DeepShare: Assurance-Driven Deep Learning Job Scheduling for Multi-Tenant Clusters

Jinghao Wang[1], Yihang Zhou[1], Xiao Zhou[1], Xinlei Zheng[1], Xiaoyang Sun[2†], Tianyu Wo[1], Chunming Hu[1], Renyu Yang[1]
[1]Beihang University [2]University of Leeds
{wang_jinghao, zhou_yihang, zhouxiao2021, xinleizh, woty, hucm, renyuyang}@buaa.edu.cn; x.sun4@leeds.ac.uk

***Abstract*—Multi-tenant GPU clusters frequently remain underutilized even when tenants experience long queueing delays, because quota control, queue ordering, preemption, and GPU sharing are driven by different local signals. We present DeepShare, a scheduler that uses a continuous tenant-assurance signal to coordinate these decisions at runtime. DeepShare combines elastic quota borrowing, tenant-specific runtime prediction, cost-aware best-effort preemption, and interference-aware MPS colocation, while using the same assurance signal to decide when borrowed capacity should be reclaimed and when sharing should become more conservative. In trace-driven experiments on 23,859 Venus jobs and 3,200 internal jobs, DeepShare achieves an average GPU utilization of 70.58%, a 29.5% improvement over the strongest non-intrusive sharing baseline, while reducing average queueing delay by 46%. On a 16-GPU Kubernetes testbed, it reduces the average job completion time by 34% and maintains 93% QoS compliance for guaranteed tenants. These results show that treating tenant assurance as a runtime control loop achieves a more advantageous utilization-QoS tradeoff than optimizing quotas, scheduling, and resource sharing independently.**

I. Introduction

Deep learning (DL) training workloads [1], [2], [3] have become primary consumers of GPU resources in modern data centers, powering applications in computer vision [4], mathematical reasoning [5], and scientific research [6]. To improve hardware efficiency, industry and academia widely adopt multi-tenant GPU clusters [7], [8], where resources are centrally managed across users and teams. In practice, such clusters often host long-running training, short exploratory jobs, and opportunistic experiments, making static resource allocation difficult to keep efficient. In such clusters, unused capacity cannot simply be reassigned without affecting tenant guarantees, yet production deployments still leave large GPU fractions idle or lightly used [9], [10]. On Alibaba's PAI platform, more than 75% of tasks request less than 10% of a GPU, while average utilization remains only 25–50% [10], [11]; similar inefficiency appears in large-scale clusters [12], where compute and memory utilization remain low despite high demand.

This underutilization stems not only from static allocation but also from resource-management decisions made with little awareness of one another. Quota mechanisms assign tenant entitlements, queueing policies choose the next job, and sharing policies decide whether two jobs can safely share a GPU. The question is not only how to find idle GPUs, but also when their use remains safe for future quota recovery. These choices interact: borrowed quota must be reclaimable, short-job priority must not delay under-served tenants, and low-interference colocation may still be undesirable if it slows quota recovery [9], [10], [13].

Recent work has improved individual parts of GPU cluster management, including job colocation [14], [15], performance-aware scheduling [16], [17], [18], and fairness-oriented allocation [19]. Yet framework-level sharing mechanisms [8], [20] often sacrifice portability, fragmentation-aware strategies rely on static boundaries, runtime prediction approaches [21], [22] are sensitive to workload regularity, and fairness models provide limited support for per-tenant service differentiation. A locally attractive decision, such as admitting a short job or a low-interference pair, can still be harmful if it delays quota recovery. The missing link is runtime coordination: a scheduler must know not only whether a job is short or a pair has low interference, but also whether admitting it helps or hurts recovery of tenants whose guarantees are unmet.

We introduce DeepShare, a Kubernetes-native resource management framework built around the *Quota Assurance Degree* $\tilde{Q}_i(t)$. QAD measures how far tenant i is from receiving its guaranteed share over time: high QAD means *guaranteed* demand is served, while low QAD exposes under-service and raises recovery priority. By comparing allocation with the smaller of quota and current *guaranteed* demand, QAD avoids rewarding artificial demand inflation or penalizing temporarily idle quota. Smoothing avoids reacting to short-lived fluctuations while preserving persistent deficits for the scheduler. *Best-effort*[1] resources are reclaimed when they interfere with recovery.

DeepShare uses QAD across scheduling decisions. Deficit-aware Reclaimable Allocation (DRA, §III-A) exposes idle capacity to *best-effort* workloads, while QAD decides when borrowed capacity gives way to *guaranteed* demand. Runtime prediction (§III-B) is applied after tenant recovery priority, so short jobs are favored only among tenants with comparable assurance states. Interference-aware colocation (§III-C) considers predicted slowdown and tenant assurance, becoming more conservative when either tenant is under-served, without

†Dr. Xiaoyang Sun is the corresponding author.

[1]In this paper, *best-effort* jobs consume reclaimable surplus capacity beyond a tenant's quota and may be preempted when *guaranteed* demand rises.

modifying user code or DL frameworks. As a tenant recovers, the same signal relaxes these decisions and re-enables more aggressive sharing.

Evaluation shows that DeepShare increases average GPU utilization to 70.58% in trace-driven simulations, a 29.5% improvement over Lucid [23], while reducing average queueing delay by 46%. On a Kubernetes testbed, DeepShare reduces average job completion time by 34%, queueing delay by 66%, and maintains 93% QoS compliance for *guaranteed* tenants [24], [25]. Ablation shows that elastic borrowing, runtime-aware ordering, and QAD-aware colocation reduce queueing delay by 31% beyond DRA (§III-A) alone.

The main contributions of this paper are:

- QAD, a continuous tenant-assurance measure that compares recovery urgency across tenants and distinguishes transient quota fluctuations from persistent under-service.
- An assurance-aware scheduling policy that applies QAD before runtime prediction, reducing queueing delay without letting short-job optimization override tenant recovery.
- A Kubernetes-native scheduler with reclaimable *best-effort* borrowing, cost-sensitive preemption, and QAD-aware MPS colocation, evaluated through trace-driven simulation and a 16-GPU deployment.

II. Background and Motivation

A. GPU Cluster Scheduling and Sharing

Multi-tenant GPU clusters have become the primary platform for deep learning (DL) research and deployment [9], [10], [26], [27]. For isolation and fairness, they are commonly partitioned into *virtual clusters* (VCs) with fixed resource quotas [14], [28]. Fixed quotas make resource allocation predictable, but they also make it difficult to reuse temporarily idle capacity across tenants.

At the same time, advances in GPU compute capability have increased the gap between hardware capacity and the resource demands of many individual DL jobs. GPU sharing [20] addresses this gap at multiple levels—hardware partitioning (e.g., NVIDIA MIG [29]), compute-level multiplexing (e.g., MPS [30]), and driver-level software virtualization—each trading off isolation against flexibility. A growing body of schedulers builds on these mechanisms to colocate multiple jobs per GPU [8], [15], [20], [23], [31], [32], employing time multiplexing, spatial partitioning, or interference-aware placement combined with higher-level scheduling policies [20], [23].

B. Empirical Study of a Multi-Tenant GPU Cluster

Although administratively convenient, fixed quotas introduce a temporal mismatch: they are provisioned on weekly or monthly timescales, while DL workloads are bursty over hours or minutes. This misalignment leads to jobs queueing behind quota limits even as GPUs remain underutilized, as shown in Fig. 1. Production studies report overall GPU utilization of only 25%–52% [9], [10], [11].

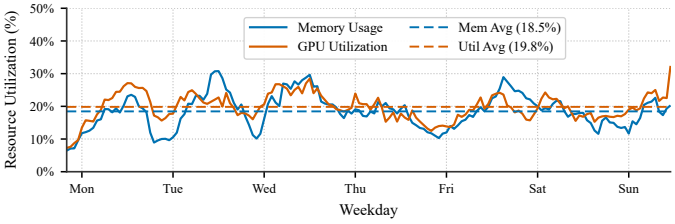


Fig. 1. GPU and memory utilization across one week.

To inform our design, we profiled a university-managed GPU cluster with 58 nodes and 219 GPUs, including NVIDIA Tesla V100-32GB and A100-40GB devices, partitioned into 12 virtual clusters with fixed monthly quotas. We collected one week of telemetry via NVIDIA DCGM, including GPU-level metrics (e.g., SM utilization, memory bandwidth), job-level statistics (e.g., resource requests), and user-level submission patterns.

Observation 1: Persistent Underutilization. Figure 1 highlights a systematic mismatch between queueing pressure and actual GPU usage. During the seven-day period, average GPU compute utilization is 19.8% and average memory utilization is 18.5%. A clear diurnal rhythm emerges: arrivals are concentrated during daytime hours (e.g., 10:00–12:00, 14:00–18:00), with substantially lower submission rates at night. Burst submissions fully consume group quotas, after which allocations remain idle until the next active window.

Observation 2: Heterogeneity and Colocation Potential. The workload shows strong heterogeneity along two dimensions. In terms of duration, short exploratory jobs (median runtime under 30 minutes) constitute 62% of submissions but use only 11% of GPU-hours, whereas long training jobs (over 4 hours) consume 73% of GPU-hours. In terms of resource profile, 41% of simultaneously running job pairs have complementary behavior: one is compute-heavy (SM utilization $> 50\%$) and memory-light ($< 40\%$ device memory), while the other is memory-heavy or compute-light. Even under a conservative 80% combined-utilization cap, many concurrent pairs could safely share a GPU under the current exclusive-assignment policy.

Observation 3: Short-Running Job Starvation. Under the existing FIFO scheduler, short jobs (62% of all submissions) incur a median queueing delay of 47 minutes, which even exceeds their own median runtime of less than 30 minutes. This unfairness occurs because the scheduler assumes runtimes are unknown and thus cannot give precedence to short jobs over long ones. However, user submission behavior is structured enough to allow data-driven prediction: per-user runtime distributions exhibit low variability, and 78% of users repeatedly submit jobs with similar characteristics and durations. Reliable runtime estimates make it possible to schedule short jobs first, avoid preempting nearly completed jobs, and predict the lifetime of colocated job pairs.

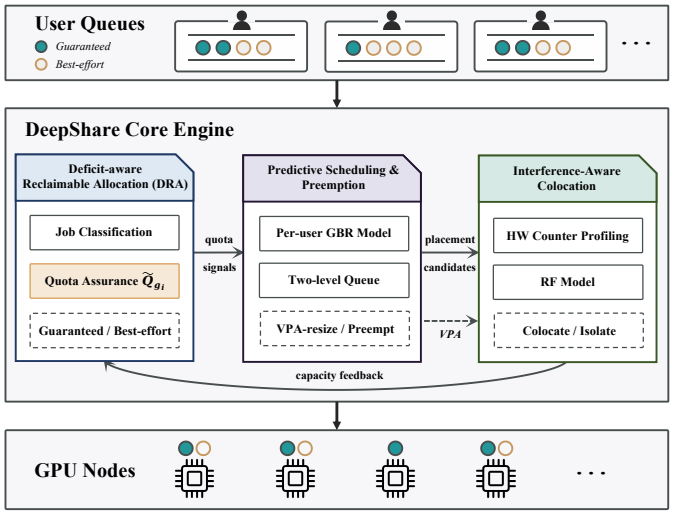


Fig. 2. The architecture of DeepShare

TABLE I
NOTATION AND DEFINITIONS.

Symbol	Description
$K,\ G_{\text{tot}}$	Number of GPUs and total GPU capacity
$G_f,\ G_p$	Free GPUs and total pending GPU demand
$d,\ \mathcal{J}_d$	GPU device and its resident jobs
R_j	GPU demand of job j
$\mathcal{D}_j^P,\ V(d)$	Preemption-feasible devices and victims on device d
$i,\ q_i$	Tenant index and *guaranteed* GPU quota
$A_i^G,\ D_i^G$	Allocated and demanded *guaranteed* GPUs
$U_i^G,\ U_i^B$	Current *guaranteed* and *best-effort* GPU usage
$Q_i,\ \tilde{Q}_i$	Instantaneous and EMA-smoothed QAD
$\eta,\ \tau_p$	*Best-effort* cap multiplier and preemption window
$\mathcal{Q}_i^G,\ \mathcal{Q}_i^B$	Tenant-level *guaranteed* and *best-effort* queues
$\mathcal{Q}^G,\ \mathcal{Q}^B$	Cycle-local *guaranteed* and *best-effort* placement queues
$\hat{T}(j),\ \bar{T}(J)$	Predicted remaining runtime and victim-set mean runtime
$C_p(j),\ \Phi(J)$	Normalized preemption overhead and victim-set cost
$\rho,\ \hat{\rho}$	Retention ratio and predicted retention
$\rho_{\text{tol}},\ \rho_{\min}$	Admission tolerance and retention floor
P	Cluster contention pressure

C. Motivation

The observations above suggest that underutilization arises primarily from rigid resource-management policies rather than insufficient workload demand. Improving utilization therefore requires elastic borrowing across tenants, differentiated treatment of *guaranteed* and *best-effort* work, runtime-aware scheduling to reduce head-of-line blocking, and interference-aware colocation to exploit complementary jobs. These mechanisms must also be tied to tenant assurance: borrowed resources should remain reclaimable, short-job acceleration should not postpone an under-served tenant, and colocation should become more conservative when it may slow quota recovery. At the same time, practical deployment demands framework-agnostic designs to avoid per-framework maintenance overhead.

III. DeepShare Design

DeepShare organizes scheduling decisions around a tenant-assurance state, the Quota Assurance Degree $\tilde{Q}_i(t)$, as shown in Figure 2. DeepShare uses this state in three places: DRA (§III-A) decides when idle quota can be borrowed or reclaimed, predictive scheduling and preemption (§III-B) order jobs and select low-cost victims, and interference-aware colocation (§III-C) decides when two jobs can safely share a GPU without delaying tenant recovery.

A. DRA: Deficit-aware Reclaimable Allocation

Strictly static quota enforcement causes GPUs to be underutilized when multi-tenant demand is bursty, even though DL workloads differ in how much reclaimable excess capacity they can safely tolerate, analogous to oversubscription [12]. DeepShare introduces *Deficit-aware Reclaimable Allocation* (DRA), which addresses these issues by translating quota fulfillment into a continuous control signal and separating scheduling semantics from the underlying physical placement.

Workload classes. DeepShare targets DL training workloads and distinguishes between quota-backed and opportunistic execution [7], [33]. Each tenant labels submitted jobs as either *guaranteed* or *best-effort*. *Guaranteed* jobs are charged against the quota q_i and prioritized for placement backed by the quota. *Best-effort* jobs are not charged against the quota and incur negligible scheduling cost; they utilize surplus capacity and remain reclaimable when *guaranteed* demand is underserved.

Quota Assurance Degree. For each tenant i, DeepShare defines the instantaneous Quota Assurance Degree (QAD) as in Eq. 1. QAD contrasts the tenant's current *guaranteed* allocation with $\min(q_i, D_i^G(t))$, where $D_i^G(t)$ is its present *guaranteed* demand. This definition prevents tenants from boosting their priority by declaring demand far above their quota, while also ensuring they are not penalized for leaving part of their quota unused. In this way, QAD measures how effectively the scheduler fulfills the *guaranteed* demand that the tenant is entitled to receive.

$$Q_i(t) = \begin{cases} 1, & D_i^G(t) = 0, \\ A_i^G(t)/\min(q_i, D_i^G(t)), & D_i^G(t) > 0. \end{cases} \tag{1}$$

Temporal smoothing and control. Binary quota mechanisms (e.g., Kubernetes ElasticQuota and Volcano [24], [25]) provide only a coarse indication of whether a tenant is within its quota. This granularity is insufficient for prioritizing recovery among under-served tenants or regulating the aggressiveness of sharing. In contrast, QAD provides a continuous signal: lower values indicate greater deficit, while smoothing mitigates transient fluctuations due to short job completions or bursty arrivals. To stabilize this signal, DeepShare applies exponential moving average (EMA) smoothing (Eq. 2) with default $\lambda = 0.3$. Under a 50 ms scheduling cycle, a sustained deficit contributes 90% of its steady-state effect within ∼350

ms, enabling timely reclamation while filtering sub-second noise.

The smoothed QAD, $\tilde{Q}_i(t)$, drives multiple control decisions. *Guaranteed* jobs are ordered by $(\tilde{Q}_i(t)\uparrow, \hat{T}(j)\uparrow)$; colocation admission is tightened when either tenant is underserved; and reclamation is prioritized before considering *best-effort* preemption. *Best-effort* allocations do not contribute to $A_i^G(t)$ and therefore do not inflate QAD. Preemption is triggered by placement failure.

$$\tilde{Q}_i(t) = \lambda Q_i(t) + (1-\lambda)\tilde{Q}_i(t-1) \tag{2}$$

Recovery dynamics. When *guaranteed* demand is feasible, i.e., $\sum_i \min(q_i, D_i^G(t)) \le G_{\text{tot}}$, a *guaranteed* job blocked solely by *best-effort* occupancy is admitted within one preemption window τ_p, subject to Kubernetes Pod termination latency and a bounded *best-effort* cleanup delay δ. Once $A_i^G = \min(q_i, D_i^G)$, we have $Q_i = 1$, and the EMA converges geometrically at rate $(1-\lambda)$ per scheduling cycle.

Under overload, where $\sum_i \min(q_i, D_i^G(t)) > G_{\text{tot}}$, full recovery is infeasible. In this regime, prioritizing tenants in ascending order of $\tilde{Q}_i$ approximates max-min fair recovery among active tenants.

B. Predictive Scheduling and Preemption

Conventional schedulers [7], [33] generally assume unknown training durations and use FIFO or static priorities, which causes head-of-line blocking. While integrating runtime prediction can alleviate this, prioritizing jobs solely by predicted time can undermine tenant-level guarantees.

DeepShare instead adopts a lexicographic policy: *guaranteed* jobs are ordered first by QAD and then by the predicted remaining time. QAD determines inter-tenant recovery priority, while prediction refines intra-tenant ordering among jobs with similar assurance levels. This design improves responsiveness and reduces unnecessary preemption of *best-effort* jobs that are near completion.

Hierarchical queueing. DeepShare implements tenant-level admission queues and cycle-local cluster placement queues. Each cycle initializes $\mathcal{Q}^G, \mathcal{Q}^B = \emptyset$ and provisional usage $a_i^G = U_i^G$, $a_i^B = U_i^B$. Appending job j immediately increases the corresponding a_i by R_j, so the promoted batch cannot exceed q_i for *guaranteed* jobs or ηq_i for *best-effort* jobs, with default $\eta = 2$. `PlaceJobs` atomically updates actual usage after successful placement; unplaced jobs remain in their tenant queues when the cycle-local state is discarded.

At the cluster level, jobs are scheduled using a lexicographic priority. *Guaranteed* jobs are considered before *best-effort* jobs; within each class, jobs are ordered by $(\tilde{Q}_i(t)\uparrow, \hat{T}(j)\uparrow)$. Because QAD is the primary key, runtime predictions only refine job ordering after tenant-assurance priority has been determined. Prediction errors therefore affect only local ordering among similarly served tenants; they cannot override quota assurance, elevate a well-served tenant above an under-served one, or trigger preemption of *guaranteed* jobs.

Cost-based preemption. DeepShare never selects *guaranteed* jobs as preemption victims. For incoming job j, the resident set $V(d)$ and preemption-feasible device set $\mathcal{D}_j^P$ are

$$\begin{aligned} V(d) &= \mathcal{J}_d, \\ \mathcal{D}_j^P &= \{d \mid \text{Feasible}(j,d),\ V(d) \neq \emptyset, \\ &\qquad \text{class}(v) = \text{BE},\ \forall v \in V(d)\}. \end{aligned} \tag{3}$$

Here, $\text{Feasible}(j,d)$ means that d satisfies j's GPU-memory and node-level CPU/memory constraints after eviction. Therefore, reclaiming a shared GPU always preempts its complete resident set.

DeepShare estimates each job's remaining runtime $\hat{T}(j)$ using an offline-trained per-tenant gradient boosting model [34], with a cluster-wide fallback for cold-start tenants. We define the preemption cost of a victim set J as Eq. 4, where $\bar{T}(J) = |J|^{-1}\sum_{j\in J}\hat{T}(j)$ denotes the mean remaining runtime, and $C_p(j)$ captures the normalized cumulative preemption overhead of job j, discouraging repeated interruption of the same *best-effort* job. The first term prioritizes preempting long-running jobs with limited prior disruption, while the second term penalizes fragmented victim sets, particularly those consisting of near-completion jobs (i.e., small $\bar{T}(J)$). We set $\alpha = 0.5$ and $\beta = 0.3$ by default.

$$\Phi(J) = \sum_{j\in J} \frac{1+\alpha C_p(j)}{\hat{T}(j)} + \frac{\beta(|J|-1)}{\bar{T}(J)}, \tag{4}$$

The `SelectVictims` procedure selects

$$d^* = \arg\min_{d\in\mathcal{D}_j^P} \Phi(V(d)), \qquad J^* = V(d^*). \tag{5}$$

Thus, both α and β directly affect online selection. The scheduler reserves d^* before eviction; if $\mathcal{D}_j^P = \emptyset$, it returns $\perp$ and leaves j queued. Ranking n candidate devices costs $O(n \log n)$.

Algorithm 1: Scheduling Strategy

Input : Tenants U, cluster state
Output: Updated placements

$\mathcal{Q}^G, \mathcal{Q}^B \leftarrow \emptyset$
foreach tenant $i \in U$ **do**
  $a_i^G \leftarrow U_i^G$
  foreach job $j \in \mathcal{Q}_i^G$ **do**
    if $a_i^G + R_j \le q_i$ **then**
      Append j to $\mathcal{Q}^G$; $a_i^G \leftarrow a_i^G + R_j$
Sort $\mathcal{Q}^G$ by $(\tilde{Q}_i(t)\uparrow, \hat{T}(j)\uparrow)$
`PlaceJobs(`$\mathcal{Q}^G$, ***true***`)`
if no remaining $j \in \mathcal{Q}^G$ is placeable **then**
  foreach tenant $i \in U$ **do**
    $a_i^B \leftarrow U_i^B$
    foreach job $j \in \mathcal{Q}_i^B$ **do**
      if $a_i^B + R_j \le \eta q_i$ **then**
        Append j to $\mathcal{Q}^B$; $a_i^B \leftarrow a_i^B + R_j$
Sort $\mathcal{Q}^B$ by $(\tilde{Q}_i(t)\uparrow, \hat{T}(j)\uparrow)$
`PlaceJobs(`$\mathcal{Q}^B$, *false*`)`

Algorithm 2: Interference-Aware Placement

Input : Queue $\mathcal{Q}_{\text{in}}$, flag *allowPreempt*, cluster state
Output: Updated job-to-GPU assignments

foreach job $j \in \mathcal{Q}_{\text{in}}$ in priority order **do**
  $d \leftarrow \text{FindExclusiveDevice}(j)$
  if $d \neq \emptyset$ **then**
    Atomically assign j to d
    continue
  placed $\leftarrow$ **false**
  $P \leftarrow (G_p/(G_f+\epsilon))^{\gamma}$
  $\mathcal{C} \leftarrow \{(d, j_e) \mid d$ is feasible for j and hosts exactly one job $j_e\}$
  Sort $\mathcal{C}$ by $(\min(\hat{\rho}(j), \hat{\rho}(j_e))\downarrow, \hat{T}(j_e)\uparrow)$
  foreach $(d, j_e) \in \mathcal{C}$ **do**
    $i \leftarrow \text{owner}(j)$; $i_e \leftarrow \text{owner}(j_e)$
    $\rho_{\text{tol}} \leftarrow \min(1, [\rho_{\min} + P(1-\rho_{\min})] \max(k - \tilde{Q}_i(t), k - \tilde{Q}_{i_e}(t)))$
    if (j *or* j_e *is BE*) *and* $\hat{\rho}(j) \geq \rho_{\text{tol}}$ *and* $\hat{\rho}(j_e) \geq \rho_{\text{tol}}$ **then**
      Colocate j on d
      placed $\leftarrow$ **true**; **break**
  if *placed* **then**
    continue
  if *allowPreempt* **then**
    Shrink BE Pods via in-place resize
    Update per-node CPU/mem headroom
    placed $\leftarrow \text{RetryPlacement}(j)$
    if *placed* **then**
      continue
    $(J^*, d^*) \leftarrow$ `SelectVictims(`j`)`
    if $J^* \neq \perp$ **then**
      Preempt J^* and bind j to reserved d^* after release

In summary, Algorithm 2 considers placement options in order of increasing disruption: exclusive allocation, interference-aware colocation, CPU/memory reclamation, and finally GPU preemption.

C. Interference-Aware Job Colocation

Sharing GPUs can increase utilization, but a low-interference pairing is not always a good choice. If one tenant is currently running below its *guaranteed* allocation, even a modest slowdown can postpone its recovery. DeepShare therefore treats colocation as an admission-control decision: a pair is scheduled together only when the predicted throughput retention is high enough given both the current cluster load and the assurance states of the tenants involved.

Performance retention prediction. DeepShare estimates pairwise interference via the throughput retention ratio $\rho = t_{\text{shared}}/t_{\text{excl}}$, where t_{shared} and t_{excl} denote the training throughput in colocated and exclusive execution, respectively.

The predictor leverages DCGM hardware telemetry, including SM activity, memory bandwidth, L2 cache behavior, PCIe traffic, tensor core utilization, DRAM throughput, and power draw, to capture both compute and memory contention effects. We use a Random Forest trained offline on isolated and colocated job profiles, balancing prediction accuracy with sub-millisecond inference latency (§V), which allows online deployment on the critical path of the scheduler. Feature selection via Recursive Feature Elimination indicates that SM activity and memory bandwidth are the dominant predictors of interference, consistent with previous observations on GPU contention (Fig. 3).

Dynamic tolerance. Let G_p be total pending GPU demand and G_f the number of free GPUs. Cluster contention pressure is defined as

$$P = \left(\frac{G_p}{G_f + \epsilon}\right)^{\gamma}, \tag{6}$$

where ϵ avoids division by zero. We set $\gamma = 0.5$ so pressure grows sub-linearly, preventing short demand bursts from disabling all colocation. For incoming job j from tenant i and resident job j_e from tenant i_e, the admission tolerance is defined in Eq. 7, where k denotes the maximum number of jobs colocated on a single GPU (default $k = 2$).

$$\rho_{\text{tol}} = \min\big(1,\ [\rho_{\min} + P(1-\rho_{\min})] \times \max(k - \tilde{Q}_i(t), k - \tilde{Q}_{i_e}(t))\big). \tag{7}$$

A pair is admitted only if it fits GPU memory and node-level CPU/memory headroom, at least one partner is *best-effort*, and both $\hat{\rho}(j)$ and $\hat{\rho}(j_e)$ exceed ρ_{tol}. The $[\rho_{\min} + P(1-\rho_{\min})]$ term raises the tolerance when pending demand is high. The clamp at one is deliberate: when contention is high or one of the tenants is significantly under-served, the system stops creating new colocations and the scheduler focuses on restoring exclusive capacity first.

Thus, admission behaves conservatively when the cluster is under heavy load or when a tenant's recovery is urgent, and only becomes more permissive when the overall pressure is low and both tenants are considered sufficiently protected.

Runtime validation and overhead. The predictor filters pairs before GPU sharing is enabled, but interference can change across training phases. DCGM counters feed an online retention estimator. If either job's observed retention stays below ρ_{tol} for w consecutive samples, the pair is marked as degraded, the colocation decision is revoked, and the *best-effort* partner is preempted in the next scheduling cycle; the *guaranteed* job continues running. The fixed-window rule provides bounded detection delay with one counter per pair, cheaper and more predictable than sequential detectors such as CUSUM.

Let m be the number of GPUs hosting exactly one resident job. Candidate generation and prediction are $O(m)$ after pruning, and sorting costs $O(m \log m)$. In practice, the scheduler's node-scoring phase (§IV-A) caps the evaluated candidates per job, keeping colocation admission within the scheduling latency budget reported in §IV-C.

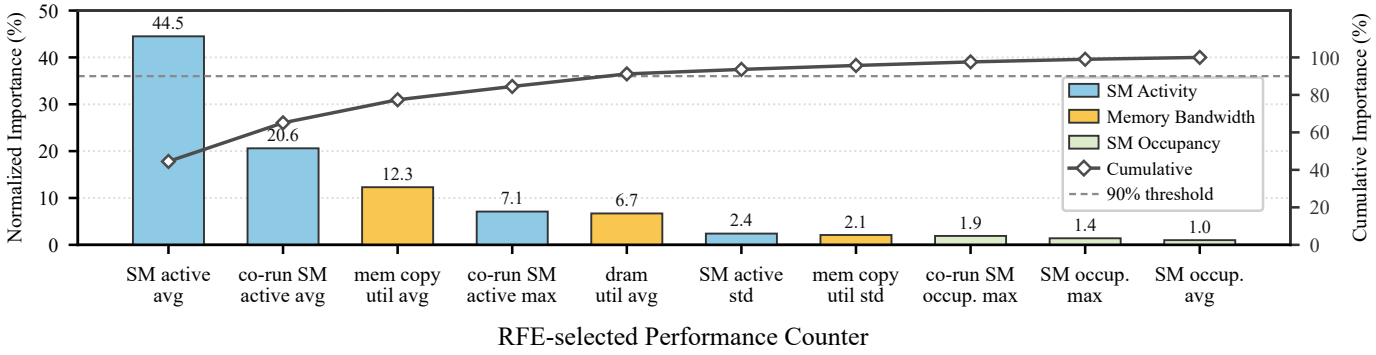


Fig. 3. Feature importance scores via Recursive Feature Elimination with the Random Forest Regressor.

IV. System Implementation

The prototype contains roughly 11.3K lines of code: Go for the scheduler plugin, quota controller, and node-local DaemonSets; Python for offline-trained Random Forest interference models and per-tenant runtime estimators; and Helm/RBAC manifests plus lightweight quota CRDs for deployment. All mechanisms run on the cluster-management layer and do not require changes to user code or DL frameworks.

A. Scheduler Critical Path

DeepShare is packaged as an out-of-tree scheduler plugin that registers with the standard Kubernetes scheduling framework [24]. Tenant quotas and *best-effort* cap multipliers are configured through a lightweight `TenantQuota` CRD, while individual jobs specify their service class using the optional annotation `deepshare.io/class: guaranteed|best-effort`.

A lightweight quota controller reconciles `TenantQuota` objects and job-class annotations into per-tenant quota metadata, but it does not make placement decisions. The scheduler plugin keeps the real-time control loop: it maintains $Q_i(t)$ and $\tilde{Q}_i(t)$ in memory, orders queues, admits colocation pairs, and invokes preemption. The QAD state is derivable from running Pods in the informer cache; after failover, the newly elected leader reconstructs it and warm-starts the EMA with the first cycle's instantaneous QAD, avoiding extra writes or storage.

The plugin maps DeepShare's policy to five extension points. (i) *Filter* keeps nodes that can host the incoming Pod on a free GPU or an eligible single-resident GPU under CPU, memory, GPU-memory, and *best-effort* cap constraints. (ii) *Score* ranks these candidates using the Random Forest interference model and the bilateral tolerance in §III-C. (iii) *Reserve* atomically records device claims and cycle-local tenant reservations in the next-cycle cluster view to avoid double booking and quota over-admission. (iv) *PostFilter* evaluates each feasible device's complete *best-effort* victim set using Eq. 4 and reserves the selected device before eviction. (v) *Permit* holds an incoming Pod only when the VPA-based CPU/memory reclamation step in Algorithm 2 has issued an in-place resize and waits until `Pod.Status.Resources` reflects the reduced allocation.

B. GPU Sharing and Runtime Protection

Spatial sharing is provided by NVIDIA MPS. One MPS control daemon is deployed per GPU in a node-local DaemonSet; it brokers client connections and sets per-client memory limits through MPS controls where supported. Because MPS multiplexes clients rather than isolating SM or memory-bandwidth contention, DeepShare treats sharing as an admission-control problem governed by the interference model in §III-C.

A DCGM poller samples each colocated pair. If observed retention stays below the bilateral ρ_{tol} for three consecutive windows, the poller raises a degradation event; the scheduler consumes it in the next cycle and preempts the *best-effort* partner. A DaemonSet recovery hook clears stale MPS client contexts after abnormal Pod exits.

In-place resize. Because `nvidia.com/gpu` is defined as a Kubernetes Extended Resource, its allocation cannot be modified once the Pod has been admitted. The scheduler relies on the Pod `resize` subresource, which is enabled via `InPlacePodVerticalScaling` on our control plane, and it maintains CPU and memory headroom above the VPA recommendations of $\max(10\%, 0.5\,\text{core})$ and $\max(10\%, 256\,\text{MB})$, respectively.

C. Scheduling Latency and Fault Tolerance

End-to-end scheduling latency stays below 50 ms per job, dominated by the Kubernetes bind round trip. Feature extraction, queue bookkeeping, and capped interference scoring together account for less than 25 ms, keeping the latency envelope on par with Lucid [23].

The plugin runs as a replicated `Deployment` with leader election through `coordination.k8s.io/Lease`. With Kubernetes default lease settings, a newly elected replica begins scheduling within one renewal period. Because preemption is expressed as Pod deletion and reconciled idempotently by the API server, no bespoke compensation logic is needed if leadership changes mid-cycle.

V. Experiments

This section assesses DeepShare in terms of prediction accuracy, colocation efficiency, multi-tenant quota management,

Fig. 4. Predicted vs. actual job execution times for 2,000 randomly sampled jobs from the Venus dataset.

TABLE II
REPRESENTATIVE SINGLE-GPU TRAINING WORKLOADS USED FOR INTERFERENCE PROFILING.

Domain	Workload	Input configuration	Batch
CV	ResNet-18 [4]	224×224 classification	128
CV	ResNet-20 [4]	32×32 classification	512
CV	MobileNetV3-Small [35]	Lightweight image classification	256
CV	ResNet-50 [4]	224×224 classification	48
3D	PointNet-style [36]	1,024 points per sample	96
GAN	DCGAN-style [37]	Adversarial image generation	64
RL	Actor–Critic-style [38]	Policy and value optimization	4,096
NLP	BERT-style MLM [39]	MLM, sequence length 128	48
Speech	DeepSpeech2-style [40]	Conv–BiGRU with CTC	16
RecSys	NCF-style [41]	Embedding with MLP	8,192
NLP	Transformer seq2seq [42]	Encoder–decoder, sequence length 64	48
CV	U-Net-style [43]	128×128 image translation	8
NLP	LSTM language model [44]	2-layer LM, sequence length 192	128

and deployment in a real cluster.

A. Experiment Setup

Evaluation environments.

We use production traces from a Kubernetes-managed cluster with 58 nodes and 219 GPUs for trace-driven evaluation. We also deploy DeepShare end to end on a 16-A100 Kubernetes testbed and run 50 jobs. The testbed uses Ubuntu 22.04, Kubernetes v1.35 with `InPlacePodVerticalScaling`, CUDA 12.8, and NVIDIA DCGM 3.3.8. The 58-node production cluster provides telemetry and traces for large-scale replay, while the 16-A100 testbed hosts the complete DeepShare prototype for end-to-end validation. This separation distinguishes large-scale policy evaluation from implementation feasibility under real Kubernetes scheduling and GPU sharing.

Profiling workloads. Table II summarizes the representative single-GPU training workloads used for interference profiling. The suite covers diverse model structures and resource behaviors, while synthetic inputs remove data-loading and storage I/O variability.

Workloads. We used two real-world datasets, summarized in Table III. The Venus trace [26] contains large-scale GPU datacenter scheduling and utilization records from September 2020, providing diverse public workloads for reproducible evaluation. The internal trace preserves user quotas, submissions, and allocation policies from our institutional cluster; its longer average runtime (36,887 s vs. 5,419 s) stresses scheduling under long-lived occupancy.

TABLE III
CHARACTERISTICS OF THE SIMULATION DATASETS.

Dataset	Source	Job Count	Avg. Execution Time
Venus [26]	Public dataset	23,859	5,419 s
Internal	Internal cluster	3,200	36,887 s

TABLE IV
ACCURACY METRICS OF JOB EXECUTION TIME PREDICTION ALGORITHMS ON THE VENUS DATASET.

Algorithm	MAPE (%)	R^2 Score
Lucid	68.72	0.6413
DeepShare	**31.84**	**0.7286**

Trace replay methodology. We implemented a trace-driven simulator that reproduces DeepShare's two-level queues, DRA module, preemption, and interference-aware colocation. It supports FIFO, SJF, QSSF, Tiresias, Lucid, and DeepShare's strategies.

For the Venus dataset, which does not include per-tenant quota assignments, we synthesize a multi-tenant quota configuration as follows: jobs are partitioned into 12 virtual clusters by user ID hash, and each virtual cluster receives a fixed quota proportional to its historical GPU-hour share (range: 4–32 GPUs), mirroring the quota distribution observed in our university cluster (§II-B).

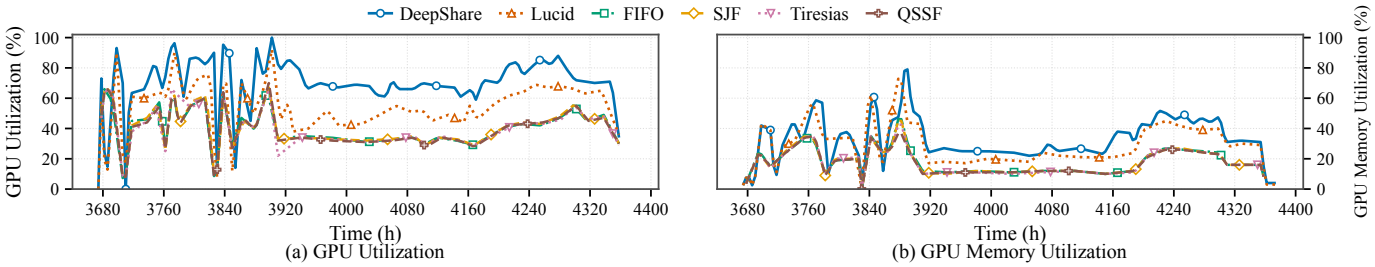
Metrics. We evaluated the performance of our scheduling strategies using the following metrics: (1) *Job Completion Time* (JCT): time from job submission to completion, encompassing both queueing and execution time; (2) *Average Queueing Delay*: average time a job spends in the queue before being scheduled; (3) *Makespan*: total time from the first submission to the last completion; (4) *Cluster GPU Utilization*: average percentage of GPU compute resources used; (5) *GPU Memory Utilization*: average percentage of GPU memory used; and (6) *Quota Assurance Degree*: as defined in Eq. 1, measuring SLA fulfillment.

Baselines. We compare DeepShare against five baselines: FIFO, SJF, QSSF [26], Tiresias [22], and Lucid [23]. FIFO, SJF, QSSF, and Tiresias represent scheduling-only policies with different ordering assumptions, while Lucid is the strongest non-intrusive sharing baseline. All baselines use identical quota assignments and workload traces.

B. Job Execution Time Prediction Accuracy

We assessed the accuracy of our job execution time prediction algorithm in comparison to Lucid [23] using the Venus dataset. Table IV summarizes the key metrics.

DeepShare reduces MAPE from 68.72% to 31.84% (a 2.16× error reduction) and raises R^2 from 0.6413 to 0.7286 on the Venus dataset (Table IV). Fig. 4 confirms tight predicted-vs.-actual alignment, with accuracy highest for users with ≥50 historical submissions (MAPE < 25%) and graceful degradation for cold-start users via the cluster-wide fallback (MAPE < 60%).



(a) GPU Utilization (b) GPU Memory Utilization

Fig. 5. GPU and GPU memory utilization over time.

TABLE V
COMPARATIVE GPU RESOURCE UTILIZATION ACROSS SCHEDULING STRATEGIES.

Metric	FIFO	SJF	QSSF	Tiresias	Lucid	DeepShare
GPU Util (%)	39.64	40.00	39.40	39.27	54.52	**70.58**
Mem Util (%)	17.94	17.72	17.42	17.48	28.74	**32.67**

C. Colocation Strategy Performance

We evaluated DeepShare's interference-aware colocation strategy against non-sharing (FIFO, SJF, QSSF, Tiresias) and sharing (Lucid) baselines.

Resource utilization. Table V compares GPU and memory utilization across all strategies.

DeepShare achieves 70.58% average GPU utilization and 32.67% memory utilization, representing a 29.5% relative improvement in GPU utilization over Lucid and a 78.1% improvement over FIFO. The utilization gain over Lucid is non-trivial because both systems perform colocation; the difference stems from three factors: (i) DeepShare's more accurate interference model ($R^2 = 0.902$ vs. Lucid's simpler scoring approach) enables it to accept more colocation pairs that Lucid conservatively rejects; (ii) the dynamic tolerance mechanism aggressively colocates jobs when contention is low (i.e., when $P = \left(\frac{G_p}{G_f+\epsilon}\right)^{\gamma} \approx 0$), exploiting off-peak periods that static thresholds cannot adapt to; and (iii) the DRA module channels additional *best-effort* jobs into idle capacity, increasing the pool of colocation candidates.

The improvement in memory utilization over Lucid is more modest (13.7%) because GPU memory is a hard constraint that limits the scope of sharing. Fig. 5 shows the temporal dynamics of GPU utilization across strategies. DeepShare maintains a consistently higher and more stable utilization over time, with fewer periods of underutilization.

Job completion time and queueing delay. Table VI presents the JCT and queueing delay results. DeepShare reduces average queueing delay by 46% over Lucid (1,068 s vs. 1,976 s) and by 98% over FIFO. This is especially important for short exploratory and opportunistic jobs, whose responsiveness is often dominated by waiting time. The JCT improvement over Lucid is smaller (6.3%) because execution time is mainly workload-dependent and remains broadly comparable across schedulers for the same job set.

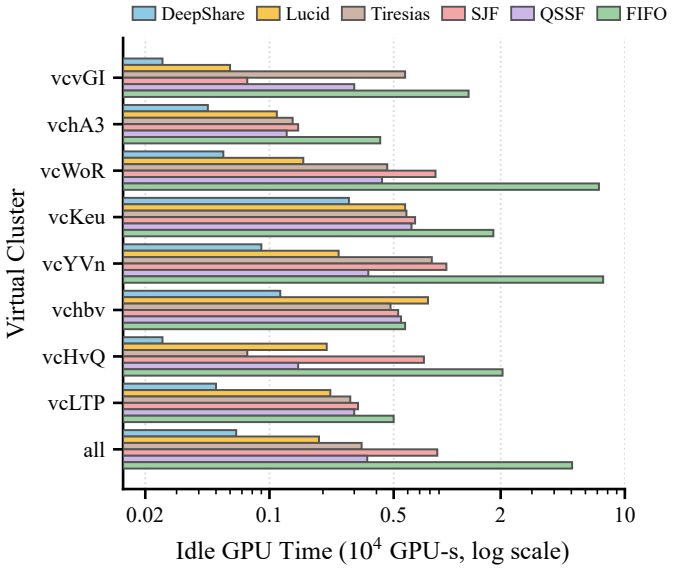


Fig. 6. Idle GPU time across different Venus virtual-cluster configurations.

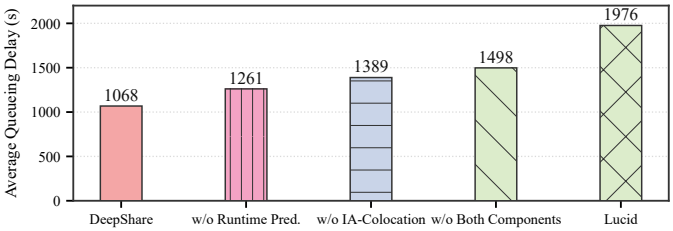


Fig. 7. Ablation of average queueing delay.

Accordingly, DeepShare should be interpreted primarily as improving admission responsiveness and tenant-level resource assurance rather than accelerating the computation of an already-running job. The end-to-end benefit is therefore most visible for exploratory and short-running jobs whose latency is dominated by queueing. For long-running training jobs, execution time dominates JCT, so the relative JCT reduction is naturally smaller even when the absolute waiting-time reduction remains substantial.

The CDF of job completion times (Fig. 8) shows that DeepShare achieves faster completion across all percentiles. Notably, the improvement is most pronounced at the tail

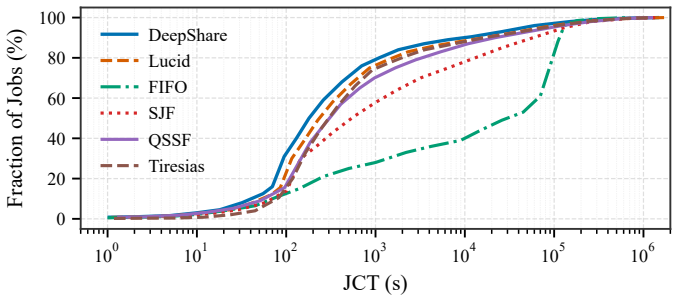


Fig. 8. CDF of job completion times.

TABLE VI
PERFORMANCE COMPARISON OF SCHEDULING POLICIES (ALL VALUES IN SECONDS).

Metric	FIFO	SJF	QSSF	Tiresias	Lucid	DeepShare
JCT	64,314	21,094	15,550	14,684	13,877	**13,001**
Queueing Delay	52,546	9,325	3,090	2,900	1,976	**1,068**

(P95 and P99), where DeepShare reduces tail JCT by 23% compared to Lucid, indicating that it effectively prevents worst-case scenarios for long-queued jobs.

DeepShare consistently achieves the lowest idle GPU time across all tested Venus cluster configurations, shown in Fig. 6. Averaged over the eight configurations, it reduces idle GPU time by 71.0% relative to Lucid and 96.8% relative to FIFO; in the aggregate "all" configuration, idle GPU time further drops to 0.065×10^4 GPU-s, compared with 0.190×10^4 for Lucid and 5.05×10^4 for FIFO.

Ablation study. Fig. 7 isolates the contributions of runtime prediction and interference-aware colocation to average queueing delay. The full DeepShare strategy reduces average queueing delay by 45.9% relative to Lucid ($p < 0.01$, Wilcoxon signed-rank test). Removing runtime prediction increases queueing delay by 18.4%, while removing interference awareness increases it by 30.1%, showing that both components matter and that colocation contributes the larger share.

Their interaction with DRA is evaluated separately in Fig. 10. In all configurations, QAD remains the primary control signal, preventing short-job optimization and aggressive sharing from overriding tenant recovery.

D. Multi-Tenant Quota Management

We evaluate DRA's impact on fairness and queueing delay against fixed-quota schedulers, Tiresias, and real-world system traces (*Real*) from the internal cluster. The *Observed* series in Figs. 9 and 10 denotes this original internal-cluster behavior and corresponds to *Real* in Table VII. We report average instantaneous QAD $Q_i(t)$ to measure raw quota satisfaction, and use smoothed $\tilde{Q}_i(t)$ for cycle-level QoS compliance because it matches the scheduler's control signal. Table VII presents the key metrics.

Among the evaluated strategies, DeepShare is the only one that achieves an average $Q_i(t)$ of 1.0 while maintaining low queueing delay (1,168 s). Fixed-quota strategies (FIFO, SJF, QSSF) trivially achieve average $Q_i(t)$ of 1.0 but at the cost of extremely high queueing delays (36–84× higher). Tiresias improves scheduling efficiency but severely violates quota guarantees (average $Q_i(t)$ of 0.15), making it unsuitable for multi-tenant environments with SLAs. The real system traces show a compromise (average $Q_i(t)$ of 0.75, delay 4,466 s) that neither fully satisfies quotas nor minimizes delay.

TABLE VII
QUOTA GUARANTEE AND QUEUEING DELAY ACROSS STRATEGIES.

Strategy	Avg. $Q_i(t)$	Avg. Queueing Delay (s)
FIFO	1.00	98,000
SJF	1.00	42,387
QSSF	1.00	97,271
Tiresias	0.15	86,754
Real	0.75	4,466
DeepShare	**1.00**	**1,168**

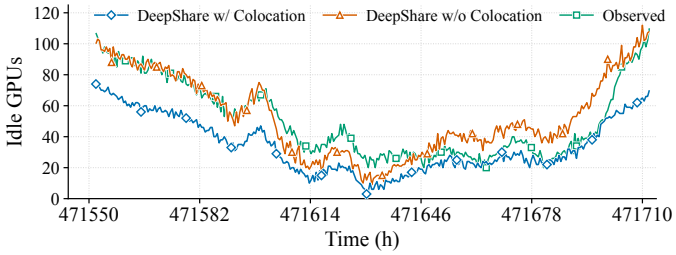


Fig. 9. Idle GPU counts over time.

DRA's *elastic quota mechanism* makes this possible: tenants opportunistically use idle capacity, while QAD keeps borrowed capacity reclaimable when *guaranteed* demand becomes under-served. This decouples quota compliance from utilization, so the two are no longer in tension.

Fig. 9 shows the time-series of idle GPU counts. DeepShare maintains consistently fewer idle GPUs, confirming that its elastic allocation effectively fills resource gaps.

QoS compliance analysis. A scheduling cycle is compliant for tenant i if $\tilde{Q}_i(t) \geq 0.95$. In the physical deployment (§V-E), 93% of tenant-cycle pairs meet this threshold. The remaining 7% of violations concentrate during two scenarios: (i) burst arrivals where multiple tenants temporarily exceed aggregate capacity, and (ii) transient periods immediately following preemption events, before $\tilde{Q}_i(t)$ recovers. In the Venus simulation, the median per-tenant $\tilde{Q}_i(t)$ is 0.98 (IQR 0.94–1.00), and no tenant experiences $\tilde{Q}_i(t) < 0.85$ for more than 2% of scheduling cycles, confirming per-tenant (not just aggregate) compliance.

QAD ablation. To isolate the role of QAD, we ablate its use in job ordering (-Ord.), colocation gating (-Colo.), *best-effort* reclamation (-Rec.), and all QAD-dependent decisions (-All), while keeping the remaining DeepShare mechanisms enabled. As shown in Fig. 11, removing QAD consistently weakens tenant assurance. In the QAD-ablation workload, Full DeepShare achieves about 92% QoS compliance and a worst-tenant average QAD of about 0.95, whereas -All reduces

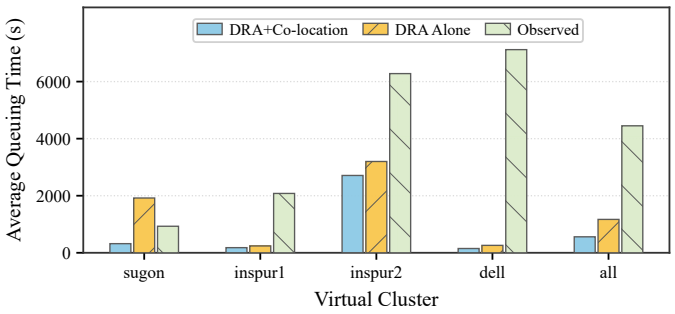


Fig. 10. Queueing delay with DRA alone and with DRA plus interference-aware colocation. The further 31% reduction shows the synergy between elastic quota allocation and colocation.

them to about 66% and 0.81, respectively. *Guaranteed*-job queueing delay also increases from below 900 seconds in Full DeepShare to roughly 1,600 seconds in -All. In contrast, GPU utilization remains comparable and can even be slightly higher without QAD, because the scheduler admits borrowing and colocation more aggressively. This confirms that QAD is not primarily a utilization booster; rather, it is the control signal that prevents prediction, borrowing, and colocation from sacrificing under-served tenants.

Overload behavior. When aggregate *guaranteed* demand exceeds cluster capacity (approximately 8% of Venus peak-hour cycles), DeepShare degrades gracefully: Algorithm 1 prioritizes the most under-served tenants by ascending $\tilde{Q}_i(t)$ order. The worst-case per-tenant $\tilde{Q}_i(t)$ during overload is 0.72, with recovery to $\tilde{Q}_i(t) \geq 0.95$ within five cycles ($\sim$250 ms) after the spike subsides. *Best-effort* jobs absorb most of the overload cost, with a 2.1$\times$ increase in queueing delay versus only a 14% increase for *guaranteed* jobs, confirming the intended service differentiation.

Synergy between DRA and colocation. Fig. 10 evaluates their combined effect and shows that adding interference-aware colocation to DRA yields a further 31% reduction in queueing delay relative to DRA alone. This synergy arises because colocation increases the cluster's usable capacity by placing two jobs per GPU where possible, while QAD prevents those placements from delaying recovery of under-served tenants.

E. Evaluation on a Physical Cluster

To validate practical deployability, we evaluated DeepShare on our Kubernetes-managed testbed. We ran 50 jobs following internal cluster patterns (1-GPU: 77%, 2-GPU: 15%, 4-GPU: 8%) and measured JCT, queueing delay, and makespan.

Fig. 12 presents the results under four configurations: **Hard** (fixed quota, no colocation), **Hard+Colocate** (fixed quota with colocation), **DeepShare** (DRA without colocation), and **DeepShare+Colocate** (full system).

With DRA alone, makespan decreased from 4,429.7 s to 3,368.8 s (24.0% reduction), average JCT from 1,479.5 s to 1,205.6 s (18.5% reduction), and queueing delay from 704.9 s to 444.2 s (37.0% reduction). With the full system (DRA + Colocation), compared with Hard+Colocate, makespan was reduced by 32%, average JCT by 34%, and queueing delay by 66% (from 690.9 s to 232.8 s). The full system processes the same batch of 50 jobs in 67.6% of the time required by the Hard baseline, corresponding to approximately 1.48$\times$ the throughput on the physical testbed.

The physical cluster improvements are slightly lower than simulation results, which is expected due to the smaller scale (16 GPUs vs. simulated hundreds) limiting the opportunities for elastic resource redistribution. Nevertheless, the consistent direction and magnitude of improvements across both environments confirm the robustness of DeepShare's design.

F. Sensitivity Analysis

We analyze the robustness of DeepShare to its two most influential parameters.

Dynamic tolerance baseline ($\rho_{\min}$). We varied $\rho_{\min}$ from 0.5 to 0.95 and measured GPU utilization and average per-job degradation of colocated jobs. At $\rho_{\min} = 0.5$ (aggressive), GPU utilization increases to 71.2% but average degradation reaches 18%, causing a net negative impact on JCT. At $\rho_{\min} = 0.7$ (default), the system achieves 70.58% utilization with $<$8% average degradation. At $\rho_{\min} = 0.9$ (conservative), utilization drops to 55.3% as colocation opportunities become too restricted. The default thus provides a robust balance between cluster efficiency and individual job impact.

Preemption cost weight (α). We varied α from 0 to 2.0. Higher values reduce repeated preemptions of the same job but may cause suboptimal resource reclamation. We find $\alpha \in [0.3, 0.8]$ consistently near-optimal across both datasets, with default $\alpha = 0.5$. The multi-victim penalty β exhibits similar robustness across $[0.1, 0.6]$.

G. Scalability and Limitations

Victim-set and colocation candidate ranking cost $O(n \log n)$ and $O(m \log m)$, respectively, while interference inference costs $O(1)$ per candidate pair. Extending DeepShare to heterogeneous accelerators requires retraining the interference model on per-device profiling data.

Our interference predictor is trained on the diverse profiling workloads summarized in Table II, covering computer vision, 3D perception, generative modeling, reinforcement learning, NLP, speech recognition, recommendation, and machine translation. Rather than relying on model-specific identifiers, it uses transferable DCGM hardware counters, such as SM activity and memory bandwidth. Variations in unseen architectures, training phases, batch sizes, and GPU types are therefore reflected in the runtime signals observed by the predictor. The physical testbed (16 GPUs) is smaller than production clusters; results are most directly applicable to departmental-scale clusters (tens to low hundreds of GPUs), while simulation on 23,859 jobs provides evidence at larger scale.

To guard against incorrect predictions, online retention validation continuously checks colocated pairs and revokes a colocation when the observed retention falls below the

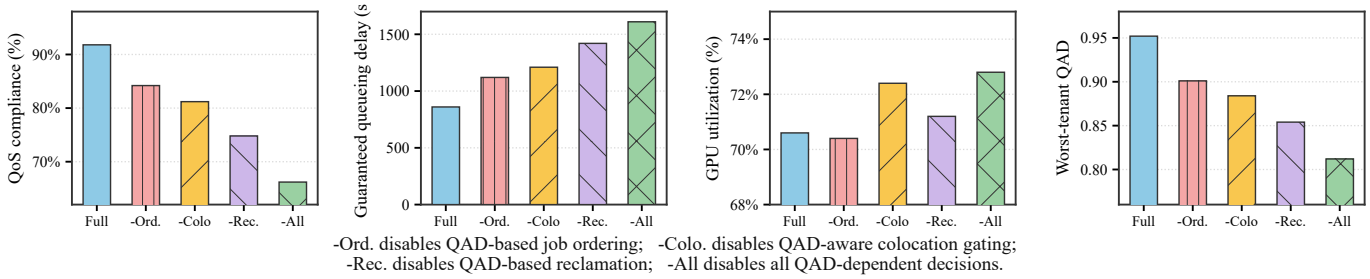


Fig. 11. QAD ablation across ordering, colocation gating, and reclamation.

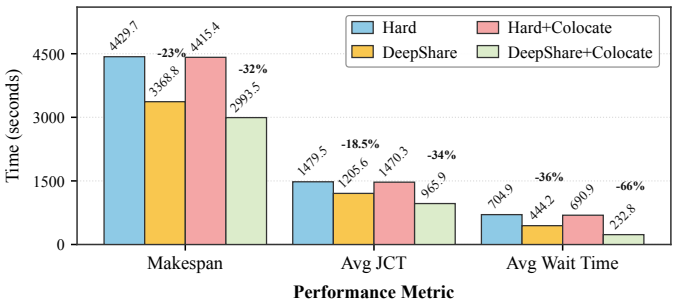


Fig. 12. Physical cluster results. DeepShare with colocation achieves the best performance across all metrics, confirming practical deployability.

admission threshold. This fallback affects only the reclaimable *best-effort* partner and does not compromise *guaranteed* jobs.

VI. Related Work

GPU cluster scheduling and workload characterization. Large-scale traces have revealed persistent underutilization in multi-tenant GPU clusters [9], [10], [14]. Scheduling policies address this from different angles: Tiresias [22] prioritizes jobs via a multi-level feedback queue without runtime knowledge; Chronus [45] targets deadline-aware scheduling; Pollux [46] co-adapts batch sizes and allocations for goodput maximization. Other recent policies target notebook reclamation [47], serverless ML [48], non-linear multi-GPU scalability [49], and heterogeneous GPU placement [50]. Maestro [51] further explores workload-aware cross-cluster scheduling for LLM-based multi-agent systems by jointly considering stage-level execution costs, model readiness, KV-cache feasibility, and network latency.

Interference-aware GPU sharing and colocation. Gandiva [20] and AntMan [8] pioneer DL job colocation with framework-level and asymmetric memory-scaling mechanisms, respectively. TGS [52] achieves transparent sharing via driver-level rate control, while Orion [15] partitions SMs and memory at the kernel level for inference. Other systems address GPU fraction management [53] and memory-layer multiplexing [54]. On the modeling side, SCHEDTUNE [55] profiles interference on heterogeneous GPUs, Jacquet et al. [56] characterize per-job GPU power under sharing, and ISACPP [32] advances interference prediction with graph attention networks at 50–200 ms latency per pair.

Fairness, quota management, preemption, and runtime prediction. Themis [57], ASTRAEA [58], and Shockwave [19] formalize various fairness objectives but do not differentiate production from *best-effort* workloads or integrate colocation. Gavel [17] optimizes max-min fairness over effective throughput but assumes exclusive GPU placement. HiveD [13] guarantees resources via static cell partitioning; Sia [18] optimizes goodput across heterogeneous GPUs but leaves colocation and quota out of scope. Other work addresses priority backfilling for HPC [59] and MPI process malleability [60]. In the Kubernetes ecosystem, the ElasticQuota plugin [24] and Volcano [25] provide namespace-level min/max quotas with binary reclamation. For preemption, REEF [61] targets microsecond kernel-level preemption for inference, GPREEMPT [62] generalizes GPU preemptive mechanisms, GFS [63] forecasts organizational demand using checkpoint recency, and Parcae [64] migrates LLM training proactively. On runtime prediction, Optimus [21] fits loss curves online and ElasticFlow [65] scales workers by predicted throughput.

VII. Conclusion

This paper presented DeepShare, an assurance-driven resource management framework for multi-tenant GPU clusters that coordinates elastic quota regulation, predictive scheduling, and interference-aware colocation through the Quota Assurance Degree $\tilde{Q}_i(t)$. Trace-driven simulation on 23,859 Venus jobs and 3,200 internal jobs shows 70.58% average GPU utilization (29.5% above Lucid) and 46% lower queueing delay than Lucid, while a 16-GPU Kubernetes deployment confirms a 34% JCT reduction and 93% tenant-cycle QoS compliance. Combining DRA with colocation further reduces queueing delay by 31% beyond DRA alone. Future work includes broader validation and model adaptation across heterogeneous GPU generations, as well as support for larger distributed training jobs.

Acknowledgment

This work was supported in part by National Key R&D Program of China (Grant No. 2024YFB4505604), in part by the National Natural Science Foundation of China (Grant No. 62402024), in part by Beijing Natural Science Foundation (No. L241050), in part by the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[2] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[5] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. Li, Y. Wu *et al.*, "Deepseekmath: Pushing the limits of mathematical reasoning in open language models," *arXiv preprint arXiv:2402.03300*, 2024.

[6] A. Merchant, S. Batzner, S. S. Schoenholz, M. Aykol, G. Cheon, and E. D. Cubuk, "Scaling deep learning for materials discovery," *Nature*, vol. 624, no. 7990, pp. 80–85, 2023.

[7] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15, 2015, pp. 1–17.

[8] W. Xiao, S. Ren, Y. Li, Y. Zhang, P. Hou, Z. Li, Y. Feng, W. Lin, and Y. Jia, "AntMan: Dynamic scaling on GPU clusters for deep learning," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 2020, pp. 533–548. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/xiao

[9] 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)*. USENIX Association, 2019, pp. 947–960. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/jeon

[10] 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)*. USENIX Association, 2022, pp. 945–960. [Online]. Available: https://www.usenix.org/conference/nsdi22/presentation/weng

[11] Q. Weng, L. Yang, Y. Yu, W. Wang, X. Tang, G. Yang, and L. Zhang, "Beware of fragmentation: Scheduling GPU-Sharing workloads with fragmentation gradient descent," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, 2023, pp. 995–1008. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/weng

[12] X. Sun, C. Hu, R. Yang, P. Garraghan, T. Wo, J. Xu, J. Zhu, and C. Li, "Rose: Cluster resource scheduling via speculative over-subscription," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 949–960.

[13] H. Zhao, Z. Han, Z. Yang, Q. Zhang, F. Yang, L. Zhou, M. Yang, F. C. Lau, Y. Wang, Y. Xiong, and B. Wang, "HiveD: Sharing a GPU cluster for deep learning with guarantees," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 2020, pp. 515–532. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/zhao-hanyu

[14] Q. Hu, Z. Ye, Z. Wang, G. Wang, M. Zhang, Q. Chen, P. Sun, D. Lin, X. Wang, Y. Luo, Y. Wen, and T. Zhang, "Characterization of large language model development in the datacenter," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 709–729. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/hu

[15] F. Strati, X. Ma, and A. Klimovic, "Orion: Interference-aware, fine-grained GPU sharing for ML applications," in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys '24, 2024, pp. 1075–1092.

[16] R. Gu, Y. Chen, S. Liu, H. Dai, G. Chen, K. Zhang, Y. Che, and Y. Huang, "Liquid: Intelligent resource estimation and network-efficient scheduling for deep learning jobs on distributed gpu clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2808–2820, 2022.

[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)*. USENIX Association, 2020, pp. 481–498. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak

[18] S. Jayaram Subramanya, D. Arfeen, S. Lin, A. Qiao, Z. Jia, and G. R. Ganger, "Sia: Heterogeneity-aware, goodput-optimized ML-cluster scheduling," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23, 2023, pp. 642–657.

[19] P. Zheng, R. Pan, T. Khan, S. Venkataraman, and A. Akella, "Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, 2023, pp. 703–723. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/zheng

[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)*. USENIX Association, 2018, pp. 595–610. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/xiao

[21] 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, pp. 1–14.

[22] 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)*. USENIX Association, 2019, pp. 485–500. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/gu

[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*, 2023, pp. 457–472.

[24] Kubernetes SIG Scheduling, "Kubernetes scheduler plugins: ElasticQuota," https://github.com/kubernetes-sigs/scheduler-plugins/tree/master/pkg/capacityscheduling, 2024, accessed: 2025-03-01.

[25] Volcano Community, "Volcano: Cloud native batch system for high-performance workloads," https://volcano.sh/, 2024, accessed: 2025-03-01.

[26] Q. Hu, P. Sun, S. Yan, Y. Wen, and T. Zhang, "Characterization and prediction of deep learning workloads in large-scale GPU datacenters," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, 2021, pp. 1–15.

[27] Z. Chen, W. Quan, M. Wen, J. Fang, J. Yu, C. Zhang, and L. Luo, "Deep learning research and development platform: Characterizing and scheduling with QoS guarantees on GPU clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 1, pp. 34–50, 2020.

[28] Z. Ye, W. Gao, Q. Hu, P. Sun, X. Wang, Y. Luo, T. Zhang, and Y. Wen, "Deep learning workload scheduling in GPU datacenters: A survey," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024.

[29] NVIDIA Corporation, *GPU Operator with MIG - NVIDIA Documentation Hub*, 2024. [Online]. Available: https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-operator-mig.html

[30] ——, *Multi-Process Service - NVIDIA Documentation Hub*, 2024. [Online]. Available: https://docs.nvidia.com/deploy/mps/index.html

[31] J. Liu, Z. Cai, Y. Liu, H. Li, Z. Zhang, R. Ma, and R. Buyya, "Smore: Enhancing gpu utilization in deep learning clusters by serverless-based co-location scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 5, pp. 903–917, 2025.

[32] Z. Liu, Y. Cheng, C. Chen, J. Hu, R. Fu, and D. Zhang, "ISACPP: Interference-aware scheduling approach for deep learning training workloads based on co-location performance prediction," *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 8, pp. 1591–1607, 2025.

[33] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet another resource negotiator," in *Proceedings of the 4th*

Annual Symposium on Cloud Computing, ser. SoCC '13. Association for Computing Machinery, 2013, pp. 1–16.

[34] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001. [Online]. Available: https://doi.org/10.1214/aos/1013203451

[35] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[36] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.

[37] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *International Conference on Learning Representations*, 2016.

[38] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 48, 2016, pp. 1928–1937.

[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[40] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *Proceedings of the 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 48, 2016, pp. 173–182.

[41] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 173–182.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[43] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention*, ser. Lecture Notes in Computer Science, vol. 9351. Springer, 2015, pp. 234–241.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] W. Gao, Z. Ye, P. Sun, Y. Wen, and T. Zhang, "Chronus: A novel deadline-aware scheduler for deep learning training jobs," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 609–623.

[46] 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, 2021, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/qiao

[47] B. Carver, J. Zhang, H. Wang, K. Mahadik, and Y. Cheng, "NotebookOS: A replicated notebook platform for interactive training with on-demand GPUs," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '26. Association for Computing Machinery, 2026, pp. 183–202.

[48] H. Wu, J. Deng, H. Fan, S. Ibrahim, S. Wu, and H. Jin, "QoS-Aware and cost-efficient dynamic resource allocation for serverless ML workflows," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2023, pp. 886–896.

[49] J. Han, M. M. Rafique, L. Xu, A. R. Butt, S.-H. Lim, and S. S. Vazhkudai, "MARBLE: A multi-GPU aware job scheduler for deep learning on HPC systems," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 272–281.

[50] S. Dongare, R. I. S. Khan, H. Albahar, N. Zhao, D. Meléndez-Maita, and A. R. Butt, "Hybrid learning and optimization-based dynamic scheduling for DL workloads on heterogeneous GPU clusters," in *Proceedings of the 2025 ACM Symposium on Cloud Computing (SoCC)*, ser. SoCC '25. Association for Computing Machinery, 2026, pp. 557–570.

[51] J. Wang, X. Zhou, X. Sun, Y. Zhang, Y. Li, T. Wo, X. Wang, C. Hu, and R. Yang, "Maestro: Workload-aware cross-cluster scheduling for llm-based multi-agent systems," *arXiv preprint arXiv:2606.12950*, 2026.

[52] B. Wu, Z. Zhang, Z. Bai, X. Liu, and X. Jin, "Transparent GPU sharing in container clouds for deep learning workloads," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, 2023, pp. 69–85. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/wu

[53] T.-A. Yeh, H.-H. Chen, and J. 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*, ser. HPDC '20, 2020, pp. 173–184.

[54] P. Yu and M. Chowdhury, "Salus: Fine-grained GPU sharing primitives for deep learning applications," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 2, 2020, pp. 98–111.

[55] H. Albahar, S. Dongare, Y. Du, N. Zhao, A. K. Paul, and A. R. Butt, "SCHEDTUNE: A heterogeneity-aware GPU scheduler for deep learning," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 695–705.

[56] P. Jacquet, M. Agusti, E. Caron, C. Coti, M. D. De Assunção, L. Lefèvre, and A.-C. Orgerie, "Untangling GPU power consumption: Job-level inference in cloud shared settings," in *Proceedings of the 21st European Conference on Computer Systems*. Association for Computing Machinery, 2026, pp. 624–640.

[57] 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)*. USENIX Association, 2020, pp. 289–304. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/mahajan

[58] Z. Ye, P. Sun, W. Gao, T. Zhang, X. Wang, S. Yan, and Y. Luo, "Astraea: A fair deep learning scheduler for multi-tenant gpu clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2781–2793, 2022.

[59] A. Gainaru, S. Klasky, and G. Pallez, "Priority-BF: a task manager for priority-based scheduling," in *Euro-Par 2025: Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Nature Switzerland, 2025, pp. 219–232.

[60] S. Iserte, I. Martín-Álvarez, K. Rojek, J. I. Aliaga, M. Castillo, W. Folwarska, and A. J. Peña, "Resource optimization with MPI process malleability for dynamic workloads in HPC clusters," *Future Generation Computer Systems*, vol. 174, p. 107949, 2026.

[61] M. Han, H. Zhang, R. Chen, and H. Chen, "Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, 2022, pp. 539–558. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/han

[62] R. Fan, T. Ren, M. Xie, S. Gao, J. Shu, and Y. Lu, "GPREEMPT: GPU preemptive scheduling made general and efficient," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. USENIX Association, 2025, pp. 263–272. [Online]. Available: https://www.usenix.org/conference/atc25/presentation/fan

[63] J. Duan, S. Xu, S. Qian, D. Yang, K. Wang, C. Liao, Y. Yu, Q. Hua, H. Hu, Q. Wang, W. Wu, D. Bao, T. Lu, J. Cao, G. Xue, G. Yang, L. Zhang, and G. Chen, "GFS: A preemption-aware scheduling framework for GPU clusters with predictive spot instance management," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. ACM, 2026, pp. 117–131.

[64] J. Duan, Z. Song, X. Miao, X. Xi, D. Lin, H. Xu, M. Zhang, and Z. Jia, "Parcae: Proactive, Liveput-Optimized DNN training on preemptible instances," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, 2024, pp. 1121–1139. [Online]. Available: https://www.usenix.org/conference/nsdi24/presentation/duan

[65] D. Gu, Y. Zhao, Y. Zhong, Y. Xiong, Z. Han, P. Cheng, F. Yang, G. Huang, X. Jin, and X. Liu, "ElasticFlow: An elastic serverless training platform for distributed deep learning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 266–280.