focus on S2, either by collecting online utilization metrics for performance analysis [12] or by predicting utilization and model characteristics of unseen workloads [13]. In contrast, S1 and S3 remain underexplored.

S1: Packing pattern support. Existing work is typically limited to packing jobs of identical scale—e.g., two sGPU jobs, two dual-GPU jobs, or two 4-GPU jobs [10,12,13]. We refer to this as symmetric packing (illustrated in Fig. 1(a) and (b)). However, symmetric packing is not always effective in practice, since production clusters run workloads of heterogeneous scales, including both sGPU and mGPU jobs (see Section 2.3). For example, a 4-GPU job can benefit from packing only when co-located with another 4-GPU job. However, in the Saturn cluster [5], 4-GPU jobs account for only 8.4% of the workload, implying that such jobs fail to exploit sharing opportunities in over 90% of

* Corresponding author at: Department of Computer Science and Engineering, South China University of Technology, GuangZhou, 510000, China.
E-mail addresses: 202111088204@mail.scut.edu.cn (X. Wang), linww@scut.edu.cn (W. Lin), 202420143306@mail.scut.edu.cn (H. Wu), zhaijidong@tsinghua.edu.cn (J. Zhai), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.future.2026.108596
Received 15 January 2026; Received in revised form 5 May 2026; Accepted 9 May 2026
Available online 15 May 2026

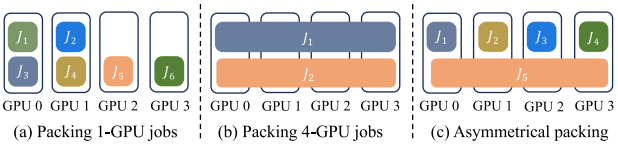

Fig. 1. Comparison of *Symmetrical packing* and *Asymmetrical packing*. Symmetrical packing includes (a) and (b), where only jobs with identical GPU demands can be co-located. In contrast, Asymmetrical packing additionally supports (c), allowing both sGPU and mGPU jobs to be packed together on a 4-GPU node.

cases. This skewed distribution is also observed in the Venus and Philly clusters [14], indicating that GPU scale heterogeneity fundamentally constrains the effectiveness of symmetric packing.

To overcome this limitation, we introduce asymmetric packing (illustrated in Fig. 1(c)), which enables sGPU and mGPU jobs to be packed on the same server. Revisiting the previous example, a 4-GPU job can be co-located with up to four sGPU jobs, all executing concurrently on a 4-GPU server. This design offers two key advantages. First, it substantially increases packing opportunities for mGPU jobs by removing the requirement for same-scale arrivals. Second, it improves job concurrency. Under symmetric packing, a 4-GPU server can host at most two 4-GPU jobs. In contrast, under asymmetric packing, the same server can accommodate one 4-GPU job together with four sGPU jobs, for a total of five concurrent jobs—corresponding to a 2.5× increase in job concurrency. Despite these advantages, asymmetric packing introduces two critical challenges (S2 and S3) that must be addressed.

S2: Job Performance Prediction. Existing approaches predominantly rely on symmetric packing and employ binary classification models that predict whether a job pair can be safely co-located [12, 13]. While such methods can determine whether two jobs should be packed, they fail to capture the performance of each individual job after packing. This limitation stems from a fundamental difference in interference pattern. Under symmetric packing, interference is confined to each paired job, whereas asymmetric packing introduces interference propagation: performance degradation in an sGPU job can slow down the co-located mGPU job, which in turn propagates the slowdown to all co-located sGPU jobs (Section 3). To accurately model this propagation, we must track performance variations at the granularity of individual jobs.

To this end, we propose *Interference Profiling*, a method that systematically characterizes interference patterns by representing each job along two dimensions: its impact on the performance of co-located jobs and its sensitivity to external interference. This approach is conceptually similar to prior work that uses short-duration or local profiling to derive interference-aware workload representations, such as Paragon [15] and the bubble-up methodology [16,17]. However, rather than summarizing interference behavior with a single score, we represent each job using a feature vector that captures both interference sensitivity and influence capability. This representation supports fine-grained performance prediction under asymmetric packing scenarios.

S3: Sharing Incentive. Average job completion time (JCT) is the primary optimization objective in this work. We consider a multi-tenant training cluster where jobs arrive online and are scheduled dynamically. In such environments, JCT is widely regarded as a critical metric for scheduler performance, as it captures both execution time and queueing delay. While job packing reduces queueing delay by enabling immediate execution, it also introduces interference that slows down job execution. To quantify this effect, we define *speed* as the normalized throughput of a job under co-location relative to its throughput during exclusive execution. Lower values indicate performance degradation due to interference. As a result, indiscriminate packing can increase JCT across the cluster, sometimes even performing worse than no packing at all. Users are only willing to pack jobs if the scheduling strategy ensures that the overall JCT of packed job pairs does not exceed that of non-packed jobs. This term as sharing incentive [18]. A straightforward approach is to pack only job pairs with low interference, as adopted in prior work. For example, Lucid [12] only packs jobs with at least 80% speed. Although such conservative strategies protect JCT, they miss many profitable packing opportunities and limit the benefits of sharing. Consider job J_1 (duration 10) arriving after job J_2 (duration 300) has already started execution: queuing yields average JCT of 305, whereas packing with 0.1× speed (severe interference) achieves JCT of 245—a 20% improvement.

Our key observation is that the impact of packing on JCT depends not only on inter-job interference but also on job duration. Specifically, jobs with highly imbalanced durations can often be packed together regardless of interference. Building on this insight, we design an *Incentive-Aware Packing Placement* algorithm. This algorithm guarantees non-degradation of JCT in most scenarios, while importantly requiring no prior knowledge of job durations—a parameter that is typically unknown in practice [19].

In summary, the main contributions of this paper are as follows:

- We propose Concord, a novel GPU-sharing scheduler that enables efficient packing of sGPU and mGPU jobs.
- We characterize the behavior of asymmetric packing and introduce a feature representation based on *interference impact* and *interference sensitivity*, together with lightweight interference prediction models for estimating job speed under co-location.
- We analysis the job packing process and show that the JCT of packed jobs depends on both job duration and speed under interference. Building on this analysis, we establish a non-degradation bound on JCT, ensuring that, in most scenarios, jobs are incentivized to pack without relying on job duration estimation.
- We evaluate Concord on two generations of GPUs using real-world cluster traces and demonstrate that it outperforms state-of-the-art GPU-sharing-enabled cluster schedulers. Specifically, Concord reduces JCT by 1.68×, increases GPU utilization by 29%, significantly improves job concurrency, and reduces queuing time.

2. Background and motivation

2.1. Deep learning training

DL models consist of neural network units that perform nonlinear transformations on tensors to extract features, with model parameters updated through gradient descent and backpropagation. DL training jobs exhibit several key characteristics that significantly impact resource scheduling and system design.

Iterative Computing. DL training follows a structured iterative paradigm wherein each training epoch consists of sequential mini-batch processing phases, with model parameters updated through gradient descent optimization. This inherent periodicity enables predictive profiling of resource usage and facilitates systematic analysis of interference effects under concurrent execution.

Trial-and-error exploration. DL model training often involves repeated experimentation with numerous hyperparameters. To achieve optimal performance, a vast search space of hyperparameter combinations must be explored, a process known as hyperparameter tuning [10]. AutoML [20] can automate and accelerate this exploration, but due to random errors or insufficient improvements, many trials are discarded. Consequently, accurate prediction of training job completion times remains a fundamental challenge in resource management systems. This unpredictability renders duration-dependent scheduling strategies (e.g., Shortest Remaining Time First) unreliable in practice—a key motivation for our duration-agnostic approach.

Heterogeneous Scale. Contemporary DL workloads increasingly adopt distributed parallelization strategies, including data, model, and

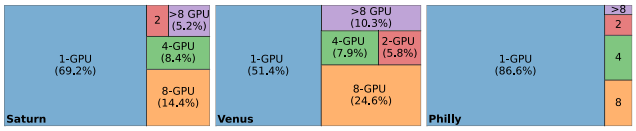

Fig. 2. Job scale distribution across three public GPU traces: Saturn, Venus, and Philly.

pipeline parallelism-to scale large models across multiple compute nodes. This distributed training paradigm introduces significant heterogeneity in computational scale, ranging from sGPU job to mGPU deployments, and spanning single-node to multi-node configurations. mGPU jobs are commonly used either because a model exceeds the memory or computational capacity of a single GPU, or to accelerate training through data-parallel execution.

2.2. GPU sharing

GPU sharing enables multiple jobs to utilize the same GPU resources, categorized into time-sharing and space-sharing. Time-sharing divides execution time, allowing jobs to alternate on the same GPU, but incurs overhead due to job switching [21]. In contrast, space-sharing partitions the GPU's spatial resources, enabling jobs to use different computational units simultaneously. While these units operate independently, shared components like caches can cause interference, as seen with NVIDIA's Multi-Process Service (MPS) [22]. Recent GPUs based on the Ampere architecture support Multi-Instance GPU (MIG) [23], which isolates shared components and reduces interference. However, MIG is limited to newer GPU generations, and dynamic repartitioning incurs non-trivial overhead since existing instances must typically be destroyed before new partitions can be created. Moreover, fixed hardware partitions may still lead to resource fragmentation and underutilization. To improve utilization, MPS can be used either independently or in combination with MIG to enable finer-grained sharing [24]. In both cases, interference between co-located jobs must be carefully managed, which is the focus of our research. While our method is validated on MPS, it is applicable to other space-sharing technologies for job scheduling and interference management.

2.3. Characteristics of DL clusters

Existing methods predominantly focus on symmetric packing, under which packing opportunities for mGPU jobs are more limited in heterogeneous-scale production environments. To illustrate this limitation, we analyze the job scale distributions across three clusters—Saturn, Venus, and Philly [14], where Saturn and Venus are virtual clusters within the Helios cluster [5]. As shown in Fig. 2, sGPU jobs dominate the workload, accounting for up to 86.6% in the Philly cluster. In this work, we focus on scenarios where at most two jobs are co-located on a GPU. We also examined three-job packing cases and found that they typically suffer from severe speed degradation, consistent with observations in prior work [12,25].

The distribution of mGPU jobs, however, is considerably more skewed. While sGPU jobs can be readily packed due to their high prevalence, mGPU jobs face significantly greater challenges. In particular, 8-GPU jobs exhibit the highest packing probability because of their relatively larger population, whereas 2-GPU jobs are much harder to pack, as their scarcity requires multiple such jobs to be present concurrently. As a result, mGPU jobs often fail to benefit from GPU-sharing schedulers and, in most cases, continue to execute in exclusive mode.

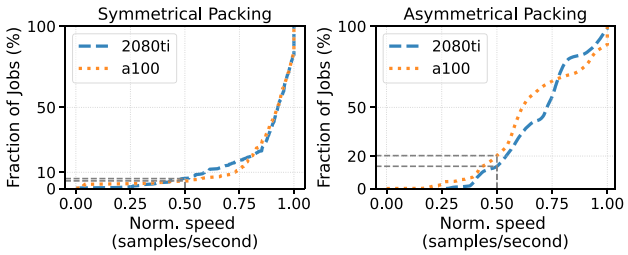

Fig. 3. Empirical CDFs of normalized job speed under symmetrical (left) and asymmetrical (right) packing on 2080ti and a100 GPUs.

2.4. Opportunities for packing DL jobs

The Potential of Asymmetric Packing. To assess the potential of asymmetric packing between sGPU and mGPU jobs, we conduct experiments on representative DNN workloads spanning multiple application domains, including image-to-image translation [26], reinforcement learning [2], neural machine translation [3,28], and image classification [1,31,33,35,37]. We adopt standard benchmark datasets from prior work [27,29,30,32,34,36]. These workloads exhibit diverse resource utilization characteristics, including GPU memory usage, GPU memory bandwidth intensity, communication intensity,[1] and SM utilization, as summarized in Table 1 and Table 6 for the 2080ti and a100 platforms, respectively. All experiments are implemented in PyTorch using Distributed Data Parallel (DDP) to support mGPU execution and are conducted on a testbed equipped with NVIDIA 2080ti and a100 GPUs. Detailed experimental settings are described in Section 6.1. Job performance under interference is quantified using speed, which captures throughput degradation due to co-location.

Under symmetric packing, as illustrated in Fig. 3, more than 90% of jobs retain over 0.5 speed, indicating limited performance degradation. However, symmetric packing offers limited packing opportunities for mGPU jobs, as analyzed in Section 2.3. Consequently, it fails to fully exploit workload scale heterogeneity. In contrast, asymmetric packing introduces more pronounced interference; nevertheless, over 80% of jobs still sustain speeds above 0.5. This observation suggests that, despite increased interference, asymmetric packing remains largely performance-tolerable for the majority of jobs. These results highlight the potential of asymmetric packing, while also underscoring the necessity of more careful interference management to guarantee sharing incentive.

Performance Degradation and Sharing Incentive. Most existing scheduling approaches rely on threshold-based heuristics, implicitly assuming that jobs with low interference characteristics consistently lead to beneficial packing outcomes. However, our analysis shows that average JCT depends jointly on job duration and the degree of inter-job interference, rather than on interference alone.

To illustrate this issue, we consider a simple packing scenario involving two jobs, denoted as Job A and Job B. The duration of Job A is $d_a = 100$, while Job B has a duration of $d_b = 90$. The post-packing execution speeds of the two jobs are denoted as v_a and v_b, respectively, capturing the impact of different levels of interference. The heatmap encodes the average JCT after packing under varying interference combinations. For ease of comparison, we define ΔJCT as the difference between the packed and exclusive execution cases, i.e., $\text{JCT}_{\text{packed}} - \text{JCT}_{\text{exclusive}}$. The results are illustrated in Fig. 4(left). Furthermore, beyond interference, we highlight that job duration also

[1] GPU memory bandwidth intensity is calculated as $\frac{T_{\text{memory}}}{T_{\text{total}}}$, and communication intensity is calculated as $\frac{T_{\text{communication}}}{T_{\text{total}}}$. Here, T_{memory} denotes the time during which the GPU memory subsystem is active, and $T_{\text{communication}}$ denotes the time spent on communication operations (e.g., all-reduce) in DDP training.

Table 1
Summary of the models and datasets used in our experiments, along with their profiled resource characteristics (GPU memory usage, GPU memory bandwidth intensity, SM utilization, and communication intensity for 4-GPU training) on the 2080ti.

Task	Model	Dataset	Batch Size	GPU Mem. Usage (GB)	GPU Mem. BW Int. (%)	SM Util. (%)	Comm. Int. (%)
✻	DCGAN[26]	LSUN[27]	32/64/128	2.8/2.9/3.7	28/33/35	93/94/95	46/33/22
✻	PPO[2]	LunarLander	32/64/128	0.7/0.7/0.7	0.1/0.1/0.1	11/11/11	5/4/1
◆	LSTM[28]	Wikitext2[29]	32/64/128	1.5/2.1/3.8	59/63/68	90/91/93	61/49/34
◆	Transformer[3]	Multi30k[30]	32/64/128	10.1/10.2/10.5	28/35/42	77/92/95	78/68/52
★	MobileNetV3[31]	ImageNet[32]	32/64/128	1.3/1.9/3.3	41/65/69	71/98/99	28/22/13
★	PointNet[33]	ShapeNet[34]	32/64/128	2.3/4.2/8.0	82/82/82	93/95/96	10/5/4
★	ResNet-18[35]	CIFAR-10[36]	32/64/128	3.4/3.7/4.5	58/64/68	88/93/94	53/46/32
★	VGG-11[1]	CIFAR-10[36]	32/64/128	2.1/2.1/2.2	48/41/34	76/61/54	64/57/44
★	PNASNetB[37]	CIFAR-10[36]	32/64/128	1.0/1.5/2.5	26/44/52	62/87/96	24/18/10

CV: ✻ Image-to-Image Translation ★ Image Classification NLP: ◆ Language Translation RL: ✻ Physics Control (Box2D).

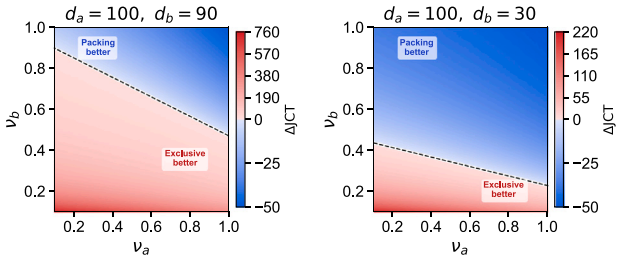

Fig. 4. Impact of job duration and interference on average JCT. *Left*: The durations of Job A and Job B are 100 and 90, respectively. *Right*: The durations of Job A and Job B are 100 and 30, respectively. Dashed lines indicate $\Delta\text{JCT} = 0$, where $\Delta\text{JCT} = \text{JCT}_{\text{packed}} - \text{JCT}_{\text{exclusive}}$.

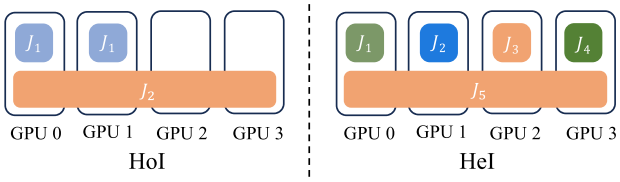

Fig. 5. Interference analysis illustration. Colored blocks denote different DL job types. *HoI*: multiple identical sGPU jobs co-located with one mGPU job. *HeI*: heterogeneous sGPU job types co-located with one mGPU job.

plays a critical role in determining the impact of packing on JCT. Fig. 4(right) presents the corresponding results for $d_a = 100$ and $d_b = 30$. We observe that for short-duration Job B, packing remains beneficial even when its speed drops to 0.3. In contrast, when Job B is long-running ($d_b = 90$), packing yields JCT improvements only if its speed exceeds 0.5.

While conservative threshold-based strategies – such as permitting packing only when jobs maintain execution speeds above 0.9 – offer strong guarantees for preserving sharing incentives, these overly restrictive criteria substantially limit co-location opportunities and, consequently, cap the efficiency gains achievable through GPU sharing.

3. Characterization of asymmetric packing

To characterize the interference effects arising from co-locating sGPU and mGPU DL jobs on shared GPUs, we conduct an extensive empirical analysis using the same workloads and experimental setup described in Section 2.4. This analysis quantifies how job performance changes under different co-location scenarios and provides the foundation for understanding interference patterns in asymmetric packing.

Interference Analysis Framework. We design two complementary experimental paradigms with controlled configurations, each isolating a distinct interference factor.

(1) *Homogeneous Interference (HoI) Analysis*: This analysis quantifies the impact of the number of co-located sGPU jobs on the performance of an mGPU job. As shown in Fig. 5(left), we vary the number of sGPU jobs while keeping the sGPU job type identical. This setup isolates interference effects attributable solely to packing density, excluding workload diversity.
(2) *Heterogeneous Interference (HeI) Analysis*: This analysis examines how different compositions of sGPU job types influence interference characteristics. As illustrated in Fig. 5(right), it captures the impact of workload heterogeneity on co-location performance, as well as how interference propagates among sGPU jobs through a co-located mGPU job.

3.1. Homogeneous interference

Fig. 6 illustrates how the performance of mGPU jobs varies with the degree of HoI induced by co-located sGPU jobs on 2080ti GPUs. The corresponding results on a100 GPUs are presented in Fig. 18 in Appendix.

Non-Linear Dependence of mGPU Performance on the Number of Colocated sGPU Jobs. As shown in Fig. 6(a)–(c), this non-linear behavior is consistently observed across different mGPU workloads. We random select three workloads – *DCGAN*, *ResNet18*, and *VGG* – to illustrate this phenomenon. Specifically, the marginal performance degradation introduced by each additional sGPU job is highly uneven. In some cases, co-locating the first sGPU job leads to a substantial speed drop, while subsequent sGPU jobs incur diminishing or irregular incremental impact. In contrast, other workloads remain relatively stable under light co-location and experience abrupt performance degradation only after the number of co-located sGPU jobs exceeds a threshold. These observations indicate that interference imposed on mGPU jobs cannot be captured by a linear or fixed per-job model. Instead, the number of co-located sGPU jobs fundamentally modulates interference intensity, making it a critical factor that must be explicitly incorporated into interference prediction models.

Jobs Exhibit Intrinsic Interference Impact. Fig. 6(d) reports results averaged across all mGPU workloads and varying numbers of co-located sGPU jobs, aiming to isolate whether sGPU job types exhibit intrinsic interference impact. We observe that *PointNet* consistently imposes the highest interference impact, followed by *LSTM*, while *PPO* exhibits the lowest impact. Notably, this ordering persists after averaging over heterogeneous mGPU workloads and different degrees of co-location, indicating that interference impact is an inherent property of the workload itself rather than an artifact of specific job pairings or hardware configurations(see Fig. 18(d)). Consequently, workloads

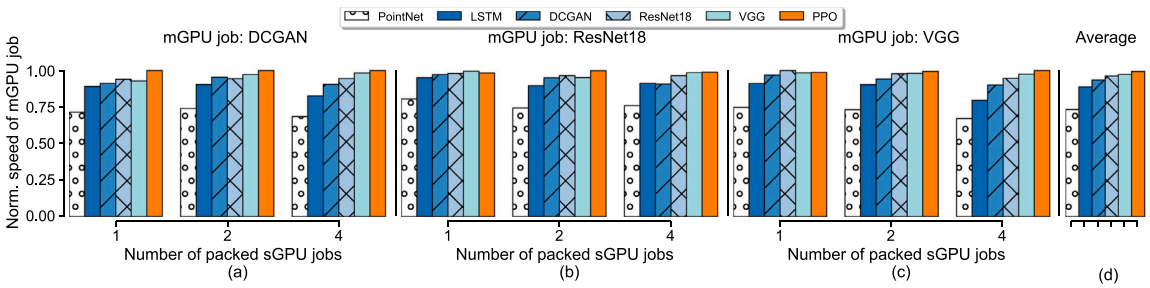

Fig. 6. HoI: Normalized speed of mGPU jobs running on a 4-GPU node under various co-location scenarios on the 2080ti. (a–c) depicts the normalized speed of a specific fixed mGPU workload (DCGAN, ResNet18, or VGG). Within each subfigure, the x-axis represents the number of co-located sGPU jobs from 1 to 4. Different colors bars indicate the specific type of sGPU job (e.g., PointNet, LSTM) being packed with the mGPU job. (d) reports the aggregate average normalized speed across all configurations.

with higher intrinsic interference impact are more likely to induce substantial performance degradation when co-located with other jobs.

The root cause lies in the distinct resource utilization patterns of these workloads. *PointNet* primarily consists of massively parallel point-wise operations with poor data locality, leading to severe contention for shared GPU resources under colocation. In contrast, *PPO* is comparatively CPU-bound, spending a significant fraction of its execution time interacting with the simulation environment to compute rewards and perform environment rollouts, thereby exerting limited pressure on shared GPU resources and resulting in minimal interference.

3.2. Heterogeneous interference

In the previous section, we fixed the sGPU job type and varied only the number of co-located jobs. However, real-world systems often co-locate jobs of different types. To capture the impact of workload diversity, we co-locate mGPU jobs with different combinations of sGPU job types. For clarity of presentation and analysis, we use the configuration of one mGPU job co-located with four distinct sGPU jobs as a representative example for visualization and discussion. This configuration induces substantial resource contention and interference diversity, thereby exposing both direct interference on the mGPU job and indirect interference propagation among sGPU jobs. Figs. 7 present the speed of these five jobs after being co-located on four GPUs, where the x-axis represents GPU IDs. Similarly, we conducted identical experiments on a100 GPUs, with results presented in Fig. 19 in Appendix. To further broaden the coverage of mGPU workload behaviors and validate the generality of our observations, we additionally present results from two groups of HeI experiments. The corresponding results are reported in Figs. 20 and 21 in Appendix.

Job-Level Interference Sensitivity under Fixed Interference. In the HeI setting, all sGPU jobs are co-located with the same mGPU job under an identical execution environment, ensuring exposure to the same interference sources and contention patterns. Despite these fixed interference conditions, Fig. 7 reveals pronounced disparities in performance degradation across different sGPU jobs. Specifically, in Fig. 7(a), the mGPU job (*DCGAN*) imposes an identical interference impact on all co-located sGPU jobs. However, the sGPU jobs exhibit markedly different speed degradation: *VGG* suffers the most severe slowdown, whereas *PPO* is minimally affected. Similar trend are consistently observed in Figs. 7(b)–(d), indicating that this phenomenon is robust across different job combinations.

These observations cannot be explained by interference impact alone, since all sGPU jobs are subject to the same external interference. Instead, the results expose an intrinsic, job-level property that governs how a workload responds to interference. We term this property *interference sensitivity*. In contrast to interference impact, which quantifies how much a job degrades the performance of others, interference sensitivity characterizes how strongly a job's own performance is affected when exposed to interference.

Consistent Job Performance under Similar Interference Impacts. We next examine whether jobs exhibit similar performance behavior when subjected to comparable interference conditions. To this end, we perform a pairwise analysis of Figs. 7(a)–(b) and Figs. 7(c)–(d), where the mGPU jobs impose similar levels of interference.

As shown in Figs. 7(a) and 7(b), the mGPU jobs *DCGAN* and *ResNet18* achieve nearly identical speed (0.80 and 0.79, respectively), indicating comparable interference impact on co-located sGPU jobs. Under these comparable interference conditions, sGPU jobs exhibit highly consistent performance degradation. For example, *PPO* achieves speed of 0.83 in Fig. 7(a) and 0.81 in Fig. 7(b), reflecting nearly identical slowdown behavior. Similar consistency is observed across other sGPU workloads, which demonstrate analogous patterns in the two settings.

A similar phenomenon is observed in Figs. 7(c) and 7(d). Although *DCGAN* and *ResNet-18* correspond to different model architectures, they exhibit similar interference characteristics when co-located with the same set of sGPU jobs, leading to comparable performance degradation. This similarity in degradation arises because these two mGPU jobs have comparable interference characteristics. We note that in Section 4.1 we describe how to quantify the interference characteristics of individual jobs, and in Section 4.2 we formally define how to measure the aggregate interference of a combination of co-located sGPU jobs.

Interference Propagation under Asymmetric Packing. We further demonstrate that under asymmetric packing, interference effects are not confined to isolated pairs of co-located jobs on the same GPU, but instead propagate across the entire packed job set. To isolate this effect, we compare Figs. 7(a) and 7(c), as well as Figs. 7(b) and 7(d). In each comparison, the sGPU composition differs only by replacing *PointNet* with *DCGAN*, while the remaining sGPU jobs and the mGPU job are kept identical. Despite this seemingly local change, we observe substantial performance variations across all jobs in the pack. Specifically, the impact is not limited to the newly introduced sGPU job: the speed of the other sGPU jobs, as well as that of the mGPU job, also changes markedly.

This observation indicates that interference under asymmetric packing is not a purely local phenomenon confined to contention on a single GPU. Instead, changes in sGPU composition reshape the global interference environment, affecting contention over shared resources such as memory bandwidth, cache hierarchy, and interconnects. Consequently, the interference experienced by each job is jointly determined by the composition of the entire packed workload set, rather than by any single co-located job in isolation.

4. Interference profiling and modeling

4.1. Job profiling

Prior work has explored performance-aware co-location through runtime monitoring and resource-based profiling. Gandiva [10] infers co-location feasibility by continuously monitoring fine-grained

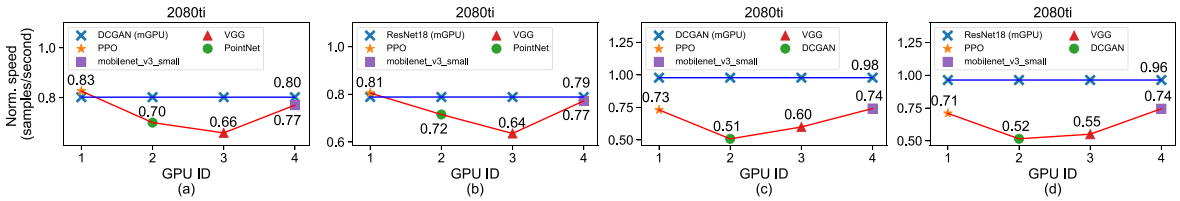

Fig. 7. HeI: (a,b) Normalized speed of mGPU and sGPU jobs composed of PPO, PointNet, VGG, MobileNet. (c,d) Normalized speed of mGPU and sGPU jobs composed of PPO, DCGAN, VGG, MobileNet. The x-axis denotes GPU IDs on 2080ti.

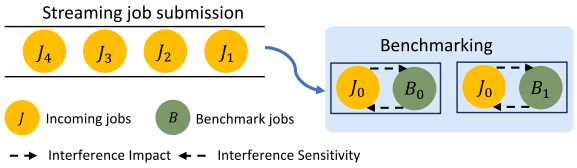

Fig. 8. Illustration of the job profiling procedure with two benchmark jobs.

CPU/GPU resource usage and estimating job progress via application-level feedback. Such feedback-driven approaches typically require several minutes and introduce non-negligible perturbations to running jobs. Lucid [12] and Horus [13] instead rely on coarse-grained GPU utilization metrics to characterize interference, enabling offline prediction of whether a job-pair can be safely co-located or the average slowdown of a pair. However, as shown in Section 3.2, asymmetric packing introduces pronounced *interference propagation*, where interference is no longer confined to a job pair but affects the entire packed job set. Under such conditions, pairwise feasibility prediction or aggregate slowdown estimation is insufficient. Accurate scheduling decisions require predicting the post-packing performance of each individual job.

In Concord, we address this limitation by profiling jobs using two high-level interference-oriented attributes: *interference impact* and *interference sensitivity*, rather than hardware-level resource metrics. Interference impact quantifies the extent to which a job degrades the performance of co-located jobs (outgoing interference), whereas interference sensitivity captures how susceptible a job is to performance degradation when exposed to external interference (incoming interference). Together, these two attributes form a dual characterization of a job's interference behavior under colocation.

To obtain these features, we preselect a set of benchmark jobs. Upon arrival, an incoming job is temporarily co-located with each benchmark job for a short profiling period. A DL training job consists of repeated mini-batch iterations; following prior work [10], the profiling duration should cover at least one mini-batch iteration. In our experiments, a 30-second profiling window is sufficient to obtain stable measurements. During profiling, the incoming job's *interference impact* is quantified as the speed of the benchmark job, while its *interference sensitivity* is measured as the speed of the incoming job itself. Fig. 8 illustrates the profiling procedure using two benchmark jobs. By co-locating the incoming job with multiple benchmark jobs, we construct its interference feature vector.

4.2. Interference prediction models

We next develop predictive models to estimate per-job speed under different packing patterns considered in this work. The key challenge lies in the fundamentally different interference behaviors of symmetric and asymmetric packing.

To systematically capture this diversity, we organize interference prediction models according to their interference behaviors along two orthogonal dimensions: (1) *interference symmetry*, which characterizes whether interference relations between co-located jobs are mutual and symmetric, and (2) *job scale heterogeneity*, which indicates whether the packing involves only sGPU jobs, only mGPU jobs, or a mixture of both. Together, these dimensions induce four canonical packing patterns, each exhibiting distinct interference mechanisms and thus necessitating a dedicated prediction strategy.

Table 2
Root Mean Square Error (RMSE) for speed prediction.

	Linear	SVM	XGBoost	Random Forest
RMSE	0.095	0.106	0.081	0.078

For each pattern, we construct a specialized predictor Φ that maps job-level interference features to speed. The notation $\Phi_{\text{sym/asym-s/m}}$ encodes both the interference symmetry and the predicted job type, enabling a clear correspondence between packing pattern and model selection. To ensure computational efficiency and enable real-time scheduling decisions, all predictive models are implemented using lightweight Random Forest models. Table 2 reports the prediction error of several lightweight models, including Linear Regression, SVM [38], XGBoost [39], and Random Forest [40].

Symmetric packing models. Such scenarios arise both for sGPU jobs and for mGPU jobs of equal scale.

For symmetric sGPU packing, where two sGPU jobs are co-located on the same device, we model speed as

$$v_{s_1}, v_{s_2} = \Phi_{\text{sym-s}}(\mathbf{f}_{s_1}, \mathbf{f}_{s_2}) \tag{1}$$

where $v_{s_i} \in (0, 1]$ denotes the speed of sGPU job s_i under co-location. Here, $\mathbf{f}_{s_1}$ and $\mathbf{f}_{s_2}$ denote the interference feature vectors of the two sGPU jobs, each encoding both interference impact and interference sensitivity.

For symmetric mGPU packing, symmetric interference persists but is compounded by synchronization overhead from collective communication. To capture this effect, we use $\Phi_{\text{sym-m}}$ to fit it:

$$v_{m_1}, v_{m_2} = \Phi_{\text{sym-m}}(\mathbf{f}_{m_1}, \mathbf{f}_{m_2}) \tag{2}$$

where $v_{m_i} \in (0, 1]$ denotes the speed of mGPU job m_i under co-location, and $\mathbf{f}_{m_1}$ and $\mathbf{f}_{m_2}$ are the corresponding interference feature vectors.

Asymmetric packing models. Asymmetric interference arises when an mGPU job is co-located with multiple sGPU jobs, introducing two key challenges. First, the number of interfering sGPU jobs varies, resulting in variable-length interference feature inputs. Second, interference dependencies become circular: the performance of sGPU jobs depends on the degraded execution state of the mGPU job, which in turn is determined by the aggregate interference imposed by all co-located sGPU jobs. To address these challenges, we adopt a two-stage prediction framework.

Stage I: mGPU speed prediction. Results from the HeI experiments (Section 3.2) show that mGPU performance degradation is largely similar when the compositions of co-located sGPU jobs are comparable. This motivates aggregating sGPU interference into a single vector:

$$v_m = \Phi_{\text{asym-m}}\left(\mathbf{f}_m, I_{\text{agg}}, \alpha, \beta\right), \tag{3}$$

where α denotes the GPU count allocated to the mGPU job, and β denotes the number of co-located sGPU jobs. The aggregated interference

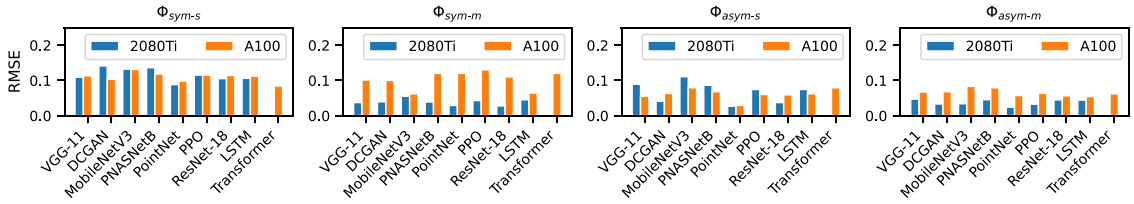

Fig. 9. RMSE of four interference prediction models under different job packing behaviors, using various benchmark jobs to construct interference features. The x-axis denotes the benchmark models, and the y-axis reports the prediction error.

representation is defined as

$$I_{\text{agg}} = \frac{1}{\beta}\sum_{i=1}^{\beta}\mathbf{f}_{s_i} \tag{4}$$

This aggregation compresses a variable-cardinality set of sGPU interference features into a fixed-dimensional representation.

Stage II: sGPU speed prediction.

HoI experiments (Section 3.1) further show that once the performance of the co-located mGPU job is determined, the speed of an sGPU job depends primarily on the mGPU job's performance and is largely independent of the specific composition of the other sGPU jobs. Accordingly, we predict each sGPU job independently as

$$v_{s_i} = \boldsymbol{\Phi}_{\text{asym-s}}(\mathbf{f}_{s_i}, v_m), \tag{5}$$

where $v_m \in (0, 1]$ denotes the speed of the mGPU job after packing. During training, v_m is obtained from measured historical data. At inference time, v_m is replaced by its predicted value, yielding a non-iterative cascade:

$$\hat{v}_{s_i} = \boldsymbol{\Phi}_{\text{asym-s}}\left(\mathbf{f}_{s_i}, \boldsymbol{\Phi}_{\text{asym-m}}(\mathbf{f}_m, I_{\text{agg}}, \alpha, \beta)\right), \tag{6}$$

which resolves circular dependencies.

4.3. Model validation

We evaluate the proposed interference prediction models using the experimental setup described in Section 6.1, with prediction error measured by the root mean squared error (RMSE). The prediction target is the normalized throughput (denoted as *speed* in this paper), which lies in [0, 1], making the RMSE dimensionless. Fig. 9 presents the RMSE of the four specialized models when different benchmark jobs are used to construct interference feature vectors. Based on these results, *VGG* is selected as the benchmark in this paper. Its prediction accuracy on both 2080ti and a100 is moderate – neither the best nor the worst – thus better reflecting the general performance of the proposed method.

Specifically, the $\boldsymbol{\Phi}_{sym-s}$ model exhibits the highest mean error, reaching 0.11 on 2080ti and 0.1 on a100. We attribute this to the characteristics of sGPU jobs, which are typically compute-intensive, making them highly sensitive to fine-grained resource contention (e.g., fluctuations in memory bandwidth and cache access latency). Such high-frequency and stochastic interference is inherently more difficult to capture using aggregated feature vectors. In contrast, communication contention dominates in the $\boldsymbol{\Phi}_{sym-m}$ setting, enabling more accurate prediction when communication-related effects are explicitly captured. This is particularly evident on 2080ti, where GPUs communicate over PCIe (32 GB/s), making communication bandwidth a primary bottleneck. On a100, NVLink (up to 300 GB/s) substantially alleviates communication contention, shifting the dominant source of interference toward fine-grained resource contention and resulting in prediction accuracy comparable to that of $\boldsymbol{\Phi}_{sym-s}$. Under asymmetric packing, increased system load further amplifies communication contention, leading to higher overall prediction accuracy compared to symmetric packing. Although $\boldsymbol{\Phi}_{asym-s}$ employs cascade prediction and may accumulate errors from $\boldsymbol{\Phi}_{asym-m}$, the RMSE remains below 0.1, indicating that error propagation is well controlled.

5. Design of concord

This section presents the three key components of Concord: (i) an offline profiler that characterizes jobs based on their interference characteristics, (ii) a scheduler that determines job priorities, and (iii) a placement manager that makes packing decisions while ensuring shared incentives.

5.1. Profiler

As described in Section 4, upon job arrival, Concord characterizes each job using two key features: interference impact and interference sensitivity. To extract these characteristics, the target job is briefly co-located with a set of benchmark jobs during a short profiling run. The resulting measurements are then used to predict the job's performance when packed with other jobs.

5.2. Scheduling

Due to heterogeneity in job resource demands and execution times, FIFO scheduling leads to poor resource utilization and head-of-line (HoL) blocking. While priority-based scheduling can improve efficiency, the challenge under asymmetric packing is that changes to a single job (e.g., preemption or reallocation) may cascade and disrupt all co-located jobs, amplifying scheduling overhead.

To address this, we generalize the least-attained-service (LAS) discipline [19] to account for packing compactness, yielding *Compact-Least-Attained-Service (C-LAS)*. C-LAS assigns each job a priority based on the amount of time it has been running so far and its packing density.

Formally, the priority of job j at time t is defined as:

$$p_j(t) = \sum_{i=0}^{t}\frac{g_j(i)}{c_j(i)}, \tag{7}$$

where $g_j(i)$ is the number of GPUs allocated to job j at time slot i, and $c_j(i)$ is the number of jobs co-located with j (including itself) at slot i. Jobs with smaller $p_j(t)$ are assigned higher scheduling priority, favoring those with less attained service and higher packing density.

Example. Consider a 4-GPU job co-located with sGPU jobs. When one sGPU job is packed, we have $g_j = 1$ and $c_j = 2$ for the sGPU job, yielding $p_j = 0.5$ per slot. When four sGPU jobs are packed, each has $g_j = 1$ but $c_j = 5$, yielding $p_j = 0.2$ per slot. Under C-LAS, the denser configuration (5 jobs) accumulates priority more slowly, making it less likely to be preempted. This preserves compact placements and minimizes cascading disruptions.

5.3. Incentive-aware placement

A key challenge in packing is ensuring that jobs benefit from sharing GPUs rather than waiting for exclusive access. We formalize this requirement as a *sharing incentive*: packing should not increase JCT compared to queuing for exclusive execution. This section establishes sufficient conditions for incentive-compatible placement and derives a practical packing policy independent of job duration estimates.

Problem Formulation. Consider a single GPU serving two jobs under FIFO arrival order. Job J_1 arrives at time t_1 with duration D_1 under exclusive access. Job J_2 arrives later at time $t_2 > t_1$ with duration D_2. At time t_2, job J_1 has remaining execution time $R_1 = D_1 - (t_2 - t_1)$. The scheduler must decide whether to pack J_2 with J_1 or to defer J_2 until J_1 completes.

Under packing, resource contention degrades each job's speed to a fraction $v_i \in (0, 1]$ of its exclusive-access speed. Our objective is to characterize conditions under which packing reduces the average JCT of the two jobs.

Analysis of Sharing Incentives. We analyze job completion time trade-offs across four canonical scenarios: three based on the duration ratio $\chi = D_2/R_1$ (short, comparable, and long jobs), and one capturing sustained packing as a worst-case scenario. For each scenario, we derive sufficient conditions under which both jobs complete earlier under packing than under sequential execution. Detailed proofs are provided in Appendix.

Proposition 1 (*Short Job Packing*). *If J_2 is sufficiently short relative to the remaining execution time of J_1 (i.e., $\chi \ll 1$), packing J_2 with J_1 is always beneficial, as the queuing delay saved by J_2 dominates any interference overhead.*

Proposition 2 (*Comparable-Duration Packing*). *When J_2's duration is comparable to J_1's remaining time ($\chi \leq 1$), packing is beneficial if and only if:*

$$\frac{1}{v_1} + \frac{1}{v_2} \leq \frac{1}{\chi} + 2. \tag{8}$$

Since $\chi \in (0, 1]$, the right-hand side is always at least 3.

Proposition 3 (*Long Job Packing*). *When J_2 exceeds J_1's remaining time ($\chi > 1$), packing is beneficial if and only if:*

$$\frac{1}{v_1} + \frac{1}{v_2} \leq 3. \tag{9}$$

Proposition 4 (*Sustained Packing*). *Consider a long-running job J_1 that has been continuously packed prior to time t_2 with average speed $\bar{v}_1$. The incentive compatibility of admitting a new job J_2 depends jointly on $\bar{v}_1, v_2$ and the duration ratio χ:*

- *If $\bar{v}_1 = 0.5$, packing is beneficial when $\frac{1}{v_1} + \frac{1}{v_2} < 3$.*
- *If $\bar{v}_1 < 0.5$ and $\chi \ll 1$, packing degrades J_1 excessively and should be avoided.*
- *If $\bar{v}_1 > 0.5$ and $\chi \ll 1$, packing is always beneficial.*

Across all four propositions, the packing conditions consistently depend on the sum of inverse speed, $\frac{1}{v_1} + \frac{1}{v_2}$. This term captures the aggregate slowdown experienced by both jobs. We therefore define it as the *Performance Degradation Metric (PDM)*:

$$\mathrm{PDM}(J_1, J_2) = \frac{1}{v_1} + \frac{1}{v_2}. \tag{10}$$

Propositions 1–3 show that PDM serves as a unifying indicator for sharing incentives, with a critical threshold of 3 emerging naturally from the analysis.

Placement Policy. Building on the theoretical results, we formulate a practical placement rule: jobs J_1 and J_2 are packed if and only if $\mathrm{PDM}(J_1, J_2) \leq \Theta$, where Θ is a system-wide parameter. As shown in Propositions 1–4, setting $\Theta = 3$ is sufficient to ensure sharing incentives across a wide range of representative scenarios. We refer to Θ as the *packing threshold* and adopt $\Theta = 3$ as the default value in our implementation. The impact of varying Θ on end-to-end scheduling performance is evaluated in Section 6.3.3.

For symmetric packing (packing two jobs), we directly apply the packing condition above. For asymmetric packing, we extend the condition as follows: (1) predict the speed v_i for each job under the candidate placement, (2) compute pairwise PDM between each sGPU job and the mGPU job, and (3) accept the placement if $\max_{J_s \in S} \mathrm{PDM}(J_s, J_m) \leq \Theta$, where S denotes the set of sGPU jobs and J_m denotes the mGPU job in the placement. This ensures that all packed jobs satisfy the sharing incentive. Algorithm 1 presents our *Incentive-Aware Packing Placement* procedure. For each candidate placement, we compute PDM and accept the placement if PDM $\leq \Theta$.

Algorithm 1 Incentive-Aware Packing Placement

Require: Job queue Q, GPU cluster G, packing threshold Θ
Ensure: Placement mapping $\mathcal{P} : Q \to G$

```
Initialize P ← ∅
for each job J ∈ Q do
    F ← ∅ {Feasible placements}
    for each candidate placement p ∈ G do
        if p is symmetric packing then
            Compute PDM(J, J') where J' is the packed job
            if PDM(J, J') ≤ Θ then
                F ← F ∪ {p}
            end if
        else if p is asymmetric packing then
            Predict v_i for all jobs in p
            Compute PDM_max = max_{J_s ∈ S} PDM(J_s, J_m)
            if PDM_max ≤ Θ then
                F ← F ∪ {p}
            end if
        end if
    end for
    if F ≠ ∅ then
        Select p* = arg min_{p ∈ F} PDM(J, p)
        P ← P ∪ {(J, p*)}
    else
        Defer J for next scheduling round
    end if
end for
return P
```

To reduce search complexity, we impose two practical constraints:

- Single-node packing: Restrict placements to GPUs within a single node. This is justified by cluster trace analysis showing that 90%+ jobs are single-node workloads (94.7% in Saturn, 89.6% in Venus, 98.9% in Philly), as analyzed in Section 2.3.
- Consolidated multi-node placement: For jobs requiring multiple nodes, use consolidated allocation to minimize resource fragmentation.

6. Evaluation

This section presents a comprehensive evaluation of Concord through a series of experiments employing a diverse set of representative workloads (Table 1). Our evaluation seeks to answer the following questions:

- Effectiveness: Can Concord guarantee non-degraded JCT compared to SOTA schedulers under high-density asymmetric packing scenarios?
- Utilization: To what extent does asymmetric packing improve job concurrency and streaming multiprocessor (SM) utilization?
- Sensitivity: How do the packing threshold (Θ), job distribution characteristics, and system load intensity impact Concord's end-to-end scheduling performance?

Table 3
99th percentile queuing delay (s) across different scheduling approaches, clusters, and GPUs.

	2080ti		a100	
	Venus	Philly	Venus	Philly
FIFO	613 475	119 695	582 705	120 228
SRTF	17 755	25 426	11 669	25 025
QSSF	68 635	66 889	70 980	63 632
Tiresias	102 153	69 994	64 942	64 646
Lucid	43 449	4235	14 330	**0**
Concord	**2042**	**0**	**1593**	515

6.1. Experimental setup

Implementation. We developed a discrete-time cluster simulator based on a high-fidelity simulation framework [12]. Experimental data were collected on two servers: (i) RTX 2080ti (11 GB) GPUs connected via PCIe Gen4 (32 GB/s) with two Intel Xeon Gold 5115 CPUs (40 threads); and (ii) NVIDIA a100 (80 GB) GPUs connected via NVLink (300 GB/s) with two Intel Xeon Gold 6354 CPUs (72 threads). A total of 23,869 data points were collected, excluding packing configurations that exceeded total GPU memory. These measurements, including job speeds under packing, SM utilization, and memory, are used as inputs to the simulator.

Traces. Our experimental evaluate using two real production level traces from SenseTime [5] and Microsoft [14]. The SenseTime traces are from September, while the Microsoft trace covers the first week of October, consistent with the data period used in Lucid. For SenseTime traces, we used models from Table 1, randomly assigning them to workloads not in the trace. For the Philly trace, workloads were randomly assigned to jobs.

Baselines. Prior work on GPU sharing falls into two categories that are fundamentally incompatible with our evaluation framework: (1) *intrusive hardware-level solutions* such as Orion [21] and IADeep [41], which require kernel modifications or specialized hardware unavailable in commodity clusters, and (2) *time-sharing schedulers* such as Horus [13] and Gandiva [10], which multiplex GPUs temporally through context switching rather than spatial co-location.

To systematically evaluate the benefits of Concord's key design choices – duration-agnostic scheduling, GPU sharing, and asymmetric packing – we compare against representative baselines that isolate each dimension:

1. **First-In-First-Out (FIFO)** [7,8]: The default scheduler in production systems such as YARN and Kubernetes, allocating GPUs exclusively to jobs in arrival order without considering job characteristics or durations.
2. **Shortest-Remaining-Time-First (SRTF)** [42]: An optimal duration-aware policy for exclusive allocation that prioritizes jobs with the least remaining execution time, providing a theoretical upper bound for non-sharing schedulers.
3. **Quasi-Shortest-Service-First (QSSF)** [5]: A duration-aware scheduler that predicts and prioritizes shorter jobs under exclusive allocation to minimize average JCT.
4. **Tiresias** [19]: A duration-agnostic scheduler employing least-attained-service policy with preemption support, representing the state-of-the-art for exclusive allocation without requiring job duration information.
5. **Lucid** [12]: A duration-aware GPU-sharing scheduler that enables symmetric packing through data-driven interference prediction, supporting only co-location of same-scale jobs in a non-preemptive manner.

Baseline algorithms require job duration estimates for optimal performance. To ensure fair comparison, we provide baselines with oracle durations, while Concord operates without. Despite this disadvantage, Concord achieves competitive performance against SOTA algorithms.

Table 4
Evaluation of Concord on large (>8 GPUs) and small-scale (≤8 GPUs) jobs in Venus cluster in 2080ti.

	Avg. JCT (s)		P99 Queue (s)	
	Lucid	Concord	Lucid	Concord
Large-Scale	22 620	18 012	117 753	24 045
Small-Scale	14 628	13 552	33 868	738

6.2. Overall performance comparison

6.2.1. JCT

Fig. 10 compares the average JCT across scheduling algorithms. Concord consistently achieves the lowest JCT in all configurations, reducing JCT by 1%–10% relative to all baselines, including Lucid with job duration priors. This improvement stems from Concord's incentive-aware packing strategy, which effectively identifies asymmetric packing opportunities that significantly mitigate queuing delays. Notably, Concord also outperforms SRTF, which is optimal in non-sharing settings, indicating that effective GPU sharing can surpass traditional optimal policies. The gains are more pronounced on the Venus trace, where higher cluster load prolongs queuing delays and thus creates more opportunities for asymmetric packing. These improvements remain consistent across both 2080ti and a100 GPUs.

6.2.2. P99 queuing delay

Most scheduling algorithms emphasize average performance, often neglecting worst-case user experience. Table 3 reports the 99th percentile queuing delay for each algorithm. Concord achieves lower delays in most cases, with improvements ranging from 88% to 100%. The only exception is in the Philly cluster on a100 GPUs, where delays remain below 10 min.

6.2.3. Concurrency

Fig. 11 shows the distribution of job concurrency. Under high-load conditions (Venus trace), Concord achieves 1.44–1.46× higher concurrency than non-sharing schedulers. Even under lower load (Philly trace), Concord sustains a 1.24–1.27× concurrency improvement. The improvement is mainly due to the increased opportunities for packing. When cluster load is high, jobs experience longer queueing delays, providing more opportunities to co-locate jobs.

6.2.4. Utilization & makespan

GPU streaming multiprocessor (SM) utilization is a key indicator of system efficiency. As shown in Fig. 12, Concord improves SM utilization by 13%–26% over Lucid and by 17%–31% over SRTF. We further evaluate overall workload completion using makespan. On the high-load Venus trace, Concord reduces makespan by up to 9 days compared to Lucid, while achieving smaller yet consistent improvements on the low-load Philly trace. This SM utilization improvement arises from Concord's asymmetric packing, which allows more mGPU jobs to share GPU resources concurrently with sGPU jobs. As a result, more jobs execute in parallel rather than waiting in the queue. Makespan, which measures the completion time of the last finishing job, is highly sensitive to queuing delays. By increasing concurrent execution through asymmetric packing, Concord reduces queuing time for mGPU jobs, yielding shorter makespan.

6.3. Sensitivity analysis

6.3.1. Sensitivity to job size

Table 4 shows that Concord consistently outperforms Lucid for both large and small jobs, demonstrating that large jobs do not suffer starvation under Concord's scheduling.

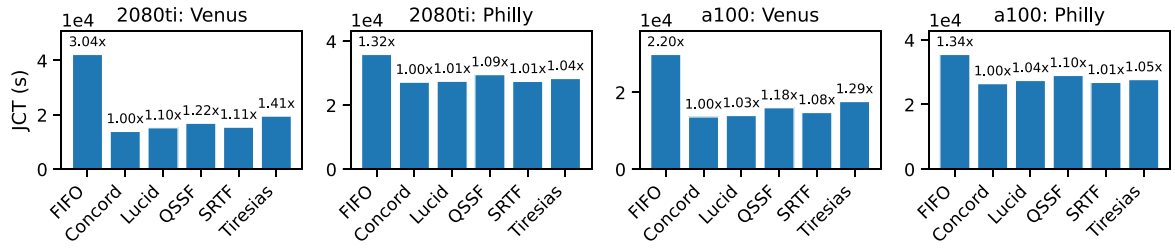

Fig. 10. JCT comparison of different scheduling algorithms on Venus and Philly clusters using 2080ti and a100. Numbers above the bars indicate the relative JCT with respect to Concord.

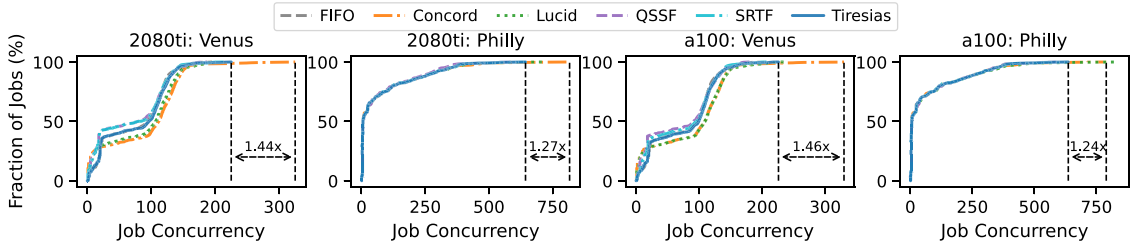

Fig. 11. CDF of job concurrency under different scheduling algorithms on the Venus and Philly clusters usingd 2080ti and a100 GPUs.

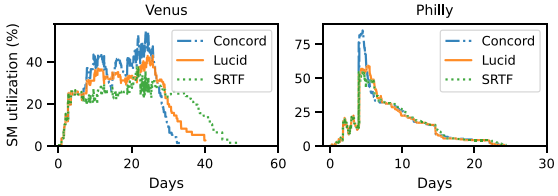

Fig. 12. GPU SM utilization over time on the Venus and Philly clusters under Concord, Lucid, and SRTF, evaluated on 2080ti.

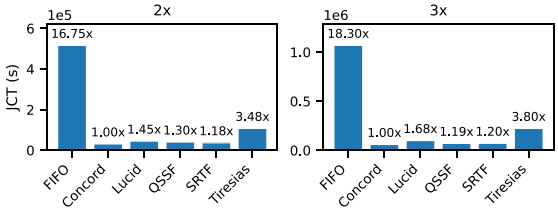

Fig. 13. JCT comparison for various scheduling algorithms across Venus clusters on 2080ti under 2× and 3× workload intensities.

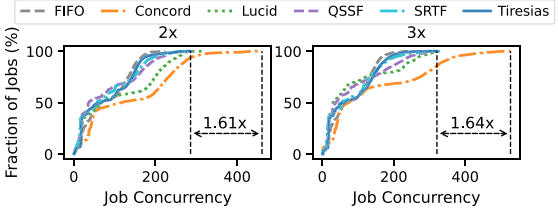

Fig. 14. CDF of job concurrency under different scheduling algorithms on the Venus clusters, evaluated on 2080ti with 2× and 3× workload intensities.

6.3.2. *Impact of workload intensity*

As discussed earlier, the performance of different schedulers varies across the Venus and Philly traces due to their different workload intensities. To further study how workload intensity affects scheduling performance, we construct two additional traces by resampling the original job arrivals, resulting in traces with 2× and 3× the workload intensity of the original trace.

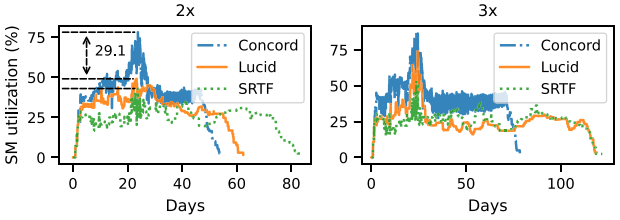

Fig. 15. GPU SM utilization over time on the Venus cluster under increased workload intensities (2× and 3× of the original trace) for Concord, Lucid, and SRTF on the 2080ti.

Fig. 13 reports the average JCT under different workload intensities. As the system load increases, the performance gap between Concord and Lucid becomes more pronounced. Under higher intensities, Concord reduces JCT by 45%–68% compared to Lucid.

We also examine how workload intensity affects system concurrency and resource efficiency. As shown in Fig. 14, Concord achieves up to 1.64× higher concurrency than Lucid. Furthermore, Fig. 15 shows the GPU SM utilization over time under different workload intensities. Concord consistently maintains higher SM utilization, achieving up to a 29% improvement compared to the baselines.

6.3.3. *Impact of packing threshold*

The packing threshold Θ directly influences JCT. We assess the robustness of the analytically derived threshold ($\Theta = 3$) from Section 5.3 across varying fractions of high-interference jobs (Fig. 16). We observe that $\Theta = 3$ consistently delivers near-optimal JCT across diverse packing regimes, demonstrating both the robustness of the proposed model and the validity of the derived threshold.

6.4. *Asymmetrical packing evaluation*

Next, we evaluate the effectiveness of asymmetric packing. Since asymmetric packing expands opportunities by co-locating sGPU with mGPU jobs, its benefit is inherently tied to the sGPU job fraction in the workload. We compare asymmetric and symmetric packing under varying sGPU proportions. Fig. 17 shows that the advantage of asymmetric packing becomes increasingly pronounced as the sGPU

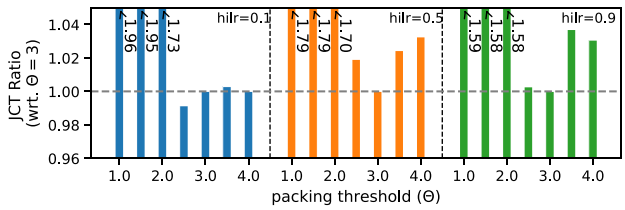

Fig. 16. Performance of Concord in terms of average JCT (normalized to $\Theta = 3$) under different Θ and various high-interference load ratios (hilr). The x-axis represents different values of the packing threshold Θ.

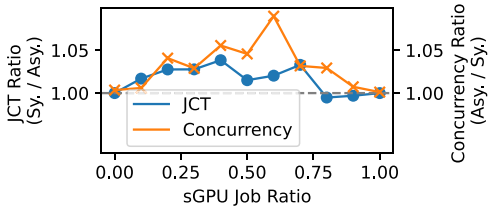

Fig. 17. JCT and Concurrency jobs Comparison of *Asymmetrical packing* and *Symmetrical packing* placement policies.

Table 5
Computational Overhead of Predictive Models.

Metric	Φ_{sym-s}	Φ_{sym-m}	Φ_{asym-s}	Φ_{asym-m}
Training (s)	0.0466	0.055	0.049	3.6
Inference (ms)	0.0934	0.13	0.3	0.0024
Memory (KB)	16.66	22.8	19.5	1.74

proportion grows. When sGPU jobs account for 40% of the workload, asymmetric packing reduces JCT by up to 4% and improves job concurrency by 6%.

6.5. Interference profiling overhead

Finally, we evaluate the time and space complexity of the four models during the profiling process, using VGG-11 as the benchmark. Table 5 reports the profiling overhead for different models. The prediction model calculates PDM within 0.3 s with minimal memory overhead (around 20KB), indicating that Concord's profiling process has negligible impact on system performance.

7. Related works

DL Job Schedulers. Recent research has primarily focused on optimizing average JCT and reducing queuing delays in DL clusters through various GPU sharing mechanisms [11–13]. Time-sharing approaches such as Gandiva [10] and Horus [13] achieve multiplexing through temporal partitioning and context switching, while intrusive hardware-level solutions like Orion [21] and IADeep [41] employ kernel modifications or specialized hardware for fine-grained sharing. Although effective, these approaches incur significant overhead from context switching or require invasive modifications incompatible with commodity clusters. Bless [43] further proposes a bubbleless spatial–temporal sharing system that exploits idle GPU bubbles to reduce co-located application latency, yet relies on fine-grained kernel-level scheduling that remains intrusive and hardware-specific. Non-intrusive spatial sharing methods like Lucid [12] improve GPU utilization by packing jobs with predicted performance degradation below empirical thresholds (e.g., 20% slowdown) [13,44]. However, such packing strategies remain heuristic-based, lack theoretical justification for sharing incentives, and are limited to symmetric packing – co-locating only same-scale jobs – which constrains packing opportunities in heterogeneous production workloads. Mudi [45] extends this direction to SLO-aware multiplexing of inference and training workloads, but optimizes training throughput rather than cluster-wide JCT. Some works focus on container-level GPU virtualization without JCT optimization [46,47], while others rely on job duration prediction for scheduling [48]. In contrast, our work provides JCT guarantees without requiring duration.

Interference estimation. Several techniques model interference and predict performance degradation in shared GPU environments. For example, Bubble Up [9] and Bubble-Flux [17] model application cache sensitivity. Themis [18] predicts job slowdown based on event statistics, and Gandiva [10] employs online profiling for job colocation. However, runtime profiling of DL job kernels to infer interference and create performance profiles can significantly extend training times, from minutes to hours. To address this, Horus [13] converts models into ONNX [49] graphs and extracts workload features. However, this approach requires exposing model structures.

8. Conclusion and future work

In this paper, we present Concord, a GPU cluster scheduler designed to efficiently packing mGPU and sGPU jobs by managing workload interference. Unlike prior approaches that rely on runtime hardware metrics or job duration estimates, Concord employs lightweight offline profiling to characterize jobs based on their interference impact and sensitivity. By integrating an incentive-aware packing placement strategy with theoretically-grounded sharing incentives and fine-grained interference prediction models, Concord enables asymmetric packing while guaranteeing non-degraded JCT. Experimental results using real-world cluster traces demonstrate that Concord achieves a 1.68× reduction in average JCT and up to 29% improvement in GPU SM utilization compared to state-of-the-art schedulers.

Looking ahead, we plan to extend this approach to LLM [50] deployment, where dynamic execution patterns and shifting resource bottlenecks will require richer profiling and more adaptive interference models. Additionally, Concord's coarse-grained GPU memory capacity model could be strengthened by fragmentation-aware allocation to improve robustness under high consolidation.

CRediT authorship contribution statement

Xinhua Wang: Writing – original draft, Software, Methodology, Formal analysis. **Weiwei Lin:** Writing – review & editing, Supervision, Funding acquisition. **Haijie Wu:** Writing – review & editing, Investigation. **Jidong Zhai:** Writing– review & editing, Supervision. **Keqin Li:** Writing– review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is supported by Guangxi Key Research and Development Project(2024AB02018), Guangdong Provincial Natural Science Foundation Project (2025A1515010113), Shandong Provincial Natural Science Foundation Project (ZR2024LZH012), and the Major Key Project of PCL, China under Grant PCL2025A08 and PCL2025A11. Weiwei Lin is the corresponding author.

Appendix. Proof of the sharing-incentive packing strategy

This appendix establishes sufficient conditions under which GPU sharing improves JCT compared to exclusive execution. We consider a two-job packing scenario and derive conditions that lead to Propositions 1–4.

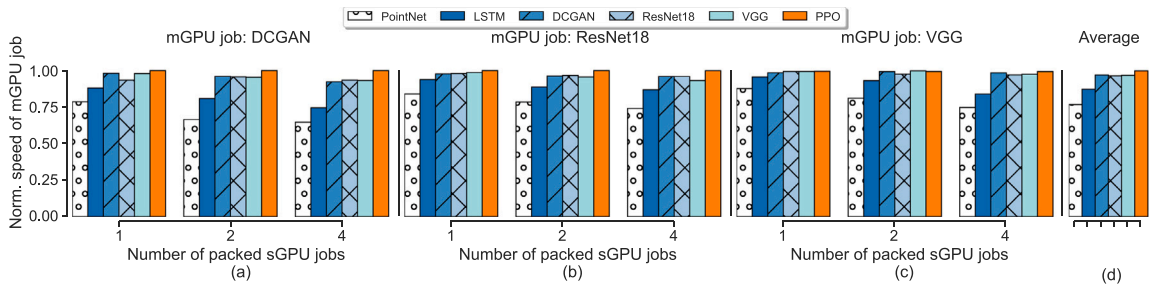

Fig. 18. HoI: Normalized speed of mGPU jobs running on a 4-GPU node under various co-location scenarios on the a100. (a–c) depicts the normalized speed of a specific fixed mGPU workload (DCGAN, ResNet18, or VGG). Within each subfigure, the x-axis represents the number of co-located sGPU jobs from 1 to 4. Different colors bars indicate the specific type of sGPU job (e.g., PointNet, LSTM) being packed with the mGPU job. (d) reports the aggregate average normalized speed across all configurations.

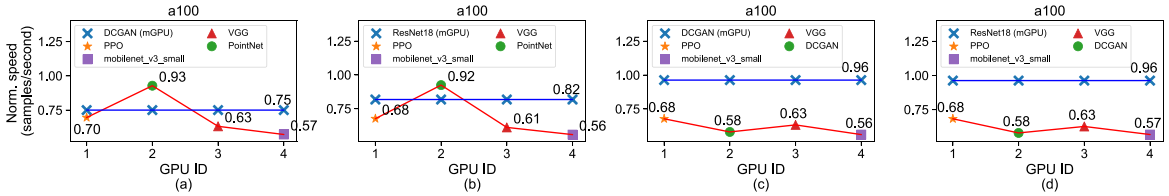

Fig. 19. HeI: (a,b) Normalized speed of mGPU and sGPU jobs composed of PPO, PointNet, VGG, MobileNet. (c,d) Normalized speed of mGPU and sGPU jobs composed of PPO, DCGAN, VGG, MobileNet. The x-axis denotes GPU IDs on a100.

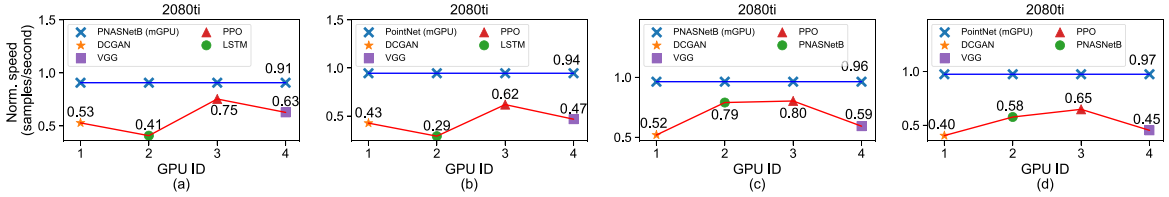

Fig. 20. HeI: (a,b) Normalized speed of mGPU and sGPU jobs composed of PPO, DCGAN, VGG, LSTM. (c,d) Normalized speed of mGPU and sGPU jobs composed of PPO, DCGAN, VGG, PNASNetB. The x-axis denotes GPU IDs on 2080ti.

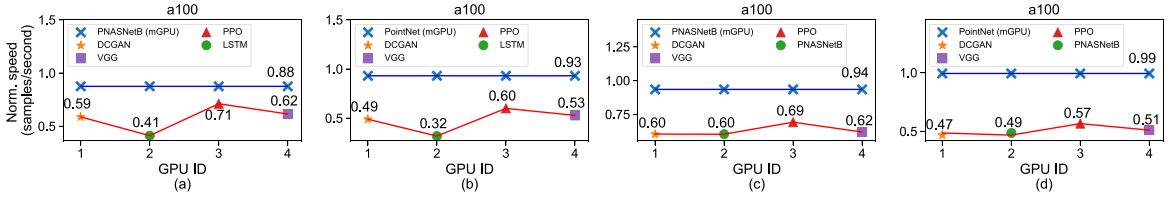

Fig. 21. HeI: (a,b) Normalized speed of mGPU and sGPU jobs composed of PPO, DCGAN, VGG, LSTM. (c,d) Normalized speed of mGPU and sGPU jobs composed of PPO, DCGAN, VGG, PNASNetB. The x-axis denotes GPU IDs on a100.

A.1. System model and baseline

We consider a cluster with unit processing capacity. Two jobs, denoted by J_1 and J_2, arrive at times $t_1 = 0$ and $t_2 > 0$, with standalone execution times D_1 and D_2, respectively. Under GPU sharing, the normalized throughputs of the two jobs are $v_1, v_2 \in (0, 1]$.

Our goal is to compare the average JCT under: (i) *exclusive execution*, where jobs execute sequentially, and (ii) *GPU sharing*, where the two jobs are colocated after t_2.

A.2. Exclusive execution

If $t_2 \geq D_1$, the two jobs do not overlap and both placements are equivalent:

$$JCT_{\text{exclusive}} = \frac{D_1 + D_2}{2}. \tag{11}$$

When $t_2 < D_1$, job J_2 waits until J_1 completes. The resulting average JCT is

$$JCT_{\text{exclusive}} = \frac{2D_1 - t_2 + D_2}{2}. \tag{12}$$

A.3. GPU sharing execution

When $t_2 < D_1$, the two jobs are colocated after t_2. Let $R_1 = D_1 - t_2$ denote the remaining execution time of J_1 at t_2.

We distinguish two cases based on the relative duration of J_2.

A.3.1. Case I: $D_2 \geq R_1$ (comparable or long job)

Both jobs execute concurrently until J_1 completes. The completion times are:

$$J_1 = t_2 + \frac{R_1}{v_1}, \qquad J_2 = \frac{R_1}{v_2} + D_2 - R_1. \tag{13}$$

Table 6
Summary of the models and datasets used in our experiments, along with their profiled resource characteristics (GPU memory usage, GPU memory bandwidth intensity, SM utilization, and communication intensity for 4-GPU training) on the a100.

Task	Model	Dataset	Batch Size	DRAM Usage (GB)	DRAM BW Int. (%)	SM Util. (%)	Comm. Int. (%)
❋	DCGAN[26]	LSUN[27]	32/64/128	3.0/3.3/3.5	10/15/19	41/50/45	45/38/34
❋	PPO[2]	LunarLander	32/64/128	0.5/0.5/0.5	0.1/0.1/0.1	17/18/19	1/3/3
◆	LSTM[28]	Wikitext2[29]	32/64/128	1.4/2.1/3.5	39/45/49	92/94/95	60/52/40
◆	Transformer[3]	Multi30k[30]	32/64/128	9.9/10.1/10.3	4/3/6	84/91/87	41/49/17
★	MobileNetV3[31]	ImageNet[32]	32/64/128	1.6/2.2/3.5	11/24/38	51/82/93	26/21/16
★	PointNet[33]	ShapeNet[34]	32/64/128	2.7/4.6/8.5	47/50/51	54/54/55	14/12/3
★	ResNet-18[35]	CIFAR-10[36]	32/64/128	2.3/3.6/4.0	14/26/41	49/66/79	51/49/42
★	VGG-11[1]	CIFAR-10[36]	32/64/128	2.6/3.4/3.6	10/9/10	38/35/34	66/59/46
★	PNASNetB[37]	CIFAR-10[36]	32/64/128	1.3/1.7/2.9	5/13/23	45/70/93	27/23/21

CV: ❋ Image-to-Image Translation ★ Image Classification NLP: ◆ Language Translation RL: ❋ Physics Control (Box2D).

The average JCT under sharing is therefore

$$JCT_{\text{share}} = \frac{1}{2}\left[\left(\frac{1}{\nu_1}+\frac{1}{\nu_2}-1\right)R_1 + t_2 + D_2\right]. \tag{14}$$

Comparing with Eq. (12), GPU sharing is beneficial if and only if

$$\frac{1}{\nu_1}+\frac{1}{\nu_2} \le 3. \tag{15}$$

When $D_2 \le R_1$ (i.e., $\chi = D_2/R_1 \le 1$), this establishes Proposition 2. When $D_2 > R_1$ (i.e., $\chi > 1$), the same condition applies and establishes Proposition 3.

A.3.2. Case II: $D_2 < R_1$ (short job)

Job J_2 completes before J_1 finishes. The completion times are

$$J_1 = D_1 + \frac{D_2}{\nu_1} - D_2, \qquad J_2 = \frac{D_2}{\nu_2}. \tag{16}$$

The resulting average JCT is

$$JCT_{\text{share}} = \frac{1}{2}\left[\left(\frac{1}{\nu_1}+\frac{1}{\nu_2}-1\right)D_2 + D_1\right]. \tag{17}$$

Let $D_2 = \chi R_1$ with $\chi \in (0,1)$. Sharing improves JCT if

$$\frac{1}{\nu_1}+\frac{1}{\nu_2} \le \frac{1}{\chi}+2, \tag{18}$$

which directly implies Proposition 1. In the limit $\chi \ll 1$, the condition is always satisfied.

A.4. Worst-case sustained packing

We now consider a worst-case scenario in which J_1 has been continuously packed with other jobs prior to t_2. Let $\iota \in (0,1)$ denote the average normalized throughput of J_1 before t_2, and R_1 its remaining execution time at t_2.

A.4.1. Exclusive execution

Under exclusive execution after t_2, the completion times are

$$J_1 = \frac{D_1 - R_1}{\iota} + R_1, \qquad J_2 = R_1 + D_2. \tag{19}$$

A.4.2. GPU sharing

When J_2 is packed with J_1 at t_2, the completion times become

$$J_1 = \frac{D_1 - R_1}{\iota} + \frac{R_1 - D_2}{\iota} + \frac{D_2}{\nu_1}, \qquad J_2 = \frac{D_2}{\nu_2}. \tag{20}$$

Sharing is beneficial if $JCT_{\text{share}} \le JCT_{\text{exclusive}}$, which yields the condition

$$\frac{1}{\nu_1}+\frac{1}{\nu_2} \le 1 + \frac{1}{\iota} + \frac{R_1}{D_2}\left(2-\frac{1}{\iota}\right), \quad 0 < \iota < 1. \tag{21}$$

This establishes Proposition 4 and completes the proof.

Data availability

Data will be made available on request.

References

[1] Karen Simonyan, Andrew Zisserman, Very deep convolutional networks for large-scale image recognition, Int. Conf. Learn. Represent. (2015).
[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Proximal policy optimization algorithms, 2017, ArXiv Preprint.
[3] A. Vaswani, Attention is all you need, Adv. Neural Inf. Process. Syst. (2017).
[4] Simon Gathu, et al., High-performance computing and big data: Emerging trends in advanced computing systems for data-intensive applications, J. Adv. Comput. Syst. (2024).
[5] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, Characterization and prediction of deep learning workloads in large-scale gpu datacenters, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021.
[6] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Mlaas in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters, in: USENIX Symposium on Networked Systems Design and Implementation, 2022.
[7] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, John Wilkes, Borg, omega, and kubernetes: Lessons learned from three container-management systems over a decade, Queue (2016).
[8] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Apache hadoop YARN: yet another resource negotiator, in: Proceedings of the Annual Symposium on Cloud Computing, 2013.
[9] Jaeung Han, Seungheun Jeon, Young-ri Choi, Jaehyuk Huh, Interference management for distributed parallel applications in consolidated clusters, ACM SIGPLAN Not. (2016).
[10] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, Fan Yang, Gandiva: Introspective cluster scheduling for deep learning, in: USENIX Symposium on Operating Systems Design and Implementation, 2018.
[11] Ting-An Yeh, Hung-Hsin Chen, Jerry Chou, KubeShare: A framework to manage GPUs as first-class and shared resources in container cloud, in: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020.
[12] Qinghao Hu, Meng Zhang, Peng Sun, Yonggang Wen, Tianwei Zhang, Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs, in: Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2023.
[13] Gingfung Yeung, Damian Borowiec, Horus: Interference-aware and prediction-based scheduling in deep learning systems, IEEE Trans. Parallel Distrib. Syst. (2021).
[14] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Analysis of large-scale multi-tenant GPU clusters for DNN training workloads, in: USENIX Annual Technical Conference, 2019.
[15] Christina Delimitrou, Christos Kozyrakis, Paragon: Qos-aware scheduling for heterogeneous datacenters, in: Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2013.
[16] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations, in: Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture, 2011.

[17] Hailong Yang, Alex Breslow, Jason Mars, Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers, ACM SIGARCH Comput. Archit. News (2013).
[18] Wenyi Zhao, Quan Chen, Hao Lin, Jianfeng Zhang, Jingwen Leng, Chao Li, Themis: Predicting and reining in application-level slowdown on spatial multitasking GPUs, in: IEEE International Parallel and Distributed Processing Symposium, IEEE, 2019.
[19] Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Tiresias: A GPU cluster manager for distributed deep learning, in: USENIX Symposium on Networked Systems Design and Implementation, 2019.
[20] AutoML, AutoML. http://www.ml4aad.org/automl/.
[21] Foteini Strati, Xianzhe Ma, Ana Klimovic, Orion: Interference-aware, fine-grained GPU sharing for ml applications, in: Proceedings of the Nineteenth European Conference on Computer Systems, 2024.
[22] NVIDIA, Multi-Process Service. https://docs.nvidia.com/deploy/mps.
[23] NVIDIA, Multi-Instance GPU. https://www.nvidia.com/en-us/technologies/multi-instance-gpu/.
[24] Bowen Zhang, Yuhang Wang, Zhuozhao Li, BOER: Enhancing resource utilization for deep learning inference with hybrid spatial GPU sharing, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2025.
[25] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, Matei Zaharia, Heterogeneity-aware cluster scheduling policies for deep learning workloads, in: USENIX Symposium on Operating Systems Design and Implementation, 2020.
[26] Alec Radford, Luke Metz, Unsupervised representation learning with deep convolutional generative adversarial networks, Int. Conf. Learn. Represent. (2016).
[27] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, Jianxiong Xiao, Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop, 2015, ArXiv Preprint.
[28] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, Neural machine translation by jointly learning to align and translate, Int. Conf. Learn. Represent. (2015).
[29] Stephen Merity, Caiming Xiong, Pointer sentinel mixture models, Int. Conf. Learn. Represent. (2017).
[30] Desmond Elliott, Stella Frank, Khalil Sima'an, Lucia Specia, Multi30k: Multilingual english-german image descriptions, In Proc. the 5th Work. Vis. Lang. (2016).
[31] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Searching for mobilenetv3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019.
[32] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, Li Fei-Fei, Imagenet: A large-scale hierarchical image database, in: IEEE Conference on Computer Vision and Pattern Recognition, 2009.
[33] Charles R. Qi, Hao Su, Pointnet: Deep learning on point sets for 3d classification and segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017.
[34] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Shapenet: An information-rich 3d model repository, 2015, arXiv preprint arXiv:1512.03012.
[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.
[36] Alex Krizhevsky, The CIFAR10 Dataset. https://www.cs.toronto.edu.
[37] Chenxi Liu, Barret Zoph, Progressive neural architecture search, in: European Conference on Computer Vision, 2018.
[38] Mayank Arya Chandra, S.S. Bedi, Survey on SVM and their application in image classification, Int. J. Inf. Technol. (2021).
[39] Tianqi Chen, Carlos Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, 2016.
[40] Andy Liaw, Matthew Wiener, et al., Classification and regression by randomforest, R News (2002).
[41] Wenyan Chen, Zizhao Mo, Huanle Xu, Kejiang Ye, Interference-aware multiplexing for deep learning in gpu clusters: A middleware approach, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2023.
[42] Xiaoyu Wang, Sun Xu, SARS: Towards minimizing average coflow completion time in MapReduce systems, Comput. Netw. (2024).
[43] Shulai Zhang, Quan Chen, Weihao Cui, Han Zhao, Chunyu Xue, Zhen Zheng, Wei Lin, Minyi Guo, Improving GPU sharing performance through adaptive bubbleless spatial-temporal sharing, in: Proceedings of the European Conference on Computer Systems, 2025.
[44] Rajat Phull, Cheng-Hong Li, Kunal Rao, Hari Cadambi, Interference-driven resource management for GPU-based heterogeneous clusters, in: Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing, 2012.
[45] Wenyan Chen, Chengzhi Lu, Huanle Xu, Kejiang Ye, Chengzhong Xu, Multiplexing dynamic deep learning workloads with SLO-awareness in gpu clusters, in: Proceedings of the Twentieth European Conference on Computer Systems, 2025.
[46] Wenfeng Shen, Zhengsen Liu, Yunjie Tan, Kubegpu: efficient sharing and isolation mechanisms for GPU resource management in container cloud: W. Shen et al., J. Supercomput. (2023).
[47] Jing Gu, Shengbo Song, Ying Li, Hanmei Luo, Gaiagpu: Sharing GPUs in container clouds, in: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications, 2018.
[48] Wu-Chun Chung, Jyun-Sen Tong, Zhi-Hao Chen, A fine-grained GPU sharing and job scheduling for deep learning jobs on the cloud, J. Supercomput. (2025).
[49] ONNX, Open Neural Network Exchange. https://github.com/onnx.
[50] Xinkai Wang, Chao Li, AUM: Unleashing the efficiency potential of shared processors with accelerator units for LLM serving, in: 2026 IEEE International Symposium on High Performance Computer Architecture, IEEE, 2026.

Xinhua Wang received the B.S. degree in automation from the Dalian Jiaotong University in 2018, the M.S. degree in astronomical techniques and methods from the University of Chinese Academy of Sciences in 2021. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His research interest includes cloud computing and deep reinforcement learning.

Weiwei Lin (Senior Member, IEEE) received his PhD degree in Computer Application from South China University of Technology in 2007. Currently, he is a professor in the School of Computer Science and Engineering, South China University of Technology. His research interests include cloud computing, etc. He has been the reviewers for many international journals, including IEEE TPDS, TSC, TCC, TC, etc. He is a distinguished member of CCF and a senior member of the IEEE.

Haijie Wu is currently working toward the MS degree in the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China supervised by Dr. Weiwei Lin. His research interests mainly include cloud edge collaboration, edge computing, and AI algorithms.

Jidong Zai (Senior Member, IEEE) received the BS degree in computer science from the University of Electronic Science and Technology of China, Chengdu, in 2003, and the PhD degree in computer science from Tsinghua University, in 2010. He is a tenured professor with the Department of Computer Science and Technology, Tsinghua University. His research interest is performance evaluation for high-performance computers.

Keqin Li (Fellow, IEEE) is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a Distinguished Professor at Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing. He has served on the editorial boards of the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Computers, and the IEEE Transactions on Services Computing.