



SensaScale: A deadline-aware scheduler for DNN training tasks based on resource sensitivity

Kaimin Wu^a, Xinhua Wang^a, Zhiping Peng^b, Rong Xie^c, Weiwei Lin^{a,*}, Keqin Li^d

^a School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China

^b Wuyi University, Jiangmen, 529020, China

^c School of Big Data and Artificial Intelligence, University of Finance and Economics, Guangzhou, 510320, China

^d Department of Computer Science, State University of New York at New Paltz, New Paltz, 12561, USA

ARTICLE INFO

Keywords:

DNN training job scheduling
Auxiliary resource sensitivity
Deadline-aware scheduling
Multi-stage cooperative scheduling

ABSTRACT

The pervasive deployment of Deep Neural Networks (DNNs) across domains such as computer vision and natural language processing has established DNN training as a dominant workload in modern cloud data centers. Existing scheduling frameworks typically provision auxiliary resources—most notably CPU and memory—in proportion to GPU allocations. However, DNN workloads exhibit substantial heterogeneity in their sensitivity to auxiliary resources, resulting in frequent deadline violations. To address this inefficiency, we present SensaScale, a deadline-aware DNN scheduler that explicitly exploits auxiliary-resource sensitivity. SensaScale classifies jobs into sensitive and insensitive categories according to their performance dependence on CPU and memory, and orchestrates them through a multi-stage cooperative scheduling framework. Specifically, jobs are ordered by latest start time (LST), paired within each cohort based on resource complementarity, and scheduled using deadline-safe backfilling coupled with deadline-aware resource adjustment mechanisms. This design effectively enhances application performance. Experimental evaluations on Helios and Philly trace-based workloads under both 16-GPU-per-node and 8-GPU-per-node cluster settings show that SensaScale outperforms the Chronus baseline, reducing Service Level Objective (SLO) violation rates by an average of 74.22%, average job completion time (JCT) by 25.86%, and average queuing time (AQT) by 82.75%.

1. Introduction

The rapid proliferation of Deep Neural Networks (DNNs) in mission-critical domains — including computer vision [1], natural language processing [2], and speech recognition [3] — has made DNN training a foundational workload in both enterprise and cloud data centers. Owing to its substantial computational intensity and prolonged execution time, DNN training is typically deployed on large-scale, multi-tenant cluster infrastructures provisioned with high-performance accelerators such as GPUs. Such architectures facilitate efficient resource sharing among multiple tenants and production teams, thereby improving overall cluster utilization [4].

Current DNN training workloads are often subject to strict deadline requirements [5]. During scheduling, jobs specify GPU resource needs and are assigned to servers only when these criteria are met. Existing scheduling approaches fall into two broad categories: those based on traditional big data systems such as Kubernetes [6] and YARN [7], and those that leverage GPU schedulers tailored to DNN

job characteristics [8,9]. The latter category incorporates specialized scheduling algorithms to improve resource utilization and optimize performance metrics such as JCT, AQT, and deadline satisfaction. Representative deadline-aware schedulers have demonstrated significant improvements in reducing deadline violations through job prioritization and deadline-driven resource management [5]. However, most existing deadline-aware schedulers continue to treat GPUs as the dominant resource and allocate auxiliary resources such as CPU and memory proportionally to GPU assignments.

Recent studies have shown that such GPU-proportional allocation can be suboptimal for DNN training workloads. A substantial portion of training execution time is often spent on data ingestion and pre-processing, which heavily rely on CPU and memory resources [10,11]. Consequently, DNN jobs exhibit heterogeneous sensitivity to auxiliary resources. This phenomenon has been observed not only in academic studies but also in production-scale deployments at companies such as Google [12] and Meta [13]. Resource-sensitive schedulers such as Synergy [14] demonstrate that allocating CPU and memory resources

* Corresponding authors.

E-mail addresses: 202421044148@mail.scut.edu.cn (K. Wu), 202111088204@mail.scut.edu.cn (X. Wang), pzhp@wyu.edu.cn (Z. Peng), xierong@gdufe.edu.cn (R. Xie), linww@scut.edu.cn (W. Lin), lik@newpaltz.edu (K. Li).

<https://doi.org/10.1016/j.future.2026.108820>

Received 22 March 2026; Received in revised form 11 September 2026; Accepted 12 September 2026

Available online 23 September 2026

0167-739X/© 2026 Elsevier B.V. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

according to job-specific sensitivity can significantly improve cluster throughput and reduce JCT. For example, image and video processing models may achieve substantial performance gains when allocated additional CPU resources, whereas models such as GNMT [15] exhibit little performance degradation even when auxiliary-resource allocations are reduced below the GPU-proportional level. These findings indicate that auxiliary resources should not necessarily be coupled rigidly with GPU allocations.

Despite these advances, an important gap remains between existing deadline-aware schedulers and resource-sensitive schedulers. Deadline-aware schedulers primarily focus on meeting job deadlines but generally rely on GPU-proportional auxiliary-resource allocation, thereby overlooking resource-sensitivity heterogeneity. In contrast, resource-sensitive schedulers such as Synergy optimize throughput-oriented objectives, including cluster efficiency and JCT, without explicitly considering deadline constraints or SLO satisfaction. Consequently, it remains unclear how resource sensitivity can be effectively exploited under deadline-constrained DNN training workloads, where scheduling decisions must simultaneously account for job urgency and deadline satisfaction.

Motivated by this observation, we investigate the problem of deadline-aware resource-sensitive scheduling for DNN training workloads. Our key insight is that auxiliary resources should be allocated according to both resource sensitivity and deadline urgency rather than being statically tied to GPU allocations. By jointly considering these factors, a scheduler can better match auxiliary resource allocation with job requirements and improve the likelihood that jobs complete before their deadlines.

To this end, we propose SensaScale, a deadline-aware and resource-sensitive scheduler for DNN training clusters. SensaScale explicitly exploits cross-job heterogeneity in CPU and memory sensitivity while incorporating deadline awareness into scheduling decisions. It dynamically groups jobs according to their auxiliary-resource sensitivity and constructs complementary scheduling units to better align auxiliary resource allocation with diverse job requirements. Furthermore, SensaScale continuously monitors job progress and performs deadline-aware resource adjustment when deadline-critical jobs are detected. By combining resource sensitivity with deadline-aware scheduling, SensaScale improves deadline satisfaction while ensuring adaptive and effective auxiliary-resource allocation.

The main contributions of this work are summarized as follows:

- We identify a previously unexplored gap between deadline-aware scheduling and resource-sensitive scheduling for DNN training workloads, and formulate the problem of deadline-aware resource-sensitive scheduling that jointly considers deadline urgency and auxiliary-resource sensitivity.
- We propose SensaScale, a deadline-aware and resource-sensitive scheduler that integrates sensitivity-aware job grouping, complementary resource matching, and adaptive deadline-aware resource adjustment to improve SLO satisfaction under multi-resource cluster constraints.
- We implement SensaScale and conduct extensive trace-driven evaluations using realistic DNN training workloads. Experimental results demonstrate that SensaScale substantially reduces SLO violations while simultaneously improving JCT and queueing performance across diverse workload compositions and cluster-load conditions.

2. Background and motivation

In this section, we review the background of resource scheduling for DNN training, formalize key concepts including auxiliary resource sensitivity and optimal demand, and analyze the principal limitations of existing scheduling approaches. Building on this analysis, we introduce a resource-sensitive cluster scheduling framework designed to improve deadline compliance.

2.1. Background

GPU-Proportional Scheduling: In DNN training, data processing typically follows a three-stage pipeline: data loading and caching, CPU-based preprocessing, and GPU computation. To simplify resource management, existing DNN schedulers [16,17] commonly adopt a GPU-centric proportional allocation strategy. Under this paradigm, GPUs are treated as the dominant resource, and auxiliary resources — such as CPU and memory — are provisioned proportionally to GPU requests. For instance, on a server equipped with 16 GPUs, 48 CPU cores, and 1000 GB of memory, a request for one GPU would be accompanied by an automatic allocation of 3 CPU cores and 62.5 GB of memory. Although straightforward to implement, this strategy overlooks the heterogeneous non-GPU resource demands of different models, potentially leading to suboptimal auxiliary-resource allocation and reduced deadline satisfaction.

Auxiliary Resource Sensitivity: Here, auxiliary resources refer specifically to CPU and memory. DNN training jobs exhibit heterogeneous sensitivity to these resources, rooted in the structure of the training pipeline. A typical workflow loads data from storage into memory, performs preprocessing and augmentation on the CPU, and then executes forward and backward propagation on the GPU. Within this pipeline, preprocessing throughput (e.g., image decoding and augmentation) is directly constrained by CPU compute capacity and core count. When CPU resources become a bottleneck, GPUs idle while awaiting data, substantially degrading overall training efficiency. Memory capacity governs the volume of data that can be cached in advance. Adequate memory enables larger prefetch buffers and reduces I/O stalls, whereas insufficient memory induces frequent data movement and elevated latency. Consequently, for preprocessing-intensive workloads (e.g., image or video recognition), increasing CPU and memory allocations can markedly accelerate training. In contrast, for compute-intensive workloads such as GNMT [15], reducing auxiliary resources exerts minimal impact on performance. Exploiting this heterogeneity in auxiliary resource sensitivity is therefore central to improving cluster-level efficiency [14].

Optimal Demand: Synergy [14] systematically models the sensitivity of DNN workloads to auxiliary resources (CPU and memory) through an optimistic analytical framework. Under a fixed GPU allocation, rather than exhaustively enumerating all CPU-memory configurations, the method constructs a performance characterization surface that maps job throughput to auxiliary resource provisioning. Concretely, it profiles throughput across varying CPU allocations (with memory provisioned at its maximum) and then infers the complete surface via analytical modeling. Based on this characterization, the optimal resource demand is defined as the minimal auxiliary resource combination that saturates throughput under a given GPU allocation—that is, the point beyond which additional CPU or memory yields diminishing returns. Identifying this operating point suggests that allocations below the threshold may lead to throughput degradation.

2.2. Motivation

Core Insight: The central observation motivating this work is that DNN training workloads exhibit pronounced heterogeneity in their sensitivity to CPU and memory provisioning. Specifically, certain jobs achieve measurable acceleration when allocated auxiliary resources beyond their GPU-proportional share, whereas others maintain stable performance even when provisioned with less than the proportional allocation. Building on this insight, we advocate replacing the conventional GPU-proportional policy with a resource-sensitivity-driven dynamic allocation framework. By aligning auxiliary resource provisioning with job-specific sensitivity, this approach seeks to enhance

Server	Job	GPU	CPU	Mem
Server ₁	Alexnet (J_1)	2	6	125GB
	Audio-M5 (J_2)	2	6	125GB
	Deepspeech (J_3)	2	6	125GB
Server ₂	GNMT (J_4)	2	6	125GB
	Deepspeech (J_3)	2	6	125GB
	GNMT (J_4)	2	6	125GB

Fig. 1. Resource allocation.

Table 1

Comparison of job execution time under different allocation strategies.

	Alexnet	Audio-M5	Deepspeech	GNMT
GPU-proportional allocation(s)	2531.2	1687.5	2334.8	2025.0
Optimal demand allocation(s)	1087.4	675.0	2334.8	2025.0

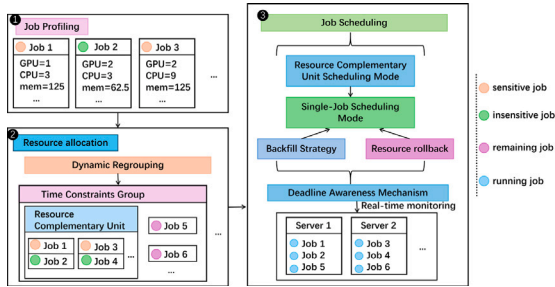


Fig. 2. SensaScale scheduler workflow.

overall cluster utilization and, consequently, improve deadline compliance. This perspective is consistent with empirical evidence from the Microsoft Philly cluster [18], which reports substantial underutilization of CPU cycles in multi-tenant settings. Collectively, these findings suggest that exploiting cross-job heterogeneity in resource demand can improve deadline satisfaction without necessitating hardware expansion.

Key Point: We pair jobs with high auxiliary-resource demand with those exhibiting low demand and co-schedule them on the same server, thereby enabling demand-aware scheduling in place of rigid GPU-proportional allocation. When such complementary pairing is not feasible, we adopt a single-job scheduling strategy that permits disproportionate, resource-sensitive auxiliary resource provisioning. This flexibility accelerates job execution and improve deadline satisfaction.

Example: We illustrate the efficiency gains of resource-sensitivity-aware scheduling through a simplified example. Consider two servers, each provisioned with 4 GPUs, 12 CPU cores, and 250 GB of DRAM. The scheduling queue contains four jobs, each requesting 2 GPUs. Fig. 1 (left) depicts GPU-proportional allocation, whereas Fig. 1 (right) presents the corresponding optimal-demand allocation. As summarized in Table 1, execution times differ across allocation strategies. Under the optimal demand-aware policy, high auxiliary-resource-demand jobs are co-scheduled with low-demand counterparts—specifically, pairing J_1 with J_3 and J_2 with J_4 . Allocating additional CPU and memory to the high-demand jobs (J_1 and J_2) markedly accelerates their execution, while the performance of low-demand jobs (J_3 and J_4) remains unaffected.

2.3. Challenges

This work investigates the principled allocation of CPU and memory resources for deadline-constrained DNN workloads. Several challenges emerge. First, relative to conventional GPU-proportional policies, how

can auxiliary resources be provisioned without degrading the execution speed of any job? Second, given heterogeneous sensitivity to auxiliary resources, how can one design an auxiliary-resource-aware scheduler for DNN training that efficiently packs jobs across multiple resource dimensions while improving deadline satisfaction?

Fairness: Resource-skewed allocation may introduce fairness concerns. To mitigate this risk, SensaScale is designed to maintain throughput comparable to or higher than that achieved under conventional GPU-proportional allocation. Leveraging the optimal CPU and memory configurations identified by Synergy [14], the scheduler preferentially provisions high auxiliary-resource-demand jobs according to their optimal demand. When auxiliary capacity is insufficient, it reverts to GPU-proportional allocation. For low-demand jobs, scheduling likewise follows their respective optimal demand profiles. To avoid pathological cases in which GPU requirements are satisfied but auxiliary resources are monopolized by high-demand jobs, SensaScale incorporates a resource rollback mechanism. This mechanism reclaims auxiliary resources exceeding the GPU-proportional share from over-provisioned jobs, thereby enabling the admission and scheduling of pending jobs.

Scheduling Mechanism: The conventional GPU-proportional policy results in resource underutilization for jobs insensitive to auxiliary provisioning, thereby diminishing overall cluster efficiency. Moreover, because each job operates under a deadline constraint, effective scheduling requires fine-grained, workload-aware allocation rather than a uniform, coarse-grained policy. To address these limitations, SensaScale exploits heterogeneity in auxiliary-resource sensitivity by grouping jobs accordingly and co-scheduling high-demand jobs with low-demand counterparts. Following group-level placement, single-job scheduling decisions are made based on real-time node resource availability and the job's sensitivity class. The scheduler further integrates a deadline-safe backfilling strategy to facilitate efficient resource reuse while maintaining scheduling performance. Finally, to improve deadline satisfaction, SensaScale incorporates a deadline-aware control mechanism that continuously monitors execution progress and dynamically adjusts resource allocations upon detecting potential deadline violations, thereby accelerating job completion.

3. Design

We present SensaScale, a scheduler for DNN training workloads that exploits job-level sensitivity to auxiliary resources to enable principled multi-resource allocation and improve deadline satisfaction. Existing GPU-proportional schedulers overlook the heterogeneous sensitivity of DNN jobs to auxiliary resources such as CPU and memory, leading to increased deadline violations. SensaScale addresses this limitation by jointly considering (i) auxiliary-resource sensitivity and (ii) deadline urgency, enabling coordinated scheduling decisions that improve SLO satisfaction. Section 3.1 provides a system overview, Section 3.2 describes the job profiling, Section 3.3 outlines the initial resource allocation strategy, Section 3.4 details the scheduling procedure, Section 3.5 summarizes the overall scheduling workflow, and Section 3.6 discusses the implementation details.

3.1. Overview

The SensaScale workflow, depicted in Fig. 2, comprises three principal modules. (1) Job Profiling. Upon submission, each DNN training job undergoes an online profiling phase to evaluate its sensitivity to CPU and memory under varying auxiliary resource configurations. Based on this assessment, jobs are classified as either sensitive or insensitive, and this classification guides subsequent resource allocation decisions. (2) Resource Allocation. Leveraging the profiling results, SensaScale applies a dynamic regrouping strategy that pairs high auxiliary-resource-demand jobs with low-demand counterparts to form resource-complementary units. This step establishes preliminary CPU

and memory allocations for each job. (3) Job Scheduling. Guided by the constructed complementary units and any remaining jobs, the scheduler executes placement decisions. During scheduling, it adaptively employs backfilling based on real-time resource availability and integrates a deadline-aware resource adjustment mechanism to further improve deadline satisfaction.

3.2. Job profiling

Before scheduling, each DNN training job is associated with workload metadata, including arrival time, GPU requirement, runtime information, deadline, optimal auxiliary-resource demand, speedup information, and resource-sensitivity class. Among these attributes, arrival time, GPU requirement, and deadline are obtained from job submission information, while runtime information is obtained from available job execution information. The remaining metadata related to auxiliary resources is generated through online profiling after job submission.

Specifically, SensaScale adopts the sensitivity-analysis methodology implemented in the Synergy codebase [14] to characterize the auxiliary-resource requirements of DNN training jobs. The profiling evaluates the performance impact of different CPU and memory allocations under a fixed GPU configuration. The optimal auxiliary-resource demand is identified as the CPU and memory configuration achieving the best observed performance, while the speedup information represents the performance improvement compared with GPU-proportional auxiliary-resource allocation. These profiling results are further used to determine the resource-sensitivity class of each job and provided as metadata to the scheduler.

SensaScale follows the optimistic profiling methodology of Synergy [14], which efficiently reduces the profiling cost. In practice, profiling a single DNN training job can typically be completed within several minutes to tens of minutes (e.g., approximately 8 min for ResNet18). Since profiling is performed only once during job characterization and DNN training jobs usually have long execution durations in production GPU clusters [18,19], the overhead can be effectively amortized and does not introduce additional latency during online scheduling.

Based on auxiliary-resource sensitivity under fixed GPU allocation, jobs are classified into two categories. Sensitive jobs exhibit measurable performance gains when provisioned with auxiliary resources beyond the GPU-proportional share, whereas insensitive jobs sustain their baseline throughput even when allocated less than the proportional share.

3.3. Resource allocation

Dynamic Regrouping Strategy: To more effectively utilize auxiliary resources and improve deadline satisfaction, we introduce a dynamic regrouping strategy that pairs sensitive and insensitive jobs for coordinated scheduling. By aligning complementary resource demands, this approach accelerates job completion through judicious auxiliary resource provisioning. The regrouping process considers both unfinished running jobs and newly arrived but unscheduled jobs. These jobs are first sorted in ascending order of their LST (T_l) and then partitioned into consecutive groups according to the specified group size. For each job, T_l is derived from its execution time (T_e), deadline (T_d), and arrival time (T_a), as defined in Eq. (1).

$$T_l = \begin{cases} T_d - T_e, & T_a \leq T_d - T_e \\ T_a, & T_a > T_d - T_e \end{cases} \quad (1)$$

Within each deadline-constrained group, sensitive and insensitive jobs are paired according to a complementarity score to form resource-complementary units. The objective is to align the combined actual CPU and memory demands of each pair as closely as possible with the ideal aggregate under GPU-proportional allocation. Formally, the

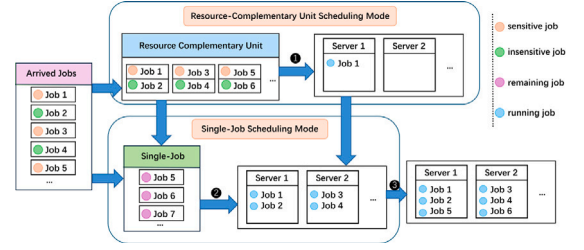


Fig. 3. Overview of the two-phase scheduling mechanism in SensaScale.

complementarity score is computed by jointly considering the GPU-proportional resource requirements (cpu_i, mem_i) and the optimal resource demands (cpu_i^*, mem_i^*) of the paired jobs. Here, cpu_i and mem_i denote the total CPU and memory demand under GPU-proportional allocation, whereas cpu_i^* and mem_i^* represent the total demand under optimal allocation. To eliminate dimensional inconsistency between CPU and memory resources, the corresponding deviations are normalized by the aggregate GPU-proportional CPU and memory demands (cpu_i, mem_i), respectively. The complementarity score is then calculated as Eq. (2), where ϵ is a small smoothing coefficient (1×10^{-9}). A larger score indicates a smaller normalized deviation between optimal and GPU-proportional resource demands, implying stronger resource complementarity between the paired jobs. Jobs within each deadline-constrained group are ranked by their complementarity scores and paired accordingly to construct resource-complementary units.

$$score = \frac{1}{\frac{|cpu_i - cpu_i^*|}{cpu_i} + \frac{|mem_i - mem_i^*|}{mem_i} + \epsilon} \quad (2)$$

Differentiated Resource Allocation for Jobs: Jobs within each resource-complementary unit are provisioned with CPU and memory according to their respective optimal demands. For remaining jobs, SensaScale applies a differentiated allocation policy based on job type. Insensitive jobs are directly assigned auxiliary resources consistent with their optimal demand profiles. In contrast, sensitive jobs are dynamically provisioned: when sufficient auxiliary capacity is available on the target node, they receive optimal-demand allocation; otherwise, the scheduler reverts to GPU-proportional allocation.

3.4. Job scheduling

The scheduler performs multi-resource allocation in which GPU demand is fixed per job, whereas auxiliary resource provisioning remains elastic. As shown in Fig. 3, SensaScale prioritizes the scheduling of resource-complementary units, allocating CPU and memory according to their optimal demand profiles to expedite completion. Once all feasible complementary units have been scheduled, any remaining arrived but unscheduled jobs are handled in a single-job scheduling mode. In this phase, pending jobs are ordered by LST, followed by per-job resource assignment. The elasticity of auxiliary resource allocation introduces a potential imbalance: auxiliary capacity may be exhausted while GPUs remain underutilized. To mitigate this risk, SensaScale incorporates a resource rollback mechanism. In addition, the scheduler integrates a backfilling strategy and a deadline-aware control mechanism to further improve deadline satisfaction.

3.4.1. Resource-complementary unit scheduling mode

The resource-complementary unit scheduling mode adopts a GPU-demand-based priority scheme. Within each deadline-constrained group, complementary units are ordered in descending order of their aggregate GPU demand. The scheduler then attempts allocation sequentially following this priority. During placement, CPU and memory resources for each complementary unit are provisioned according to

their optimal demand configurations, and all constituent jobs must be co-located on the same compute node.

Based on the scheduling status of jobs within a resource-complementary unit, two typical scenarios arise: (1) All jobs unscheduled. If none of the jobs in the complementary unit has been placed, the unit may be assigned to any compute node that satisfies its aggregate resource requirements. (2) One job already running. If one job in the complementary unit has been scheduled but not yet completed, the remaining job must be co-located on the same compute node as the running job.

3.4.2. Single-job scheduling mode

Scheduling Mechanism: The single-job scheduling mode is invoked after all feasible resource-complementary units have been processed, applying differentiated policies to the remaining jobs. In this phase, jobs are scheduled according to their sensitivity classification. (1) Insensitive jobs. These jobs are provisioned based on their optimal demand profiles. Servers are ranked by auxiliary resource pressure, quantified by CPU and memory utilization, with priority assigned to nodes exhibiting higher auxiliary load. (2) Sensitive jobs. The scheduler first attempts to allocate resources according to optimal demand on nodes with sufficient auxiliary capacity. If such placement is infeasible, it reverts to GPU-proportional allocation.

Resource Rollback Mechanism: If a job cannot be scheduled despite sufficient GPU availability, but fails due to unmet auxiliary resource requirements, this indicates that on a GPU-eligible server at least one running job occupies auxiliary resources beyond its GPU-proportional share. In such cases, the scheduler activates the resource rollback mechanism on that node, reclaiming a portion of the over-provisioned auxiliary resources to admit the pending job.

This mechanism is activated exclusively in single-job scheduling mode and aims to prevent GPU idleness resulting from imbalanced auxiliary resource allocation. The scheduler first identifies all sensitive jobs currently running on the server under optimal-demand provisioning as rollback candidates. For each candidate, it estimates the expected runtime increase incurred when reducing allocation from optimal demand to GPU-proportional provisioning. Rollback priority is assigned to jobs exhibiting a larger margin between their adjusted runtime and deadline, thereby minimizing the risk of deadline violation. The mechanism iteratively reclaims auxiliary resources until the server can satisfy the pending job's requirements. Once sufficient capacity is restored, the remaining sensitive jobs retain their original allocations, and the rollback process terminates.

Backfilling Strategy: If a job cannot be scheduled due to insufficient GPU resources, the scheduler activates a backfilling strategy to maximize GPU utilization and improve the likelihood of meeting job deadlines.

During single-job scheduling, when the current job cannot be admitted due to resource constraints, a backfilling strategy is employed to enable subsequent jobs in the waiting queue to utilize idle resources effectively. Backfilling is activated exclusively in single-job scheduling mode and differentiates between urgent and non-urgent scenarios. Urgent scenario. This arises when at least one pending job has reached its LST. The scheduler prioritizes urgent jobs for admission. If an urgent job remains unschedulable, urgent backfilling is triggered. The scheduler estimates the earliest time at which each server can accommodate the urgent job and identifies the idle window spanning from the current time to that predicted start time. It then selects and backfills the job whose resource demand best matches the available capacity within this window. For servers that are not required to host urgent jobs, scheduling proceeds directly according to LST order. Non-urgent scenario. This corresponds to situations in which no pending job has yet reached its LST. In this case, jobs are processed sequentially. If a scheduling attempt fails, the scheduler performs backfilling by selecting

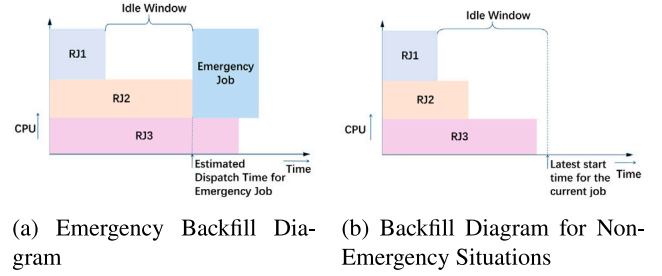


Fig. 4. Backfill strategy diagram.

a single compatible job that can be completed within the interval between the current time and the pending job's LST, thereby reducing resource contention while avoiding potential deadline violations (see Fig. 4).

3.4.3. Deadline-aware resource adjustment

During scheduling, because the allocator does not strictly enforce GPU-proportional provisioning of auxiliary resources, certain servers may retain partially idle CPU and memory capacity. To exploit these residual resources and further improve deadline satisfaction, we incorporate a deadline-aware resource adjustment mechanism. This mechanism continuously monitors the deadline-critical status of running jobs, with particular emphasis on sensitive jobs currently operating under GPU-proportional auxiliary resource allocation. When a job is identified as entering a deadline-critical state, the scheduler dynamically upgrades its auxiliary resource allocation to the optimal-demand configuration to accelerate its completion. The deadline-critical condition is determined according to Eq. (3), where S indicates whether the job is in a deadline-critical state, t_r denotes the remaining execution time, a is a sensitivity coefficient, and t_g represents the time gap between the current moment and the job's deadline.

$$S = \begin{cases} \text{yes} & (t_r \times a > t_g) \\ \text{no} & (\text{else}) \end{cases} \quad (3)$$

3.5. Overall scheduling procedure

The mechanisms presented in Sections 3.3 and 3.4 collectively form the complete scheduling workflow of SensaScale. Algorithm 1 provides a summary of the overall scheduling procedure and the interactions among the different system components.

Given a job set $\mathcal{J}$ and the current cluster state C , SensaScale first computes the LST (T_l) of each job based on its arrival time, runtime, and deadline, and then sorts jobs accordingly. The ordered jobs are then partitioned into consecutive groups according to the specified group size (Lines 1–4). Within each group $g \in \mathcal{G}$, jobs are divided into sensitive and insensitive subsets according to their auxiliary-resource sensitivity profiles. Resource-complementary units $\mathcal{U}$ are then constructed by pairing sensitive and insensitive jobs using the complementarity score, which measures the difference between GPU-proportional and optimal auxiliary-resource demands (Lines 5–12). The scheduler next prioritizes the placement of resource-complementary units according to their aggregate GPU demand. After all feasible units have been processed, the remaining jobs are scheduled individually using differentiated auxiliary-resource allocation policies. Insensitive jobs are assigned their optimal-demand resources directly, whereas sensitive jobs receive either optimal-demand or GPU-proportional allocation depending on resource availability (Lines 13–22). If a job cannot be admitted despite sufficient GPU capacity due to auxiliary-resource contention, the rollback mechanism is triggered to reclaim over-provisioned CPU and memory resources from selected running jobs (Lines 23–25). To further improve resource utilization, SensaScale applies backfilling when idle

Algorithm 1: SensaScale Scheduling Procedure

Input: Job set $\mathcal{J}$, cluster state $\mathcal{C}$
Output: Scheduling decisions $\mathcal{A}_{\mathcal{J}}$

foreach $j \in \mathcal{J}$ **do**

Compute $T_l = \begin{cases} T_d - T_e, & T_a \leq T_d - T_e \\ T_a, & \text{otherwise} \end{cases}$

Sort $\mathcal{J}$ by ascending T_l ;
 Partition the sorted jobs into consecutive groups according to the specified group size, obtaining $\mathcal{G}$;

foreach $g \in \mathcal{G}$ **do**

Partition jobs into S_g and I_g ;
foreach $j_s \in S_g$ **do**

Compute $score = \frac{1}{\frac{|cpu_i - cpu_j|}{cpu_i} + \frac{|mem_i - mem_j|}{mem_i} + \epsilon}$;
 Select $j_i = \arg \max_{j \in I_g} score$;
 Construct unit $u = (j_s, j_i)$;
 Add u to $\mathcal{U}$;

Sort $\mathcal{U}$ by aggregate GPU demand;

foreach $u \in \mathcal{U}$ **do**

if u is feasible **then**
 Schedule u ;

foreach $j \in (\mathcal{J} \setminus \mathcal{U})$ **do**

if j is insensitive **then**
 Allocate optimal-demand resources;

else
if optimal-demand allocation feasible **then**
 Allocate optimal-demand resources;
else
 Allocate GPU-proportional resources;

if GPU available **and** auxiliary resources insufficient **then**
 Trigger rollback;

Schedule j ;

if idle GPUs exist **then**
 Perform backfilling;

foreach running job j **do**

if $t_r \times a > t_g$ **then**
 Upgrade auxiliary resources to optimal demand;

return $\mathcal{A}_{\mathcal{J}}$

GPU resources are available (Lines 26–27). Finally, the scheduler continuously monitors running jobs and upgrades auxiliary-resource allocations for deadline-critical jobs, thereby reducing the risk of deadline violations (Lines 28–31).

3.6. Implementation

SensaScale advances the scheduling process in a time-driven manner, where the simulation clock progresses at a one-second interval. At each time step, the scheduler updates the global cluster state, including job arrivals, job completions, resource availability, and deadline-related information, and then performs scheduling decisions according to the proposed scheduling policy. For jobs with identical LST values, earlier-arriving jobs are prioritized. During pairing, candidate pairs are ranked by complementarity score, with ties broken by aggregate GPU demand and then arrival time. Let n and m denote the numbers of jobs and servers, respectively. The dominant computational cost of SensaScale comes from the resource-complementary pairing and resource allocation procedures, resulting in an overall complexity of $O(n^2 + nm)$.

4. Evaluation

In this section, we conduct trace-driven simulations using production cluster traces to systematically evaluate the effectiveness, robustness, and practicality of the SensaScale scheduler. The evaluation is designed to answer the following key research questions.

- **Overall Effectiveness and Generalization:** Compared with existing schedulers, how effectively can SensaScale improve deadline satisfaction under different workload compositions, job categories, and GPU cluster configurations?
- **Effect of Deadline-Constrained Group Size:** How does the size of deadline-constrained groups influence the scheduling performance of SensaScale?
- **Role of the Sensitivity Coefficient a in Deadline Awareness:** How does the sensitivity coefficient a within the deadline-aware resource adjustment mechanism affect the scheduling effectiveness of SensaScale?
- **Contribution of Core Components:** What are the individual contributions of the core components of SensaScale to the overall scheduling performance?
- **Runtime Overhead Analysis:** What runtime overhead does SensaScale introduce during online scheduling?
- **Robustness under Stochastic Workload Generation:** How robust is SensaScale under stochastic workload conditions with different workload realizations caused by random job arrivals and deadline assignments?

4.1. Experimental setup

Simulation Platform. All experiments are conducted using a high-fidelity trace-driven simulator built upon the Synergy scheduling framework [14]. The simulator replays production workload traces, maintains GPU cluster states, processes job arrivals and completions, and executes scheduling decisions according to the evaluated policies. We extend the original Synergy simulator with deadline-aware scheduling capabilities, including deadline-constrained grouping, resource-complementary allocation, rollback, backfilling, and deadline-aware resource adjustment, to evaluate SensaScale under deadline-constrained DNN training workloads.

Clusters. We simulate homogeneous GPU clusters with two different node configurations to evaluate the effectiveness and robustness of SensaScale. The first configuration follows the conventional large-node GPU cluster setting, where each node is equipped with 16 NVIDIA V100 GPUs, 48 CPU cores, and 1000 GB of DRAM. Specifically, we simulate two clusters consisting of 67 and 32 nodes, denoted as C67 and C32, respectively. To further evaluate the performance of SensaScale on modern GPU cluster configurations with more abundant auxiliary resources relative to GPUs, we additionally construct 8-GPU-per-node clusters. In this configuration, each node contains 8 NVIDIA V100 GPUs, 96 CPU cores, and 500 GB of DRAM. The number of nodes is doubled compared with the original configuration to maintain a comparable total GPU capacity, resulting in 134 nodes and 64 nodes for the two clusters, respectively. Compared with the 16-GPU-per-node configuration, the auxiliary-resource-to-GPU ratio is increased, with the CPU-to-GPU ratio rising from 3:1 to 12:1.

The 16-GPU-per-node clusters are used for the primary evaluation, while the 8-GPU-per-node clusters are used to evaluate the adaptability of SensaScale under configurations with more abundant auxiliary resources relative to GPUs and smaller GPU granularity.

Models. Based on the Synergy-based auxiliary-resource sensitivity analysis methodology [14], we perform profiling for representative DNN models to obtain their optimal auxiliary-resource requirements under fixed GPU configurations. The sensitivity analysis is conducted under a fixed auxiliary-resource-to-GPU ratio of 3 CPU cores per GPU

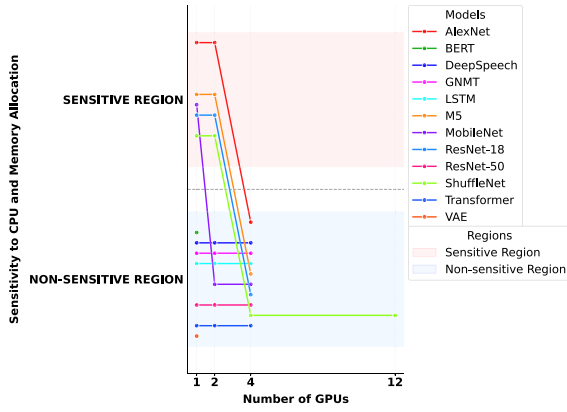


Fig. 5. DNN models sensitivity classification based on CPU/Memory allocation.

and 62.5 GB of memory per GPU. Our analysis includes 12 representative DNN models, as illustrated in Fig. 5. We use these profiling results to characterize the auxiliary-resource sensitivity of different DNN workloads. According to their auxiliary-resource demands and sensitivity characteristics, jobs are classified into two categories:

Based on the auxiliary-resource sensitivity analysis provided by the Synergy codebase [14], we derive the optimal auxiliary-resource requirements for each job type under a fixed GPU configuration. The Synergy profiling study [14] encompasses 12 representative DNN models, as illustrated in Fig. 5. According to their auxiliary-resource demands, jobs are classified into two categories:

- Sensitive jobs: including AlexNet [20], M5 [21], ResNet-18 [22], and ShuffleNet [23] under both single-GPU and dual-GPU configurations, as well as MobileNetV2 [24] under the single-GPU configuration.
- Insensitive jobs: including BERT [25], DeepSpeech [26], GNMT [15], LSTM [27], ResNet-50 [22], Transformer [2], and Variational Autoencoder [28] under 1-GPU, 2-GPU, and 4-GPU configurations, as well as AlexNet, M5, ResNet-18, ShuffleNet, and MobileNetV2 under 4-GPU configurations.

Traces. We conduct simulations using two real-world DLT job traces: the Helios trace and the Philly trace. The Helios trace was collected from a production cluster at SenseTime [19], covering a 14-day period from April 14 to 27, 2020. The Philly trace was obtained from a Microsoft data center [18], spanning October 12 to 25, 2017.

The evaluation workloads are constructed by combining trace-derived job characteristics, synthetic workload generation, and profiling information. Specifically, GPU demands and execution durations are obtained directly from the traces, while job arrivals are generated according to a Poisson process with rate parameter λ . Sensitivity classifications, optimal CPU demands, optimal memory demands, and speedup information are derived from the Synergy-based profiling framework [14]. Since the original traces do not provide explicit deadline information, deadlines are synthesized following established methodologies [29,30]. Specifically, each job's deadline is randomly generated within the range of $1.2\times$ to $2.0\times$ its actual execution time, thereby emulating realistic timing constraints. Table 2 summarizes the source of each workload attribute.

To evaluate scheduling performance under different workload characteristics, we construct three workloads with different proportions of sensitive jobs for each trace and cluster configuration. For each trace, the three workloads contain 20%, 50%, and 80% sensitive jobs, respectively, with the remaining jobs classified as insensitive, resulting in 12 workloads across the two traces and cluster configurations. The

Table 2
Sources of workload attributes.

Attribute	Source
Arrival time	Generated (Poisson)
GPU demand	Trace
Runtime	Trace
Deadline	Generated
Sensitivity class	Profiling
Optimal CPU demand	Profiling
Optimal memory demand	Profiling
Speedup	Profiling

Table 3
Summary of workloads in our experiments.

Workload	Sensitive jobs/Insensitive jobs (%)	Trace	Cluster	# of jobs
H_MIX1	20/80	Helios	C67	10993
H_MIX2	50/50	Helios	C67	10993
H_MIX3	80/20	Helios	C67	10993
P_MIX1	20/80	Philly	C32	24983
P_MIX2	50/50	Philly	C32	24983
P_MIX3	80/20	Philly	C32	24983
H8_MIX1	20/80	Helios	C134	10993
H8_MIX2	50/50	Helios	C134	10993
H8_MIX3	80/20	Helios	C134	10993
P8_MIX1	20/80	Philly	C64	24983
P8_MIX2	50/50	Philly	C64	24983
P8_MIX3	80/20	Philly	C64	24983

workloads used for the 8-GPU-per-node configuration are independently constructed from the corresponding traces rather than directly reusing those constructed for the 16-GPU-per-node configuration. Since the 8-GPU-per-node configuration provides more auxiliary resources relative to GPUs, the resulting resource environment may reduce the auxiliary-resource sensitivity of some jobs during profiling, leading to differences in the final sensitivity classification between the two cluster configurations. The resulting workloads are summarized in Table 3.

Metrics. We adopt the following metrics to evaluate the effectiveness of the SensaScale scheduler. Let T_j denote JCT of job j , n represent the total number of jobs, and n_v denote the number of jobs that violate their deadlines.

SLO Violation Rate: The SLO violation rate quantifies the proportion of jobs that fail to satisfy the deadline constraint specified in the SLO. This metric directly reflects the scheduler's capability in deadline-aware scheduling and rational multi-resource allocation.

$$V_r = \frac{n_v}{n} \times 100\% \quad (4)$$

Job Completion Time: The JCT is defined as the average time interval from job arrival to job completion. This metric serves as a primary indicator of the scheduler's overall efficiency and performance optimization capability.

$$JCT_{avg} = \frac{\sum_{j \in Jobs} T_j}{n} \quad (5)$$

Average Queuing Time: The AQT is defined as the average waiting time between job submission and the commencement of execution. This metric effectively reflects the scheduler's resource allocation efficiency and overall system throughput.

Baselines. To comprehensively evaluate the performance advantages of SensaScale, we compare it against the following SOTA schedulers.

- Chronus [31]: A deadline-aware scheduling system for DLT jobs that integrates a lease-based training mechanism with a MILP solver to improve deadline satisfaction for SLO jobs.
- AlloX [32]: This scheduler formulates the scheduling problem as a bipartite graph matching task grounded in graph theory. Although it can theoretically minimize the total Job Completion Time (JCT_{total}), its design does not explicitly incorporate deadline constraints.

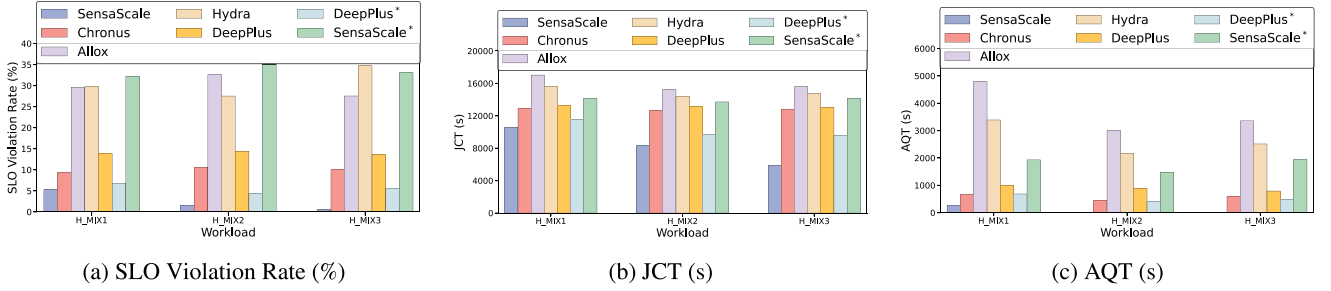


Fig. 6. Performance comparison of mixed workload scheduling based on Helios trace(C67).

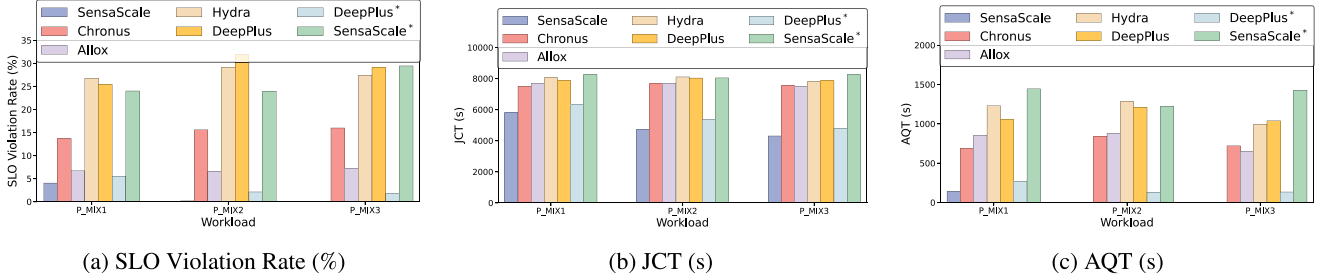


Fig. 7. Performance comparison of mixed workload scheduling based on Philly trace(C32).

- Hydra [33]: A deadline-aware and efficiency-oriented scheduler for DL jobs. It adopts a quantitative cost-comparison framework to satisfy job deadlines while optimizing overall system efficiency.
- DeepPlus [34]: A deadline-aware scheduler designed for deep learning training workloads. By incorporating mechanisms such as job classification and deadline-aware scheduling, DeepPlus effectively reduces JCT while ensuring that jobs meet their deadline requirements.
- DeepPlus*: An extended version of DeepPlus that integrates resource-sensitivity-aware allocation into the original deadline-aware scheduling framework. DeepPlus* classifies jobs based on their auxiliary-resource sensitivity and applies different allocation strategies accordingly. It directly provisions optimal CPU and memory demands for insensitive jobs, while sensitive jobs are allocated with their optimal resource configurations whenever possible and fall back to GPU-proportional allocation otherwise. This extension enables DeepPlus* to consider auxiliary-resource characteristics during scheduling while preserving the original deadline-aware resource adjustment mechanism.
- SensaScale*: SensaScale* is a simplified version of SensaScale used as a baseline for comparison. It removes the resource-sensitivity-aware scheduling mechanism while retaining the basic scheduling framework. In this version, jobs are scheduled in ascending order of their LST, and CPU and memory resources are allocated strictly in proportion to the number of GPUs requested by each job. This baseline is used to evaluate the contribution of the resource-sensitivity-based scheduling mechanism in SensaScale.

4.2. Scheduler effectiveness

This section systematically evaluates the scheduling effectiveness of SensaScale and provides a comparative analysis with existing schedulers. Three key metrics are adopted for quantitative evaluation: SLO violation rate, JCT, and AQT. To comprehensively validate the generality and robustness of SensaScale, we conduct experiments on six

datasets derived from two real-world production cluster traces, Helios and Philly. Specifically, experiments are conducted on H_MIX1, H_MIX2, and H_MIX3 workloads from the Helios trace deployed on the 67-node cluster (C67), as well as P_MIX1, P_MIX2, and P_MIX3 workloads from the Philly trace deployed on the 32-node cluster (C32). The experimental results across all datasets consistently demonstrate that SensaScale achieves superior scheduling performance under all evaluated workload scenarios.

Figs. 6(a) and 7(a) illustrate that, across both Helios- and Philly-based workload traces, SensaScale consistently achieves the lowest SLO violation rate under varying proportions of sensitive jobs. In terms of SLO violation rate, SensaScale achieves 5.25%, 1.57%, and 0.48% on H_MIX1, H_MIX2, and H_MIX3, respectively, substantially outperforming Chronus, which reports 9.30%, 10.56%, and 10.02%. The corresponding relative reductions are 43.5%, 85.1%, and 95.2%. Notably, under H_MIX3 (80% sensitive jobs), SensaScale reduces the violation rate by 95.2%. For the Philly-based workloads, SensaScale achieves violation rates of 4.05%, 0.18%, and 0.01% on P_MIX1, P_MIX2, and P_MIX3, respectively, whereas Chronus reports 13.69%, 15.59%, and 15.99%. In the P_MIX3 scenario, SensaScale attains a 99.9% reduction in violation rate compared with Chronus, nearly eliminating deadline violations altogether.

Figs. 6(b), (c) and 7(b), (c) further present the JCT and AQT achieved by each scheduler under different workload compositions. Across all evaluated scenarios, SensaScale consistently outperforms competing schedulers. Moreover, across all six datasets and the three primary metrics (SLO violation rate, JCT, and AQT), the performance advantage of SensaScale becomes increasingly significant as the proportion of sensitive jobs rises. Correspondingly, the performance gap between SensaScale and the other schedulers further widens with higher sensitivity ratios.

In contrast, Allox, Hydra, and DeepPlus exhibit relatively inferior performance. Allox ignores deadline constraints, which leads to a high SLO violation rate. Hydra is primarily designed for heterogeneous environments, and its scheduling mechanism becomes redundant and less efficient in homogeneous clusters. DeepPlus prioritizes training jobs that are close to completion, which may cause other training jobs to miss their deadlines due to limited resource availability. Although

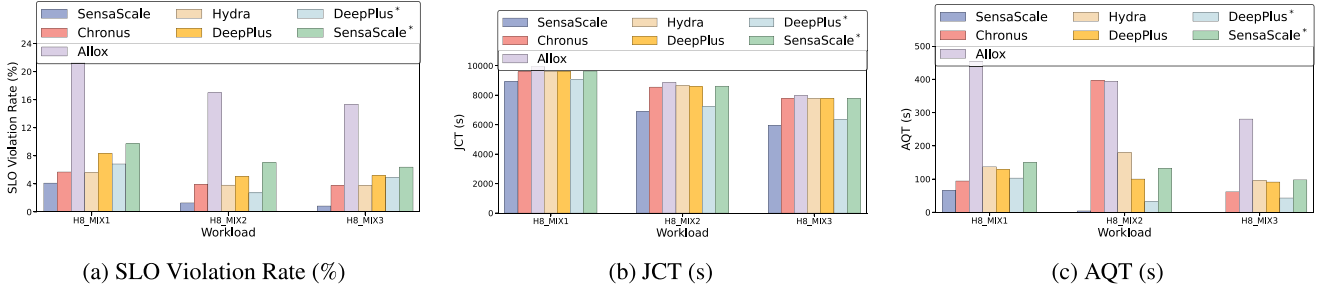


Fig. 8. Performance comparison of mixed workload scheduling based on Helios trace(C134).

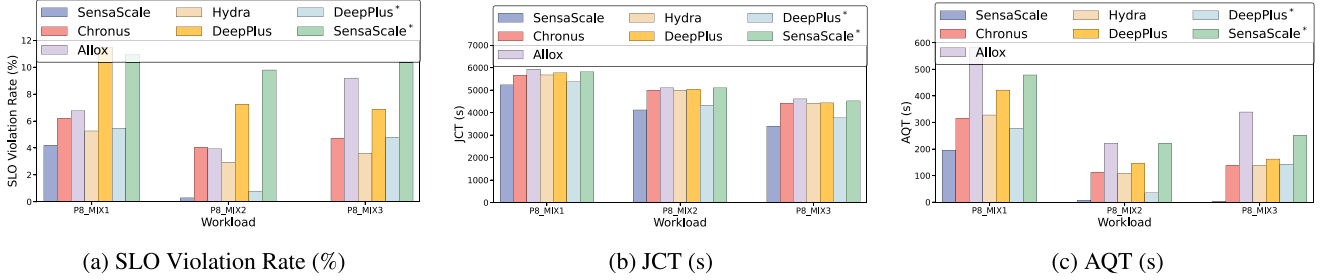


Fig. 9. Performance comparison of mixed workload scheduling based on Philly trace(C64).

Chronus demonstrates relatively stable and strong performance, its SLO violation rate and JCT remain comparatively high. All four schedulers adopt a GPU-proportional allocation strategy to assign CPU and memory resources, failing to fully exploit the heterogeneity in job sensitivity to auxiliary resources. As a result, they are unable to accelerate job execution through resource optimization, which constitutes the primary reason why their scheduling performance is significantly lower than that of SensaScale.

To further evaluate the effectiveness of incorporating auxiliary-resource sensitivity into deadline-aware scheduling, we compare DeepPlus with its enhanced variant DeepPlus*. DeepPlus* extends the original DeepPlus framework by introducing sensitivity-aware CPU and memory allocation. Experimental results show that DeepPlus* consistently improves upon DeepPlus across different workload compositions, achieving lower SLO violation rates and better execution performance. These improvements demonstrate that considering auxiliary-resource characteristics can effectively enhance deadline-aware scheduling performance.

However, DeepPlus* still underperforms SensaScale, indicating that sensitivity-aware resource provisioning alone is insufficient to achieve optimal scheduling performance. DeepPlus* mainly adjusts resource allocation according to individual job sensitivity, while SensaScale further exploits cross-job resource complementarity through deadline-constrained grouping, complementary pairing, and dynamic resource adjustment. By jointly considering job sensitivity, resource availability, and deadline urgency, SensaScale can better utilize heterogeneous auxiliary-resource demands and achieve superior scheduling performance, particularly under workloads with a higher proportion of sensitive jobs.

Compared with SensaScale*, SensaScale introduces a rational and dynamic multi-resource allocation mechanism that explicitly captures and exploits job sensitivity to CPU and memory resources. This mechanism effectively accelerates job execution while maintaining strict deadline compliance. Notably, as the proportion of sensitive jobs increases, the performance advantage of SensaScale grows substantially, demonstrating that its scheduling design is particularly well suited

for high-sensitivity workload scenarios. In environments dominated by sensitive jobs, SensaScale nearly eliminates SLO violations and significantly reduces both JCT and AQT. Such improvements carry important practical implications for managing complex mixed workloads in modern data centers.

Overall, these results demonstrate that both auxiliary-resource sensitivity awareness and cross-job resource complementarity are critical for improving deadline-constrained DNN training scheduling. SensaScale effectively integrates these mechanisms and achieves robust performance improvements across diverse workload compositions.

4.3. Validation on modern GPU cluster configurations

To evaluate the effectiveness of SensaScale under modern GPU cluster configurations, we further evaluate SensaScale using an 8-GPU-per-node cluster setting. Six workloads derived from two real-world traces are used to evaluate all schedulers. Specifically, experiments are conducted on HB_MIX1, HB_MIX2, and HB_MIX3 workloads from the Helios trace deployed on the 134-node cluster (C134), and P8_MIX1, P8_MIX2, and P8_MIX3 workloads from the Philly trace deployed on the 64-node cluster (C64).

The results are shown in Figs. 8 and 9. Under the 8-GPU-per-node cluster configuration, SensaScale achieves the lowest SLO violation rate among all evaluated schedulers across both Helios- and Philly-based workloads. For the Helios-based workloads, SensaScale achieves SLO violation rates of 4.11%, 1.28%, and 0.84% for the workloads with 20%, 50%, and 80% sensitive jobs, respectively. Compared with Chronus, SensaScale reduces the SLO violation rate by 27.77%, 67.35%, and 77.72%, respectively. For the Philly-based workloads, SensaScale achieves SLO violation rates of 4.18%, 0.29%, and 0.02%, respectively, corresponding to reductions of 32.36%, 92.79%, and 99.57% compared with Chronus. SensaScale also consistently achieves the lowest JCT and AQT across all workload compositions for both traces. As the proportion of sensitive jobs increases, the advantage of SensaScale in deadline satisfaction becomes more pronounced, particularly for the Philly-based workloads.

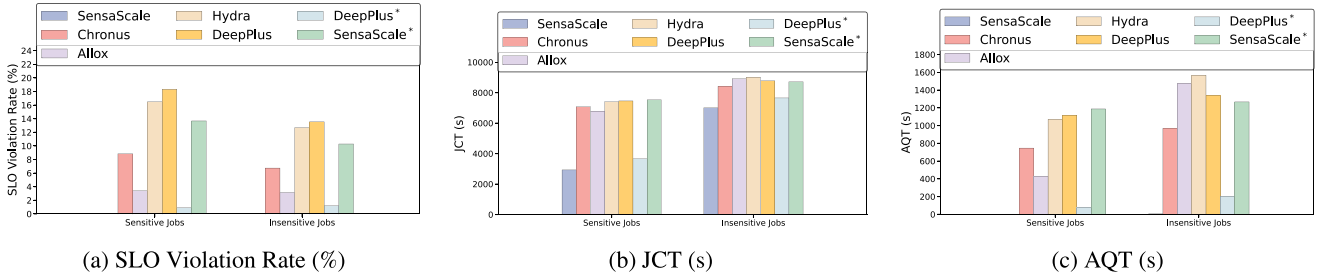


Fig. 10. Performance of SensaScale on sensitive and insensitive jobs.

Overall, these results demonstrate that SensaScale remains effective under GPU cluster configurations with more abundant auxiliary resources relative to GPUs. Although the increased auxiliary-resource-to-GPU ratio can reduce the sensitivity of some jobs to auxiliary resources, SensaScale continues to effectively exploit the remaining resource-sensitivity differences while incorporating deadline requirements into scheduling decisions. These results demonstrate the adaptability of SensaScale to modern GPU cluster configurations with finer GPU granularity and a higher auxiliary-resource-to-GPU ratio.

4.4. Performance across different job categories

To further analyze the effectiveness of resource-sensitivity-aware scheduling, we separately evaluate the performance of SensaScale on sensitive and insensitive jobs. Fig. 10 presents the SLO violation rate, JCT, and AQT of different schedulers under the P_MIX2 workload (deployed on C32).

For sensitive jobs, SensaScale achieves the lowest SLO violation rate among all evaluated schedulers, reducing it to only 0.08%, which is significantly lower than Chronus (8.85%), Allox (3.40%), Hydra (16.49%), DeepPlus (18.34%), DeepPlus* (0.92%), and SensaScale* (13.67%). This substantial improvement demonstrates that SensaScale effectively exploits auxiliary-resource sensitivity by allocating CPU and memory resources according to the optimal resource demands of each job. As a result, the execution time and queuing delay of sensitive jobs are reduced, thereby improving deadline satisfaction.

For insensitive jobs, SensaScale also maintains strong deadline performance, achieving an SLO violation rate of only 0.10%. Since insensitive jobs have limited dependency on auxiliary resources, they do not directly benefit from sensitivity-aware resource allocation. However, the improved scheduling efficiency of sensitive jobs reduces unnecessary resource contention and accelerates the overall workload progression, thereby indirectly benefiting insensitive jobs. Compared with other schedulers, SensaScale achieves lower JCT and AQT for insensitive jobs while maintaining stable deadline satisfaction, demonstrating that optimizing sensitive jobs does not compromise the performance of insensitive jobs.

Overall, these results confirm that SensaScale can effectively exploit heterogeneous auxiliary-resource requirements across different job categories. More importantly, the results validate that SensaScale's core design provides significant benefits for sensitive jobs while avoiding performance degradation for insensitive jobs. By improving the execution efficiency of resource-sensitive jobs and reducing resource contention within the cluster, SensaScale achieves better overall scheduling performance under mixed workloads.

4.5. Impact of deadline-constrained group size

To examine the impact of deadline-constrained group size on scheduling performance, we conduct experiments with fourteen different group sizes ranging from 4 to 30, increasing in increments of 2. The SLO violation rate is adopted as the primary evaluation metric.

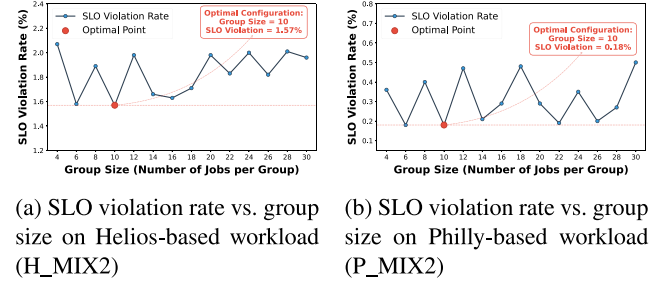


Fig. 11. Impact of deadline-constrained group size on SLO violation rate across different workloads.

Experiments are carried out on two representative workloads: H_MIX2 deployed on C67 and P_MIX2 deployed on C32.

As shown in Fig. 11(a) and (b), under the H_MIX2 and P_MIX2 workloads, both curves exhibit highly consistent trends. Specifically, within the range of group size from 6 to 16, the SLO violation rate remains relatively stable, with only minor performance fluctuations and no noticeable degradation. This indicates that SensaScale is not highly sensitive to moderate variations in group size, demonstrating strong robustness under heterogeneous workloads. Within this stable region, a group size of 10 achieves the best overall performance. Therefore, we adopt a group size of 10 as the default configuration in subsequent experiments to strike a favorable balance between stability and performance. Beyond this range, performance begins to degrade, suggesting that overly small or overly large group sizes may disrupt the balance between pairing diversity and scheduling accuracy. However, the existence of a broad stable region further indicates that SensaScale does not require fine-grained tuning of the group size parameter in practical deployments.

Overall, these results demonstrate that SensaScale maintains stable performance under moderate variations in group size, highlighting its robustness across different cluster traces and workload distributions.

4.6. Impact of the sensitivity coefficient a in the deadline-aware resource adjustment mechanism

To evaluate the impact of the sensitivity coefficient a in the proposed deadline-aware resource adjustment mechanism on scheduling performance, we conduct 28 experiments with a varying from 0.7 to 3.4 in increments of 0.1, thereby covering the entire spectrum from lenient to strict decision thresholds. The SLO violation rate is adopted as the primary evaluation metric, and violations are reported separately for all jobs, sensitive jobs, and insensitive jobs. The evaluation is performed on two representative workloads: H_MIX2 (deployed on C67) and P_MIX2 (deployed on C32).

Fig. 12(a) illustrates the variation of the SLO violation rate with respect to a on the H_MIX2 dataset. The results exhibit a clear “V-shaped” trend in the overall SLO violation rate. As a increases from 0.7 to 1.5, the violation rate decreases from 1.82% to a minimum of 1.57%,

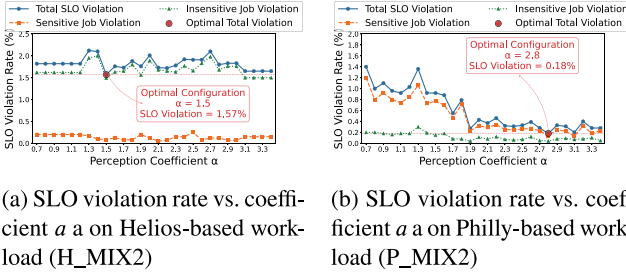


Fig. 12. Impact of sensitivity coefficient a in the deadline-aware resource adjustment mechanism on SLO violation rate.

corresponding to a relative improvement of 13.7%. Beyond this turning point, the violation rate gradually increases as a continues to grow. On the P_MIX2 dataset (Fig. 12(b)), a pronounced downward trend is observed. When a increases from 0.7 to 2.8, the overall violation rate drops substantially from 1.40% to 0.18%, representing an 87.1% improvement. The best performance is achieved at $a = 2.8$, where the overall SLO violation rate reaches its minimum value of 0.18%.

Based on the results from the two datasets, we draw the following conclusions. The sensitivity coefficient a directly regulates the urgency evaluation of sensitive jobs during scheduling, thereby influencing their resource allocation decisions and indirectly affecting the resource availability and JCT of insensitive jobs. In the H_MIX2 dataset, sensitive jobs account for a relatively small proportion of the total violating jobs. Consequently, although tuning a improves the scheduling performance of sensitive jobs, its impact on the overall SLO violation rate remains limited. In contrast, in the P_MIX2 dataset, violations of sensitive jobs dominate the total violation count. Therefore, adjusting a directly and substantially reduces the overall violation rate, resulting in a pronounced “V-shaped” trend. Although the optimal value of a differs across the two datasets, the proposed deadline-aware resource adjustment mechanism consistently reduces the overall SLO violation rate under both workloads, demonstrating strong adaptability across heterogeneous job compositions. The experimental results indicate that a moderate value of a effectively balances resource-allocation efficiency and deadline satisfaction, thereby achieving near-optimal scheduling performance across workloads with varying job characteristics.

4.7. Ablation study

To obtain deeper insights into the performance contribution of each key component in the SensaScale scheduler, this section presents a systematic ablation study. We individually disable three core mechanisms of SensaScale: (1) the dynamic regrouping strategy, (2) the deadline-aware resource adjustment mechanism, and (3) the backfilling strategy. The impact of each component is evaluated on both the H_MIX2 (deployed on C67) and P_MIX2 (deployed on C32) datasets. Three primary metrics are employed for quantitative analysis: the SLO violation rate, JCT, and AQT.

Impact of the Dynamic Regrouping Strategy on Scheduling Performance: Figs. 13 and 14 illustrate the impact of the dynamic regrouping strategy on the scheduling efficiency of SensaScale. The experimental results demonstrate that removing this strategy results in systematic degradation across all three performance metrics on both datasets. On the H_MIX2 dataset, the SLO violation rate increases from 1.57% to 2.00% (a relative increase of 27.4%), the JCT rises from 8293.15, s to 8294.27, s, and the AQT deteriorates from 16.03, s to 20.22, s (an increase of 26.1%). A similar trend is observed on the P_MIX2 dataset, although with smaller performance gaps. These findings indicate that the dynamic regrouping strategy, by co-scheduling sensitive and insensitive jobs on the same server according to resource-complementarity principles, effectively exploits the heterogeneity of

auxiliary-resource demands among different jobs. By enabling more appropriate resource assignments across complementary job pairs, this mechanism effectively reduces the SLO violation rate, JCT, and AQT.

Impact of the Deadline-Aware Resource Adjustment Mechanism on Scheduling Performance: Figs. 13 and 14 further demonstrate the critical role of the deadline-aware resource adjustment mechanism in ensuring SLO compliance. As shown in Fig. 14, removing this mechanism on the P_MIX2 dataset causes the SLO violation rate to surge from 0.18% to 1.12% (a relative increase of 522.2%), while the JCT and AQT increase by 6.4% and 1131.3%, respectively. Similarly, as illustrated in Fig. 13, all three metrics deteriorate markedly on the H_MIX2 dataset, further confirming the overall performance-enhancement effect of this mechanism. The core function of the deadline-aware mechanism is to continuously monitor sensitive jobs executing under GPU-proportional auxiliary-resource allocation, identify those entering deadline-critical states, and dynamically increase their CPU and memory allocations to accelerate execution. Through timely resource adjustments, this mechanism effectively shortens JCT, substantially reduces the SLO violation rate, and significantly improves the overall performance and deadline satisfaction.

Impact of the Backfilling Strategy on Scheduling Performance: The ablation study of the backfilling strategy underscores its critical role in mitigating resource fragmentation. Figs. 13 and 14 illustrate the impact of this strategy on the scheduling efficiency of SensaScale. On the H_MIX2 dataset, removing the backfilling strategy causes the AQT to surge from 16.03, s to 121.14, s (a relative increase of 655.7%), while the SLO violation rate rises from 1.57% to 5.16% (an increase of 228.7%). By comprehensively evaluating the LST of jobs, the backfilling strategy inserts appropriate pending jobs into idle resource windows on servers without interfering with the execution of urgent jobs. This approach significantly improves resource utilization and enhances scheduling rationality. Consequently, the mechanism effectively reduces the SLO violation rate and shortens JCT.

The ablation results collectively demonstrate that all three components are essential to the effectiveness of SensaScale. The dynamic regrouping strategy primarily improves resource utilization by exploiting resource complementarity among jobs, thereby reducing both JCT and AQT. The deadline-aware resource adjustment mechanism plays a critical role in maintaining SLO compliance by dynamically adjusting auxiliary resources for deadline-critical jobs. Meanwhile, the backfilling strategy effectively alleviates resource fragmentation and significantly shortens queueing delays. Together, these mechanisms form a complementary design that balances application performance and deadline satisfaction, enabling SensaScale to achieve robust scheduling performance across different workloads.

4.8. Scheduling overhead analysis

Beyond scheduling effectiveness, we further evaluate the runtime overhead introduced by SensaScale using the H_MIX2 and P_MIX2 workloads. Specifically, we record the number of jobs triggered by backfilling and rollback during scheduling and measure the average overhead of major scheduling components, including job grouping, complementary pairing, resource rollback, and deadline-aware resource adjustment mechanisms.

Scheduling Behavior. Table 4 summarizes the number of backfilling and rollback jobs under different workload traces. The backfilling strategy is activated when sufficient temporary resource gaps exist. Meanwhile, rollback operations are selectively triggered when excessive auxiliary resources allocated to running jobs prevent the admission of pending jobs. Compared with the total number of scheduled jobs, the number of rollback jobs is limited, indicating that the rollback mechanism is not frequently invoked during normal scheduling and only serves as a safeguard under resource contention scenarios.

Runtime Overhead. Table 5 presents the average execution latency of each scheduling component. Among all components, the grouping

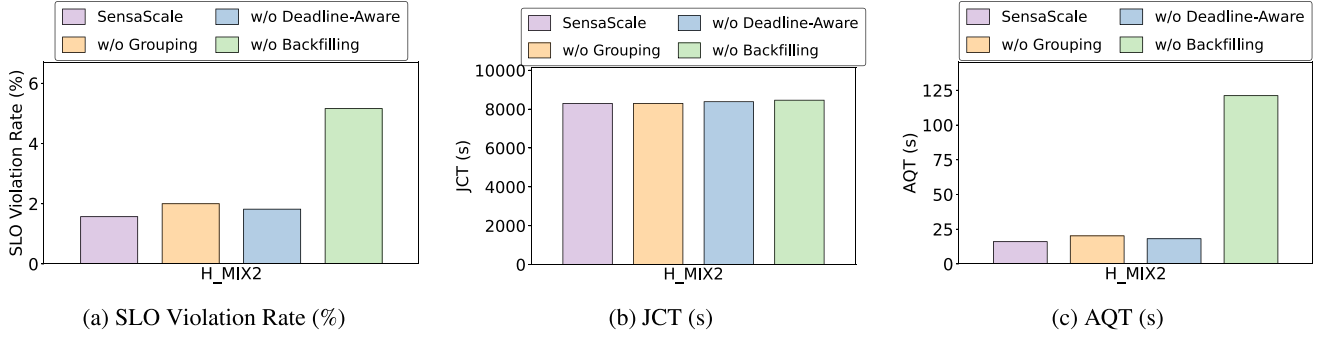


Fig. 13. Performance degradation after removing individual components on Helios-based workload (H_MIX2).

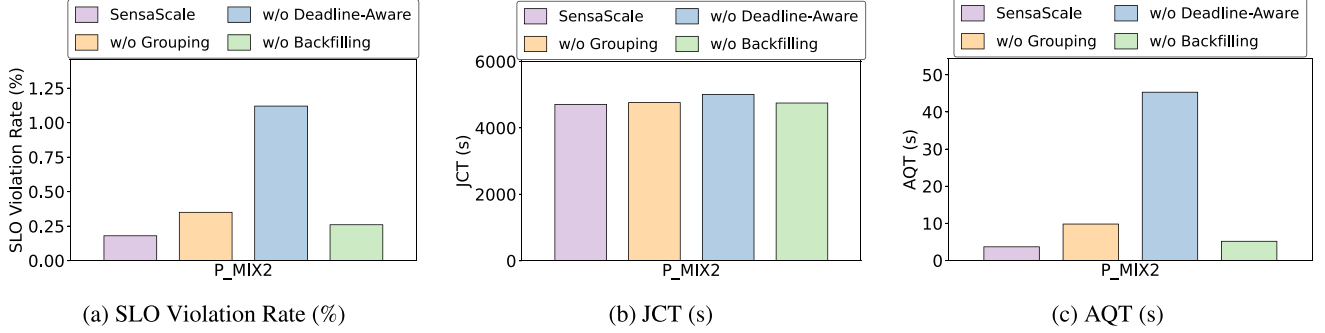


Fig. 14. Performance degradation after removing individual components on Philly-based workload (P_MIX2).

Table 4
Scheduling behavior statistics of SensaScale.

Workload	Total jobs	Backfilled jobs	Rollback jobs
H_MIX2	10993	1484	33
P_MIX2	24983	203	225

Table 5
Average runtime overhead of Sensascale components (ms).

Component	H_MIX2	P_MIX2
Grouping	0.837	1.701
Pairing	0.005	0.006
Rollback	0.179	0.183
Deadline-aware resource adjustment mechanisms	0.005	0.005

procedure introduces the largest overhead, with an average latency of 1.70 ms, because it periodically examines pending jobs and constructs deadline-constrained groups according to their latest start times. Nevertheless, this overhead remains negligible compared with the execution time of DNN training jobs. The complementary pairing, rollback, and deadline-aware resource adjustment mechanisms introduce only sub-millisecond latency, demonstrating that these operations can be efficiently executed during online scheduling.

4.9. Robustness analysis under stochastic workload generation

Since job arrivals and deadlines are generated stochastically during workload construction, different random seeds may lead to variations in workload execution patterns. To evaluate the robustness of SensaScale under stochastic workload conditions, we conduct repeated simulation runs with different random seeds. Specifically, we select the P_MIX2 workload derived from the Philly trace under the C32 cluster configuration and perform 10 independent simulation runs. In each run, job arrival times are generated according to the Poisson process, and deadline assignments are regenerated following the workload construction methodology described in Section 4.1. Other experimental

Table 6
Robustness analysis under stochastic workload generation.

Metric	Mean $\pm$ Std.
SLO violation rate (%)	0.189 $\pm$ 0.132
Average JCT (s)	4612.66 $\pm$ 93.91
Average AQT (s)	5.69 $\pm$ 3.35

settings, including workload characteristics, cluster configuration, and scheduling policies, remain unchanged.

For each run, we record the SLO violation rate, JCT, and AQT. Table 6 reports the mean values and standard deviations across the 10 independent runs. The results demonstrate that SensaScale maintains consistent performance across different workload realizations. The observed variations are introduced by the inherent randomness of workload generation, while the overall performance trend remains stable across repeated runs. These results indicate that SensaScale is robust to stochastic workload conditions.

5. Related work

Auxiliary-Resource Sensitivity of DNN Jobs. Recent studies show that DNN training jobs exhibit significant sensitivity to auxiliary resources such as CPU and memory. Jeon et al. [18] analyze production GPU cluster traces and observe heterogeneous host resource usage across jobs, where CPU is often underutilized while memory is highly contended. Based on this insight, they advocate memory-aware scheduling with isolation to mitigate interference among co-located jobs. Building on this, Mohan et al. [14] propose Synergy, which departs from conventional GPU-proportional resource allocation by modeling the impact of auxiliary resources on job performance and dynamically adjusting CPU and memory allocations according to job sensitivity. By exploiting heterogeneous resource sensitivity across jobs, Synergy improves cluster throughput and reduces JCT. Subsequent

work [35] further refines resource-aware scheduling and isolation to enhance training efficiency.

Although these studies demonstrate the benefits of resource-sensitive scheduling, their primary objective is performance optimization, such as improving cluster throughput and reducing JCT. They do not explicitly incorporate deadline constraints, job urgency, or SLO satisfaction into scheduling decisions. Consequently, it remains unclear how resource sensitivity should be exploited in deadline-constrained DNN training environments. In contrast, this work investigates how auxiliary-resource sensitivity can be integrated into deadline-aware scheduling to improve deadline satisfaction while maintaining efficient resource utilization.

Schedulers for Multi-Dimensional Resource Allocation. Multi-resource scheduling has been extensively studied in big-data systems. For instance, systems such as Tetris [36] and DRF [37] propose various scheduling strategies for multi-dimensional resource allocation and optimize cluster-level objectives under the assumption that job resource demands are known in advance. Schedulers such as Tetris and DRF were originally designed for big-data workloads (e.g., MapReduce), whose characteristics differ significantly from those of DNN training tasks. In DNN training workloads, GPUs serve as the primary computational resource and are typically explicitly specified by users when submitting jobs, while auxiliary resources such as CPU and memory provide greater flexibility in allocation. Furthermore, different types of DNN jobs exhibit varying degrees of performance sensitivity to these auxiliary resources, implying that identical CPU or memory allocations may produce substantially different performance outcomes across workloads. Motivated by this observation, SensaScale dynamically adjusts auxiliary-resource allocations according to job-specific sensitivity characteristics rather than relying on fixed GPU-proportional allocation policies.

Deadline-Aware Scheduling for Deep Learning Jobs. In recent years, several studies have proposed deadline-aware scheduling mechanisms for machine learning workloads. HyperSched [38] primarily focuses on optimizing hyperparameter tuning workloads to satisfy deadline constraints, and its design is not directly tailored for general deep learning training jobs. TetrisSched [39] adopts a Mixed Integer Linear Programming (MILP) approach to optimize scheduling decisions and has demonstrated strong performance in big-data job scheduling; however, adapting such methods to the dynamic characteristics of deep learning workloads remains challenging. Chronus [31] predicts the execution time of deep learning jobs and leverages an MILP solver to minimize deadline violation rates. Hydra [33] employs a stochastic sampling approach to estimate job execution time and improves deadline satisfaction through quantified cost comparisons, particularly for deep learning workloads exhibiting significant execution-time variation across different GPUs. DeepPlus [34] proposes a deadline- and fairness-aware scheduler, which improves cluster efficiency through mechanisms such as job classification and deadline-aware resource backfilling to improve deadline satisfaction. Despite their effectiveness, most existing deadline-aware schedulers typically treat GPUs as the dominant resource in scheduling decisions and allocate auxiliary resources such as CPU and memory proportionally to GPU demands. As a result, they overlook the heterogeneous auxiliary-resource sensitivity exhibited by DNN training jobs and may fail to fully utilize available auxiliary resources. Consequently, opportunities to further reduce deadline violations through resource-sensitive resource allocation remain largely unexplored.

Overall, existing work follows two largely independent directions. Deadline-aware schedulers focus on improving deadline satisfaction but generally rely on GPU-proportional auxiliary-resource allocation. In contrast, resource-sensitive schedulers exploit heterogeneous CPU and memory sensitivity to improve throughput and JCT but do not explicitly consider deadline constraints. Although both directions have demonstrated substantial benefits, the integration of auxiliary-resource

sensitivity and deadline-aware scheduling has received limited attention. SensaScale combines these complementary perspectives to improve deadline satisfaction and resource utilization for DNN training workloads.

6. Conclusion

This paper presents a novel deadline-aware scheduler, SensaScale, which exploits the sensitivity of DNN jobs to auxiliary resources to reduce SLO violation rates. The core innovation lies in categorizing jobs into sensitive and insensitive classes and designing a set of cooperative scheduling mechanisms accordingly. First, we propose a dynamic regrouping strategy that pairs sensitive and insensitive jobs based on resource complementarity and performs scheduling decisions by jointly considering job sensitivity and node-level resource availability. Second, we introduce a deadline-safe backfilling strategy that enables efficient resource reuse while minimizing the impact on the execution performance of urgent jobs. In addition, a deadline-aware resource adjustment mechanism continuously monitors job execution status and dynamically adjusts resource allocation when deadlines become tight, thereby accelerating the completion of time-critical jobs. Experimental results demonstrate that, compared with the Chronus baseline, SensaScale significantly reduces the SLO violation rate (by an average of 74.22%), effectively shortens the JCT (by an average of 25.86%), and substantially compresses the AQT (by an average of 82.75%) across six workload configurations from the Helios and Philly traces and both 16-GPU and 8-GPU cluster settings.

CRedit authorship contribution statement

Kaimin Wu: Writing – original draft, Software, Methodology. **Xinhua Wang:** Writing – review & editing, Methodology. **Zhiping Peng:** Writing – review & editing, Supervision. **Rong Xie:** Writing – review & editing, Supervision. **Weiwei Lin:** Writing – review & editing, Supervision, Funding acquisition. **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 (PCL2025A08 and PCL2025A11).

Data availability

Data will be made available on request.

References

- [1] Guoping Xu, et al., Development of residual learning in deep neural networks for computer vision: A survey, *Eng. Appl. Artif. Intell.* 142 (2025) 109890.
- [2] Ashish Vaswani, et al., Attention is all you need, *Adv. Neural Inf. Process. Syst.* 30 (2017).
- [3] Gengnan Fu, Qun Xie, Research on the construction of acoustic model for foreign language speech recognition based on deep neural network, in: *Proceedings of the IEEE International Conference on Digital Processing and Communication Applications*, 2025, pp. 171–174.
- [4] Wencong Xiao, et al., AntMan: Dynamic scaling on GPU clusters for deep learning, in: *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2020, pp. 533–548.

- [5] Yuchen Zhang, et al., Deadline-aware online job scheduling for distributed training in heterogeneous clusters, *IEEE Trans. Cloud Comput.* (2025).
- [6] Brendan Burns, et al., Borg, omega, and kubernetes, *Commun. ACM* 59 (5) (2016) 50–57.
- [7] Vinod Kumar Vavilapalli, et al., Apache hadoop yarn: Yet another resource negotiator, in: *Proceedings of the ACM Symposium on Cloud Computing*, 2013, pp. 1–16.
- [8] Arezoo Jahani, Seyed Mohammad Hasani, Performance prediction in training of deep learning applications on GPU-based cloud services, *Computing* 108 (3) (2026) 36.
- [9] Tianhao Fu, et al., A survey on the scheduling of DL and LLM training jobs in GPU clusters, *Chin. J. Electron.* 34 (3) (2025) 881–905.
- [10] Jayashree Mohan, et al., Analyzing and mitigating data stalls in DNN training, 2020, *arXiv preprint arXiv:2007.06775*.
- [11] Gyewon Lee, et al., Refurbish your training data: Reusing partially augmented samples for faster deep neural network training, in: *Proceedings of the USENIX Annual Technical Conference*, 2021, pp. 537–550.
- [12] Derek G. Murray, et al., Tf.data: A machine learning data processing framework, 2021, *arXiv preprint arXiv:2101.12127*.
- [13] Mark Zhao, et al., Understanding and co-designing the data ingestion pipeline for industry-scale recsys training, *Comput. Res. Repos.* (2021).
- [14] Jayashree Mohan, et al., Looking beyond GPUs for DNN scheduling on multi-tenant clusters, in: *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2022, pp. 579–596.
- [15] Yonghui Wu, et al., Google's neural machine translation system: Bridging the gap between human and machine translation, 2016, *arXiv preprint arXiv:1609.08144*.
- [16] Kshiteej Mahajan, et al., Themis: Fair and efficient GPU cluster scheduling, in: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2020, pp. 289–304.
- [17] Deepak Narayanan, et al., Heterogeneity-aware cluster scheduling policies for deep learning workloads, in: *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2020, pp. 481–498.
- [18] Myeongjae Jeon, et al., Analysis of large-scale multi-tenant GPU clusters for DNN training workloads, in: *Proceedings of the USENIX Annual Technical Conference*, 2019, pp. 947–960.
- [19] Qinghao Hu, et al., 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, pp. 1–15.
- [20] Alex Krizhevsky, et al., Imagenet classification with deep convolutional neural networks, *Adv. Neural Inf. Process. Syst.* 25 (2012).
- [21] Wei Dai, et al., Very deep convolutional neural networks for raw waveforms, in: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017, pp. 421–425.
- [22] Kaiming He, et al., Deep residual learning for image recognition, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
- [23] Xiangyu Zhang, et al., Shufflenet: An extremely efficient convolutional neural network for mobile devices, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.
- [24] Mark Sandler, et al., Mobilenetv2: Inverted residuals and linear bottlenecks, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.
- [25] Jacob Devlin, et al., BERT: Pre-training of deep bidirectional transformers for language understanding, in: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.
- [26] Awni Hannun, et al., Deep speech: Scaling up end-to-end speech recognition, 2014, *arXiv preprint arXiv:1412.5567*.
- [27] Sepp Hochreiter, Jürgen Schmidhuber, Long short-term memory, *Neural Comput.* 9 (8) (1997) 1735–1780.
- [28] Diederik P. Kingma, Max Welling, Auto-encoding variational bayes, 2013, *arXiv preprint arXiv:1312.6114*.
- [29] Sangeetha Abdu Jyothi, et al., Morpheus: Towards automated SLOs for enterprise clusters, in: *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2016, pp. 117–134.
- [30] Jun Woo Park, et al., 3Sigma: Distribution-based cluster scheduling for runtime uncertainty, in: *Proceedings of the European Conference on Computer Systems*, 2018, pp. 1–17.
- [31] Wei Gao, et al., Chronus: A novel deadline-aware scheduler for deep learning training jobs, in: *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 609–623.
- [32] Tan N. Le, et al., AlloX: Compute allocation in hybrid clusters, in: *Proceedings of the European Conference on Computer Systems*, 2020, pp. 1–16.
- [33] Zichao Yang, et al., Hydra: Deadline-aware and efficiency-oriented scheduling for deep learning jobs on heterogeneous GPUs, *IEEE Trans. Comput.* 72 (8) (2023) 2224–2236.
- [34] Yupeng Xi, Chao Jing, DeepPlus: An adaptive deadline and fairness aware scheduler for accelerating mixed deep learning training and inference workloads in GPU cluster, *Comput. Netw.* (2025).
- [35] Zhisheng Ye, et al., Deep learning workload scheduling in GPU datacenters: A survey, *ACM Comput. Surv.* 56 (6) (2024) 1–38.
- [36] Robert Grandl, et al., Multi-resource packing for cluster schedulers, *ACM SIGCOMM Comput. Commun. Rev.* 44 (4) (2014) 455–466.
- [37] Ali Ghodsi, et al., Dominant resource fairness: Fair allocation of multiple resource types, in: *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation*, 2011.
- [38] Richard Liaw, et al., Hypersched: Dynamic resource reallocation for model development on a deadline, in: *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 61–73.
- [39] Alexey Tumanov, et al., Tetrisched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters, in: *Proceedings of the European Conference on Computer Systems*, 2016, pp. 1–16.



Kaimin Wu is currently working toward the M.S. degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. Her research interests include cloud computing, GPU scheduling and DL workload scheduling.



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 DL workload scheduling.



Zhiping Peng received the B.S. degree from the China University of Petroleum, Huadong, in 1996, the M.S. degree from the Huazhong University of Science and Technology, in 2001, and the Ph.D. degree in computer applications from the South China University of Technology, in 2007. He is currently a Professor with the WuYi University. He has published more than 60 articles in refereed journals and conference proceedings. His research interests include cloud computing, machine learning, and multi-agent systems.



Rong Xie, Ph.D. is an Associate Professor and Master's Supervisor at the School of Big Data and Artificial Intelligence, Guangdong University of Finance and Economics. He received the Ph.D. degree from the School of Computer Science and Engineering, South China University of Technology. His research interests focus on deep learning, biocomputing, object detection, and defect detection.



Weiwei Lin (Senior Member, IEEE) received the B.S. and M.S. degrees from Nanchang University in 2001 and 2004, respectively, and the Ph.D. in Computer Application from South China University of Technology in 2007. He has been a visiting scholar at Clemson University from 2016 to 2017. Currently, he is a professor in the School of Computer Science and Engineering at South China University of Technology. His research interests include distributed systems, cloud computing, and AI application technologies. He has published more than 200 papers in refereed journals and conference proceedings, such as ICML, NeurIPS, ACM Computing Surveys, IEEE TPDS, TSC, TCC, TMC, and TC. He is currently served on the editorial boards of *Future Generation Computer Systems*, *TBench*, and *Medical Data Mining*. He is a distinguished member of CCF and a senior member of the IEEE.



Keqin Li received a B.S. degree in computer science from Tsinghua University in 1985 and a Ph.D. degree in computer science from the University of Houston in 1990. He is a SUNY Distinguished Professor with the State University of New York and a National Distinguished Professor with Hunan University (China). He has authored or co-authored more than 1080 journal articles, book chapters, and refereed conference papers. He holds over 75 patents announced or authorized by the Chinese National Intellectual Property Administration. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research

Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He was a recipient of the 2022–2023 International Science and Technology Cooperation Award and the 2023 Xiaoxiang Friendship Award of Hunan Province, China. He is a Member of the SUNY Distinguished Academy. He is an AAAS Fellow, an IEEE Fellow, an AAIA Fellow, an ACIS Fellow, and an AIIA Fellow. He is a Member of the European Academy of Sciences and Arts. He is a Member of Academia Europaea (Academician of the Academy of Europe).