

Article

Entropy-Guided Hierarchical Scheduling for Elastic Distributed Deep Learning

Teh-Jen Sun ¹  and Eui-Nam Huh ^{2,3,*} ¹ Department of Artificial Intelligence, Kyung Hee University, Yongin 17104, Republic of Korea; dlwmznz11@khu.ac.kr² Department of Computer Science and Engineering, Kyung Hee University, Yongin 17104, Republic of Korea³ Artificial Intelligence Research Center, Kyung Hee University, Yongin 17104, Republic of Korea

* Correspondence: johnhuh@khu.ac.kr

Abstract

Shared GPU clusters often execute multiple distributed training jobs concurrently under fluctuating contention. We reinterpret this setting as a two-scale control problem, where the micro scale captures intra-job learning dynamics and the macro scale captures inter-job resource arbitration. We propose an entropy-guided hierarchical framework that links these two scales through a unified uncertainty signal computed from training logits. Unlike existing uncertainty-aware methods that typically use uncertainty for only a single level of decision making, our approach uses the same entropy-based signal to jointly support both intra-job adaptation and inter-job scheduling within a hierarchical control loop. At the micro level, each worker estimates predictive uncertainty via normalized entropy and converts it into stable weights that drive epoch-level controls for uncertainty-aware data sharding, fixed-budget batch-size reallocation, and learning-rate modulation, while remaining compatible with standard synchronous data-parallel training. At the macro level, the same signal is aggregated into a job utility score that guides admission, ordering, and GPU quota assignment under contention. In large-scale workload-driven simulation, our method reduces average job completion time (JCT) by 23.7% and shortens cluster makespan by 15.7% relative to a strong learning-unaware baseline, demonstrating that uncertainty-aligned scheduling can improve cluster-level efficiency while preserving training correctness. We further validate scalability using a calibrated simulator up to 1024 nodes.

Keywords: distributed deep learning; multi-job scheduling; hierarchical scheduling; uncertainty-aware training; entropy; virtual data-parallel training; adaptive data sharding; batch-size adaptation; GPU cluster orchestration



Academic Editor: Javid Taheri

Received: 3 March 2026

Revised: 31 March 2026

Accepted: 8 April 2026

Published: 10 April 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Large-scale deep learning training is increasingly carried out in shared environments rather than in isolation [1–3]. In many laboratories and production settings, a single GPU cluster serves as a common infrastructure where multiple distributed jobs begin, pause, and overlap throughout the day [4,5]. In this setting, training behavior is shaped not only by the model and the dataset but also by when and where a job runs [6,7]. Network contention rises and falls, available resources shift, and the scheduler repeatedly changes the parallelism assigned to each job [8,9]. These dynamics introduce two intertwined decision scopes. One concerns within-job choices that maintain efficient learning, and the

other concerns cross-job decisions that arbitrate shared GPUs. We refer to these as micro and macro in a job-centric sense. The micro level captures decisions made within a single distributed training job, while the macro level captures decisions made across multiple concurrent jobs in the cluster.

In shared GPU clusters, learning dynamics within a job and resource allocation across jobs become coupled in practice. At the micro level, a distributed job learns through worker-local data streams, gradient synchronization, and hyperparameters tuned under an assumed effective batch and synchronization regime [10,11]. At the macro level, the cluster is shared through macro-level scheduling decisions such as ordering, admission, and the number of workers assigned to each job over time [12,13]. These two levels are coupled because macro-level allocation changes the effective training regime of a job, and the resulting learning behavior determines whether additional resources will actually shorten time to accuracy [14–16].

In multi-job execution, the central objective is often time to accuracy [17,18]. A job is considered complete not when it merely consumes its assigned epochs, but when it reaches a target validation metric under fluctuating contention [19]. Yet the information available to the scheduler is typically dominated by system measurements [20,21]. These measurements describe how busy devices are, but they do not describe how much learning progress is being produced at a given moment [22]. When data-parallel training continues with fixed sharding and static hyperparameters under changing contention, a sequence of effects follows [23]. Effective batch size and synchronization behavior drift from the assumptions used to set the learning rate. Workers contribute gradients of different informativeness as their local shards differ in difficulty, while the system continues to average these contributions uniformly [24]. Under these conditions, cluster-level decisions can reduce waiting time while still yielding slow progress in model quality [25]. This appears as a longer time to accuracy and heavier tail behavior in JCT distributions [26].

A central difficulty is that the scheduler and the training system typically lack visibility into worker-level learning utility [27]. The system cannot tell which workers are currently operating on ambiguous examples where additional computation is likely to produce larger updates to the decision boundary [28,29]. The system also cannot tell how the number of classes and dataset complexity affect the scale and persistence of uncertainty during training [30]. In multi-class workloads, uncertainty often remains higher for longer, and the gap between easy shards and hard shards becomes more pronounced. If hard shards receive insufficient exposure under fixed sharding, informative gradients arrive too infrequently and convergence slows [31,32]. If the system attempts to compensate only through aggressive updates under a changed effective batch, training can become unstable, and the intended time-to-accuracy reduction is not achieved. An online signal is needed that reflects learning difficulty under contention and is stable enough to guide both training-time control and cluster-level scheduling [33]. The specific research gap addressed in this work is that existing schedulers typically rely on system-level or single-level optimization signals, and therefore do not consistently connect intra-job learning dynamics with inter-job resource allocation. As a result, they may improve resource efficiency or waiting time, while still lacking a unified learning-aware mechanism for jointly coordinating adaptation within a job and scheduling across jobs under shared-cluster contention.

In this study, we address this problem by using predictive uncertainty computed from logits as a lightweight online signal that bridges micro-level adaptation and macro-level orchestration. Each worker summarizes uncertainty using normalized entropy so that the signal remains comparable across tasks with different numbers of classes, and the resulting worker-level signals are converted into stable weights through clamping and exponential smoothing. These weights drive three micro-level controls for the next epoch, including

uncertainty-proportional data sharding, per worker batch-size reallocation under a fixed budget, and job-level learning-rate modulation to stabilize updates as the effective regime shifts. In parallel, the same uncertainty signal is aggregated into a job score used by the cluster scheduler so that macro-level allocation decisions favor configurations expected to reduce time to accuracy rather than merely balance hardware utilization. In workload-driven multi-job simulation, our approach reduces average JCT by 23.7% and shortens cluster makespan by 15.7% relative to a representative baseline (Lucid). These gains are driven primarily by reduced queueing delay, which drops by 30.7% under contention. In large-scale simulation at 1024 nodes, our method further reduces average JCT by 25.2% and makespan by 18.1% relative to the same baseline.

The main contributions of this paper are as follows:

- A two-scale control formulation for shared-cluster training, together with a unified uncertainty signal that bridges micro-level adaptation and macro-level scheduling.
- Stable budget-aware control rules for uncertainty-aware data sharding, batch sizing, and learning-rate modulation driven by normalized entropy.
- Empirical evidence that learning-aware orchestration improves time to accuracy and training stability with negligible overhead in multi-job, multi-class workloads, along with scalability validation in simulation up to 1024 nodes.

2. Related Work

2.1. Cluster Scheduling for Distributed Deep Learning Jobs

Deep learning clusters have motivated schedulers that treat training jobs as long-running workloads with distinct progress characteristics [34]. Gandiva introduced job management primitives such as time slicing and migration to improve responsiveness for interactive and iterative training workflows [4], and it emphasized that early training signals can be useful for prioritization in practice. Tiresias explored scheduling policies for distributed training when JCTs are uncertain and proposed mechanisms that reduce average JCT without requiring complete prior knowledge [35]. These systems establish the foundations for multi-tenant GPU scheduling and motivate designs that react to changing cluster conditions while keeping training jobs practical to deploy [22,36].

2.2. Elastic and Co-Adaptive Scheduling Under Contention

Modern shared-cluster schedulers increasingly treat parallelism as a control variable, adjusting it during execution to improve cluster-level outcomes under contention [37]. Pollux introduced a co-adaptive approach that jointly considers cluster-level allocation and per job training efficiency through the notion of goodput, enabling dynamic resource reassignment based on observed training behavior [10]. Lyra further studied elastic scheduling in deep learning clusters and showed that adapting resource allocation across mixed workloads can improve overall efficiency and utilization [11]. These directions highlight that inter-job decisions can change the effective training regime, which creates an opportunity for job-aware control signals that remain lightweight and model agnostic [6,38].

2.3. Intra-Job Adaptation of Batch Size and Learning Rate

Intra-job adaptation is commonly used to stabilize optimization and improve statistical efficiency as training progresses [23]. Prior work has studied adaptive batch sizes as a mechanism for controlling gradient noise and for trading off computation and convergence [39,40], including criteria based on estimates of gradient variance and relationships between batch-size growth and learning-rate schedules [25]. In parallel, practical training systems often adjust learning rate based on training dynamics to maintain stable updates across phases of learning [41]. Intra-job adaptation techniques motivate using

lightweight training signals to coordinate batch-size and learning-rate changes under resource variability [26,42,43].

2.4. Predictive Uncertainty as a Learning Signal

Uncertainty measures have a long history as a proxy for informativeness in learning systems [27,44]. Uncertainty sampling in active learning selects examples where the model is least confident [45], and subsequent work has explored uncertainty measures and their properties across tasks [46,47]. Bayesian and ensemble-based approaches further formalize predictive uncertainty for deep models and show how uncertainty can guide data selection and learning progress in practical settings [28,48]. The literature supports the idea that uncertainty derived from predictions can serve as a lightweight signal for guiding where learning effort should be concentrated, which aligns with our use of normalized entropy to coordinate intra-job adaptation and inter-job scheduling [49,50].

Recent studies have also considered broader forms of heterogeneous and large-scale learning beyond shared-cluster scheduling. For example, personalized federated learning has been explored for heterogeneous medical image analysis tasks, highlighting the importance of adaptive coordination under heterogeneous data and training conditions [51]. In addition, recent work on knowledge distillation and teacher–student learning in medical imaging emphasizes efficient model transfer and scalable training strategies in complex learning environments [52]. Recent advances in vision–language models for medical image analysis further illustrate the growing interest in general large-model frameworks for complex multimodal learning settings [53]. Although these directions address different problem settings from shared-cluster job scheduling, they help contextualize the broader need for adaptive learning-aware mechanisms in modern large-scale training systems.

3. Problem Formulation

We consider a shared GPU cluster that runs multiple distributed training jobs concurrently. Let $\mathcal{J}$ denote the set of active jobs. The cluster consists of a set of GPU nodes $\mathcal{N}$ with capacity $G = |\mathcal{N}|$ GPUs. Each job $j \in \mathcal{J}$ trains a model with C_j classes using data parallelism over a set of workers $\mathcal{R}_j$ with cardinality R_j . Time is indexed by epochs $t \in \{0, 1, 2, \dots\}$, and decisions are made at epoch boundaries.

We focus on reducing time to accuracy under shared resource constraints. Let $A_{j,t}$ denote the validation metric of job j at the end of epoch t and let A_j^{tar} be a target level. We define the completion epoch of job j as

$$T_j = \min\{t \geq 0 : A_{j,t} \geq A_j^{\text{tar}}\}. \quad (1)$$

The system aims to reduce completion times across jobs while satisfying the cluster GPU budget at each epoch.

3.1. Learning Signal from Predictive Uncertainty

For a sample x processed by job j at epoch t , let $p_{j,t}(x) \in \mathbb{R}^{C_j}$ be the predictive distribution produced by the model. We define sample level uncertainty using normalized entropy:

$$\tilde{h}_{j,t}(x) = \frac{-\sum_{c=1}^{C_j} p_{j,t,c}(x) \log p_{j,t,c}(x)}{\log C_j}. \quad (2)$$

Each worker $r \in \mathcal{R}_j$ processes a shard $\mathcal{D}_{j,r,t}$ during epoch t and reports a worker-level mean uncertainty:

$$H_{j,r,t} = \mathbb{E}_{x \sim \mathcal{D}_{j,r,t}} [\tilde{h}_{j,t}(x)]. \quad (3)$$

The expectation is estimated by a sample mean over minibatches processed during the epoch. To compare workers within the same job, we normalize across workers using a small positive constant $\varepsilon_H > 0$. Here, ε_H denotes a numerical stabilizer introduced only to prevent division by zero or unstable normalization when the summed uncertainty becomes very small.

$$\hat{H}_{j,r,t} = \frac{H_{j,r,t}}{\sum_{s \in \mathcal{R}_j} H_{j,s,t} + \varepsilon_H}. \quad (4)$$

When an additional ambiguity signal is used, it is blended with $\hat{H}_{j,r,t}$ to form a single worker score $u_{j,r,t}$. Otherwise, we set $u_{j,r,t} = \hat{H}_{j,r,t}$.

We define normalized worker weights that summarize where learning difficulty is concentrated within a job. Let $u_{j,r,t}$ denote the resulting worker score, then the weights satisfy

$$w_{j,r,t} \geq 0, \quad \sum_{r \in \mathcal{R}_j} w_{j,r,t} = 1, \quad w_{j,r,t} = \frac{u_{j,r,t}}{\sum_{s \in \mathcal{R}_j} u_{j,s,t} + \varepsilon_W}, \quad (5)$$

where $\varepsilon_W > 0$ is a small positive constant used only for numerical stability in the denominator. In practice, the weights are computed using this stabilized normalization together with temporal smoothing to avoid oscillations across epochs.

3.2. Micro-Level Control Variables

At each epoch boundary, each job chooses its next epoch internal configuration based on the weights $w_{j,r,t}$. The configuration is represented by three decision variables for epoch $t + 1$.

First, the job chooses a per epoch sample budget $N_{j,t+1}$ and allocates it across workers as

$$N_{j,r,t+1} = N_{j,t+1} w_{j,r,t}, \quad \sum_{r \in \mathcal{R}_j} N_{j,r,t+1} = N_{j,t+1}. \quad (6)$$

In implementations, integer sample counts are realized by standard rounding and correction while preserving the equality. Second, the job chooses per worker batch sizes subject to bounds and a job-level batch budget. Let $b_{j,r,t+1}$ denote the batch size of worker r for epoch $t + 1$. The batch sizes satisfy

$$b_{\min} \leq b_{j,r,t+1} \leq b_{\max}, \quad \sum_{r \in \mathcal{R}_j} b_{j,r,t+1} = B_{j,t+1}, \quad (7)$$

where $B_{j,t+1}$ is the job-level batch budget for epoch $t + 1$.

Third, the job adapts its learning rate as a function of job-level uncertainty. We define a job-level uncertainty summary as

$$U_{j,t} = \sum_{r \in \mathcal{R}_j} w_{j,r,t} u_{j,r,t}. \quad (8)$$

The learning rate $\eta_{j,t+1}$ is selected by an update rule $\eta_{j,t+1} = f(U_{j,t}, \eta_{j,t})$ that increases stability when uncertainty is high and allows faster progress when uncertainty is low. The specific form of f is part of the proposed control policy.

3.3. Macro-Level Scheduling Variables and Constraints

At each epoch boundary, the scheduler assigns integer GPU quotas $g_{j,t+1}$ to jobs for epoch $t + 1$ under the cluster budget

$$\sum_{j \in \mathcal{J}} g_{j,t+1} \leq G, \quad g_{j,t+1} \in \mathbb{Z}_{\geq 0}. \quad (9)$$

The scheduler uses uncertainty-based learning utility as an input signal. We define a short-horizon progress proxy from the reported uncertainty summaries:

$$\Delta U_{j,t} = U_{j,t-1} - U_{j,t}, \quad (10)$$

and define a utility score:

$$S_{j,t} = \lambda U_{j,t} + (1 - \lambda) \max(\Delta U_{j,t}, 0), \quad (11)$$

where $\lambda \in [0, 1]$ balances the uncertainty level and recent progress. The GPU allocation decision is produced by a scheduling policy $g_{j,t+1} = \pi(S_{j,t}, \mathcal{J}, G)$ that maps utility scores and system constraints to integer quotas. The specific form of π is part of the proposed scheduling policy.

3.4. Linking Macro Allocation to Micro Budgets

The macro-level quota determines the next epoch processing budget of each job. Let $\rho_{j,t}$ denote the measured per GPU processing rate of job j at epoch t in samples per epoch per GPU. We set the next epoch sample budget as

$$N_{j,t+1} = g_{j,t+1} \rho_{j,t}. \quad (12)$$

This creates a closed loop. The scheduler changes $g_{j,t+1}$ based on uncertainty guided utility, which changes $N_{j,t+1}$. The job then distributes $N_{j,t+1}$ across workers and adapts batch sizes and learning rate based on the same uncertainty signal.

4. Proposed Method

4.1. Overview and Timing of the Control Loop

Figure 1 illustrates our entropy-guided hierarchical method that couples micro-level adaptation and macro-level scheduling through a shared learning signal in a shared GPU cluster. The control loop is organized around epoch boundaries, which provide a natural synchronization point already present in most data-parallel training pipelines and allow lightweight statistics aggregation without interfering with per step synchronization.

The workflow proceeds as follows. During an epoch, workers execute standard synchronous data-parallel training and locally accumulate uncertainty statistics computed from logits. At the epoch boundary, these per worker statistics are aggregated within each job to construct stable worker weights. With a one-epoch delay, the micro-level controller then uses these weights to update the next epoch's internal configuration by adjusting the data-sharding ratios in proportion to uncertainty, tuning per worker batch sizes within safe bounds, and modulating the job-level learning rate.

At the same boundary, each job also exposes a compact job score derived from the same uncertainty statistics to the macro-level scheduler. The scheduler uses this score to update integer GPU quotas for the next epoch, which determines each job's data-processing budget under cluster contention. In this way, Figure 1 represents a closed hierarchical loop: macro-level control determines how much compute a job receives, while micro-level control determines how that compute is distributed and used within the job. Both levels are therefore coordinated through the same entropy-based signal, which provides a consistent interface between learning dynamics and scheduling decisions.

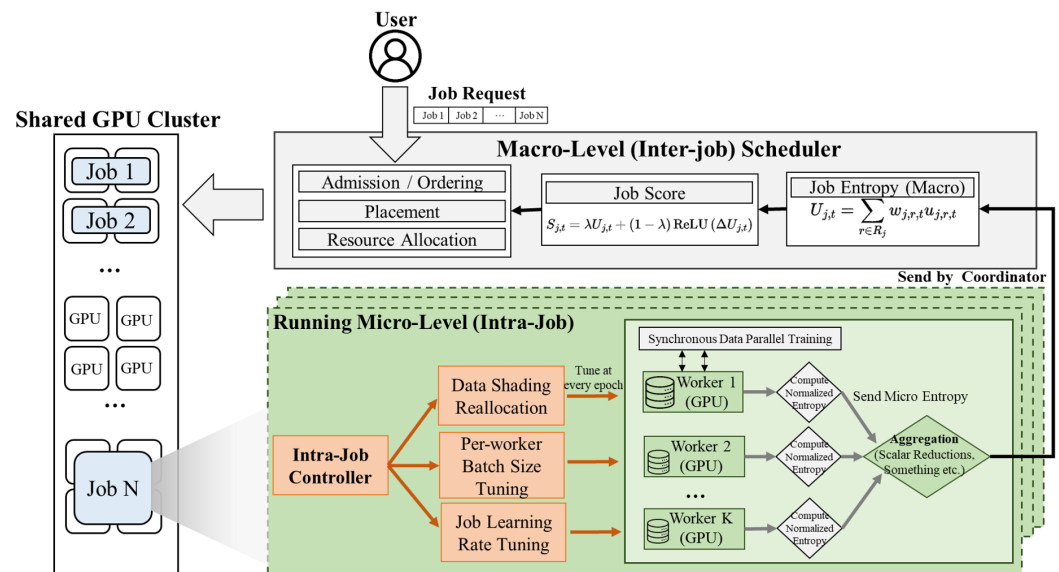


Figure 1. System overview of the entropy-guided hierarchical control loop. At epoch boundaries, workers compute normalized entropy from logits and aggregate it to form micro-level weights, which drive sharding, per worker batch sizes, and job-level learning-rate modulation. The same signal is summarized as a job score and sent to the macro-level scheduler to update GPU quotas for the next epoch.

4.2. Uncertainty Estimation from Logits

Consider a job j with C_j classes and workers $\mathcal{R}_j$. For a sample x at epoch t , the model produces logits $z_{j,t}(x) \in \mathbb{R}^{C_j}$ and probabilities

$$p_{j,t}(x) = \text{softmax}(z_{j,t}(x)). \quad (13)$$

This choice is deliberate from a systems perspective. Logits and softmax probabilities are already available during training, so uncertainty can be computed without additional forward passes, auxiliary models, or extra labels. We use Entropy because it reflects how spread the predictive distribution is over classes:

$$h_{j,t}(x) = - \sum_{c=1}^{C_j} p_{j,t,c}(x) \log p_{j,t,c}(x), \quad (14)$$

and we normalize it to keep the signal comparable across tasks with different numbers of classes:

$$\tilde{h}_{j,t}(x) = \frac{h_{j,t}(x)}{\log C_j}. \quad (15)$$

Normalized entropy is important in multi-job environments because different jobs may have different class counts, and the scheduler must compare job scores on a consistent scale. When needed, we complement Entropy with a margin-based ambiguity measure that captures near ties between the top two classes. Let $p_{(1)}(x)$ and $p_{(2)}(x)$ be the largest and second largest entries of $p_{j,t}(x)$. We set

$$q_{j,t}(x) = 1 - (p_{(1)}(x) - p_{(2)}(x)). \quad (16)$$

Entropy reflects global spread over classes, while margin ambiguity focuses on local decision boundaries. In practice, Entropy alone is often sufficient, and ambiguity is used as an optional complement when boundary confusion is a dominant mode.

Each worker r processes its local shard $\mathcal{D}_{j,r,t}$. We aggregate uncertainty at the worker level by epoch means:

$$H_{j,r,t} = \mathbb{E}_{x \sim \mathcal{D}_{j,r,t}} [\tilde{h}_{j,t}(x)], \quad Q_{j,r,t} = \mathbb{E}_{x \sim \mathcal{D}_{j,r,t}} [q_{j,t}(x)]. \quad (17)$$

This aggregation step serves two roles. It compresses per sample uncertainty into a small set of scalars per worker, which is communication-friendly, and it provides a stable estimate of how informative the worker data stream is over the duration of an epoch.

4.3. Robust Weight Construction for Uncertainty-Guided Control

We normalize the worker statistics within the job:

$$\hat{H}_{j,r,t} = \frac{H_{j,r,t}}{\sum_{s \in \mathcal{R}_j} H_{j,s,t}}, \quad \hat{Q}_{j,r,t} = \frac{Q_{j,r,t}}{\sum_{s \in \mathcal{R}_j} Q_{j,s,t}}. \quad (18)$$

Normalization ensures that weights represent relative importance among workers of the same job, independent of the absolute magnitude of uncertainty. We combine them into a single worker score:

$$u_{j,r,t} = \alpha \hat{H}_{j,r,t} + (1 - \alpha) \hat{Q}_{j,r,t}, \quad (19)$$

where α controls the mixture between normalized entropy and margin ambiguity. From a control perspective, $u_{j,r,t}$ is the raw signal that indicates where learning is currently concentrated inside the job.

Raw signals can oscillate due to minibatch stochasticity and due to changes in effective parallelism under multi-job contention. We therefore introduce guardrails that make the controller robust. We first bound the score

$$u_{j,r,t}^{\text{clip}} = \text{clip}(u_{j,r,t}, w_{\min}, w_{\max}), \quad (20)$$

which prevents a single worker from dominating and also prevents any worker from being starved. We then apply exponential smoothing:

$$\tilde{w}_{j,r,t} = \beta w_{j,r,t-1} + (1 - \beta) u_{j,r,t}^{\text{clip}}, \quad (21)$$

so that allocations evolve gradually across epochs rather than reacting aggressively to short-term fluctuations. Finally, we renormalize to obtain simplex weights:

$$w_{j,r,t} = \frac{\tilde{w}_{j,r,t}}{\sum_{s \in \mathcal{R}_j} \tilde{w}_{j,s,t}}. \quad (22)$$

These weights are the central interface between learning and systems. Inside a job, they determine how data and compute are redistributed across workers for the next epoch. Across jobs, a job-level summary derived from the same weights drives cluster scheduling decisions.

4.4. Micro-Level Controls at Epoch Boundaries

We apply three controls at epoch $t + 1$. The key design choice is that all controls are applied to the next epoch, which makes them compatible with standard synchronous training and avoids interfering with per step synchronization.

Let $N_{j,t+1}$ be the total number of samples to be processed by job j in epoch $t + 1$ under its current resource budget. We allocate per worker sample counts by

$$\bar{N}_{j,r,t+1} = N_{j,t+1} w_{j,r,t}, \quad N_{j,r,t+1} = \text{round}(\bar{N}_{j,r,t+1}). \quad (23)$$

This rule shifts data exposure toward workers whose data stream is currently more uncertain, which increases the proportion of updates drawn from ambiguous examples. Since rounding may violate the exact total, we enforce the budget constraint

$$\sum_{r \in \mathcal{R}_j} N_{j,r,t+1} = N_{j,t+1} \quad (24)$$

using a remainder correction that assigns leftover samples to the largest fractional parts. This keeps the total work per epoch comparable across methods and isolates the effect of redistribution.

We map weights to batch sizes within bounds $b_{\min}$ and $b_{\max}$ as

$$b_{j,r,t+1} = \text{clip}\left(\text{round}(b_{\min} + (b_{\max} - b_{\min})w_{j,r,t}), b_{\min}, b_{\max}\right). \quad (25)$$

Batch size interacts with gradient noise and synchronization. Prior work has shown that batch size and learning rate are coupled through their effect on optimization stability and gradient noise scale, and that conservative joint adaptation can improve training robustness under changing effective regimes. Under contention, communication delays or reduced replica counts effectively change the optimization regime seen by the job. Our mapping is therefore intended as a bounded control rule that reallocates per worker batch sizes according to uncertainty while keeping the update within a safe operating range. The role of this rule is not to claim a new optimization theorem, but to provide a lightweight and stable mechanism that is consistent with prior observations on batch-size/learning-rate coupling and large-batch training behavior. The bounds ensure predictable memory usage and avoid sudden jumps that can destabilize training.

We summarize job uncertainty using the weighted score

$$U_{j,t} = \sum_{r \in \mathcal{R}_j} w_{j,r,t} u_{j,r,t}. \quad (26)$$

This scalar captures the uncertainty state of the job at the end of epoch t and serves as a feedback measurement. We then update the learning rate using

$$\eta_{j,t+1} = \eta_{j,t} \left(\frac{U_{j,t}}{U_{j,0}} \right)^{-\gamma}, \quad (27)$$

where $\gamma > 0$ controls the sensitivity of the learning-rate modulation to changes in job-level uncertainty. Larger values of γ make the controller react more aggressively to uncertainty fluctuations, whereas smaller values lead to more conservative updates. In our implementation, γ is set to provide stable epoch-level adaptation without causing abrupt changes in the optimizer state. When uncertainty rises relative to the initial stage, the learning rate is reduced to prevent overly aggressive updates. When uncertainty declines, the learning rate relaxes to maintain efficient progress. Because the update uses an epoch-level measurement, it remains stable and adds negligible overhead.

This design also helps mitigate potential conflicts between local adaptation and global scheduling. Macro-level allocation determines how much resource budget a job receives, while the micro-level controller regulates how that budget is used so that optimization remains stable under the resulting effective training regime. In this sense, the two levels are coordinated through the same uncertainty signal, but they need not have identical objectives at every moment; rather, the shared signal provides a consistent interface that reduces mismatch between learning-aware adaptation and scheduling decisions.

4.5. Macro-Level Scheduling via Uncertainty-Guided Utility

The macro-level scheduler operates at epoch boundaries and uses the same uncertainty signal as the micro-level controller, aggregated at the job level. From (26), each job reports $U_{j,t}$ once per epoch. We define the short-horizon progress proxy

$$\Delta U_{j,t} = U_{j,t-1} - U_{j,t}, \quad (28)$$

and form a job utility score that captures both current learning demand and near-term learning progress

$$S_{j,t} = \lambda U_{j,t} + (1 - \lambda) \text{ReLU}(\Delta U_{j,t}), \quad (29)$$

where $\lambda \in [0, 1]$ balances the contribution of the current uncertainty level and the recent uncertainty reduction trend. A larger λ places more emphasis on present learning demand, whereas a smaller λ gives relatively more weight to short-term progress. In our implementation, λ is set to provide a stable compromise between these two signals so that the scheduler remains responsive without overreacting to short-term fluctuations. A higher $S_{j,t}$ indicates that allocating resources to job j is expected to yield a stronger reduction in uncertainty and thus faster convergence.

The use of $S_{j,t}$ as a scheduling priority is based on the following interpretation. A job with persistently high uncertainty still has substantial unresolved learning difficulty, while a positive recent decrease in uncertainty indicates that the job is converting computation into useful progress. The score therefore combines current learning demand with short-horizon responsiveness to additional compute. This does not imply that micro-level utility and macro-level priority are always identical in a strict optimization sense. Rather, the same normalized signal is used to align the two levels heuristically so that jobs are prioritized not only by resource occupancy but also by their expected learning benefit under contention.

At each decision point, the scheduler ranks runnable candidates by utility score rather than submission order. For newly arrived or still pending jobs, $U_{j,t}$ may be unavailable. The scheduler then falls back to a default prior $U_{j,\text{init}}$ or uses a short warm-up measurement. The scheduler selects the highest-scoring job that fits within the currently idle GPUs. This defines an entropy-guided admission rule in which execution order is determined by learning utility.

Let the cluster have G GPUs. The scheduler assigns integer quotas $g_{j,t+1}$ such that

$$\sum_{j \in \mathcal{J}} g_{j,t+1} \leq G. \quad (30)$$

We use proportional allocation to translate utility into quotas:

$$\tilde{g}_{j,t+1} = G \cdot \frac{S_{j,t}}{\sum_{k \in \mathcal{J}} S_{k,t}}, \quad g_{j,t+1} = \text{clip}(\text{round}(\tilde{g}_{j,t+1}), g_{\min}, g_{\max}), \quad (31)$$

with remainder correction to satisfy the budget. This ensures stable integer allocations while preserving the monotonic relationship between $S_{j,t}$ and allocated resources.

In practical clusters, quotas must be realized by assigning jobs to concrete GPU nodes. Let $\mathcal{N}$ be the set of nodes. We represent placement by $x_{j,n,t+1} \in \{0, 1\}$ and enforce

$$\sum_{n \in \mathcal{N}} x_{j,n,t+1} = g_{j,t+1}, \quad \sum_{j \in \mathcal{J}} x_{j,n,t+1} \leq 1 \quad \forall n \in \mathcal{N}. \quad (32)$$

Among feasible placements, the macro-level scheduler may use a deterministic ordering of assigned nodes to choose a master rank and may prefer placements that reduce expected contention. Importantly, placement does not change the learning signal itself. In-

stead, it realizes the entropy-guided decision of which jobs run and how much parallelism they receive.

The key interaction remains that $g_{j,t+1}$ determines each job's next epoch processing budget $N_{j,t+1}$, and the micro-level controller then distributes that budget across workers using (23)–(27). This closes the loop between macro-level execution decisions and micro-level adaptation, both driven by the same entropy-based utility.

4.6. End-to-End Scheduling Algorithm

Algorithm 1 summarizes the method as an epoch-level control loop. We use the epoch boundary as the control point because it already provides a natural synchronization barrier in data-parallel training, and it allows uncertainty statistics to be aggregated with negligible overhead. The algorithm applies a one epoch delay in all control actions, meaning that weights computed at epoch t determine sharding, batch size, and learning rate for epoch $t + 1$, which avoids interfering with per step synchronization. At the cluster level, job scores are computed from the same uncertainty signal and translated into GPU quotas that determine the next epoch data budget, closing the loop between macro-level allocation and micro-level adaptation.

Algorithm 1 Entropy-guided hierarchical scheduling

Require: Active jobs $\mathcal{J}$, workers $\mathcal{R}_j$ for each job, cluster GPUs G
Require: Hyperparameters $\alpha, \beta, w_{\min}, w_{\max}, b_{\min}, b_{\max}, \gamma, \lambda$

- 1: Initialize $w_{j,r,0} \leftarrow 1/|\mathcal{R}_j|$ and learning rate $\eta_{j,0}$ for all jobs and workers
- 2: **for** each epoch $t = 0, 1, 2, \dots$ **do**
- 3: **for all** jobs $j \in \mathcal{J}$ in parallel **do**
- 4: **for all** workers $r \in \mathcal{R}_j$ in parallel **do**
- 5: Train for one epoch using current shard and batch size
- 6: Accumulate $\tilde{h}_{j,t}(x)$ and optionally $q_{j,t}(x)$ from logits
- 7: Compute $H_{j,r,t}$ and $Q_{j,r,t}$ as epoch means
- 8: **end for**
- 9: Compute $\hat{H}_{j,r,t}$ and $\hat{Q}_{j,r,t}$ using (18)
- 10: Compute $u_{j,r,t}$ using (19)
- 11: Compute $w_{j,r,t}$ using (20), (21), and (22)
- 12: Compute job uncertainty $U_{j,t}$ using (26)
- 13: Compute job score $S_{j,t}$ using (28) and (29)
- 14: **end for**
- 15: Scheduler computes next quotas $g_{j,t+1}$ for all jobs using (31) under (30)
- 16: **for all** jobs $j \in \mathcal{J}$ **do**
- 17: Derive next epoch data budget $N_{j,t+1}$ from quota $g_{j,t+1}$
- 18: Compute $N_{j,r,t+1}$ using (23) and enforce (24)
- 19: Compute $b_{j,r,t+1}$ using (25)
- 20: Update $\eta_{j,t+1}$ using (27)
- 21: **end for**
- 22: **end for**

The method is designed as a thin layer on top of an existing data-parallel training stack and a cluster scheduler. Uncertainty computation is performed during the forward pass using logits already produced for the loss, and thus requires no additional model evaluations, forward/backward passes, or extra labels. Each worker accumulates running sums of normalized entropy, and it may also track a margin-based ambiguity statistic, along with a counter of processed samples. At the epoch boundary, the job performs a small number of collective operations to aggregate these scalars and to compute the normalized quantities needed for weight construction. Because only compact scalar statistics are exchanged once per epoch, the additional communication cost remains small compared with standard gradient synchronization and does not require changes to gradient all-

reduce or per step training logic. This design keeps the control overhead lightweight while remaining compatible with standard synchronous data-parallel training.

Data-sharding adaptation can be implemented by updating the sampler at each epoch boundary. When datasets are represented by index lists, the controller assigns each worker a contiguous or strided subset of size $N_{j,r,t+1}$ while ensuring that the union matches the intended epoch budget. The remainder correction in (24) can be applied deterministically to preserve reproducibility. Batch-size updates can be applied by re-instantiating each worker data loader with the new $b_{j,r,t+1}$ or by using a loader that supports dynamic batch sizes. The bounds $b_{\min}$ and $b_{\max}$ can be chosen once per job through lightweight profiling and then kept fixed. The learning-rate update in (27) is applied at the epoch boundary through the optimizer parameter group. For macro-level scheduling, each job reports the scalar score $S_{j,t}$ once per epoch, and the scheduler converts it into the next-epoch quota and data budget $N_{j,t+1}$ through a throughput model or measured step time. This separation keeps the system modular. The scheduler consumes only a compact job-level summary, while the training code consumes only the assigned quota and does not require access to other jobs.

5. Experimental Setup

Testbed. Experiments run on a shared GPU cluster with four NVIDIA RTX A5000 GPUs. All methods share the same physical cluster to induce realistic multi-tenant contention on compute and communication. Table 1 summarizes the hardware and software environment.

Table 1. Cluster and software environment.

Item	Configuration
Cluster size	4 nodes (one worker per GPU per node)
GPU per node	1 × NVIDIA RTX A5000 (24 GB VRAM)
CPU	Intel Xeon Gold 6326 (32 cores)
System memory	32 GB RAM per node
Interconnect	100 GbE
Storage	Local NVMe and NFS/Lustre for datasets
Operating system	Ubuntu 20.04.6 LTS
CUDA/cuDNN	12.2/9.1.0
PyTorch	2.6.0 (DDP backend: gloo)
All-reduce bucket size	PyTorch default (25 MB)
Precision	AMP (fp16)

Workload. We submit fifty jobs drawn from eight job types. Each job type fixes the model, dataset, base batch size, and training mode. Job types J1 to J3 run for fifty epochs, while J4 to J8 run for twenty epochs. The submission mix follows the configured multiplicities and priority levels, where J1 to J3 have priority one, J4 to J6 have priority two, and J7 to J8 have priority three. Table 2 summarizes all job types and the workload composition.

Methods and metrics. We compare the proposed scheduler against FIFO, shortest job first and Lucid. We evaluate system performance using JCT, makespan, and the cumulative distribution function of JCT with emphasis on tail behavior. We evaluate learning quality using Top-1 accuracy and loss for MNIST, Fashion MNIST, CIFAR10, and CIFAR100, and Top-1 and Top-5 accuracy with loss for Tiny ImageNet. Table 3 summarizes the training configuration shared across methods and the control ranges used by the proposed approach.

Large-scale simulation. To study scaling beyond the four-GPU testbed, we evaluate the same workload using a calibrated simulator. The simulator models step time and communication overhead as a function of allocated GPUs and contention, with parameters

calibrated from measurements on the real cluster. We evaluate performance at 64, 128, 256, 512 and 1024 nodes using the simulator.

Table 2. Job types and workload composition. Mode indicates training from scratch or training with a frozen backbone.

Job Type	Model	Dataset	Mode	Epochs	Batch	Count
J1	VGG16	MNIST	scratch	50	16	8
J2	MobileNet	Fashion MNIST	scratch	50	16	8
J3	ResNet18	CIFAR10	scratch	50	16	8
J4	EfficientNet V2	CIFAR10	freeze	20	16	6
J5	ResNet50	CIFAR100	scratch	20	16	5
J6	ShuffleNet	CIFAR100	freeze	20	16	5
J7	ConvNeXt Tiny	Tiny ImageNet	scratch	20	16	5
J8	DenseNet121	Tiny ImageNet	scratch	20	16	5

Table 3. Training configuration and control hyperparameters.

Item	Setting
Optimizer	Adam
Base learning rate	0.001
Momentum or betas	default Adam betas (0.9, 0.999)
Loss	CrossEntropyLoss
Initial batch size	16
Batch size bounds $b_{\min}, b_{\max}$	8 and 64
Weight smoothing β	0.3
Weight clipping bounds $w_{\min}, w_{\max}$	0.5 and 1.5
Learning-rate bounds	10^{-5} and 10^{-2}
Gradient accumulation step bounds	20 and 64

5.1. Experimental Results

5.1.1. Overall System Performance

Table 4 summarizes overall system performance in terms of average JCT, average queueing delay, and workload makespan, all measured in hours. We define JCT as the wall-clock time from job submission to completion, and queueing delay as the time from submission to the first GPU allocation. We define makespan as the elapsed time between the earliest submission and the last job completion in the workload.

Across the 50-job workload, Entropy achieves the lowest average JCT of 5.69 h, compared to 7.46 h for Lucid, 9.83 h for SFJ, and 10.12 h for FIFO. This corresponds to a 23.7% reduction over Lucid and a 43.8% reduction over FIFO. Entropy also reduces average queueing delay from 6.80 to 4.71 h, a 30.7% reduction, indicating that utility-guided admission and quota updates alleviate waiting under multi-tenant contention. Finally, Entropy shortens the workload makespan from 17.27 to 14.56 h, a 15.7% reduction over Lucid, improving cluster-level throughput relative to the baselines. These results suggest that aligning admission and resource allocation with learning utility can simultaneously reduce both per job latency and end-to-end workload completion time.

Table 4. Average JCT, average queueing delay, and workload makespan (hours).

Scheduler	Average JCT (h)	Avg Queueing Delay (h)	Makespan (h)
FIFO	10.12	9.33	20.00
SFJ	9.83	9.09	18.68
Lucid	7.46	6.80	17.27
Entropy (Ours)	5.69	4.71	14.56

Entropy consistently improves the upper tail of the distribution. Table 5 highlights this effect in the percentile summary. Compared to Lucid, Entropy reduces the 95th-percentile JCT from 15.03 to 11.39 h, corresponding to a 24.2% reduction, and reduces the maximum observed JCT from 17.23 to 14.52 h, corresponding to a 15.7% reduction. Entropy also improves the median (P50) from 6.72 to 4.60 h, corresponding to a 31.6% reduction, indicating that the latency improvement is not limited to a few outliers.

Table 5. JCT distribution summary (hours).

Scheduler	P50	P90	P95	P99	Max
FIFO	10.01	17.06	18.48	19.82	19.95
SFJ	10.00	17.87	18.26	18.55	18.66
Lucid	6.72	14.26	15.03	16.44	17.23
Entropy (Ours)	4.60	11.05	11.39	13.13	14.52

Figure 2 shows that Entropy shifts the JCT distribution left across a broad range of quantiles, rather than improving only a small subset of jobs. At the median, Entropy reduces JCT from 6.72 h for Lucid to 4.60 h, corresponding to a 31.6% reduction. The separation persists into the upper tail: the 90th-percentile decreases from 14.26 to 11.05 h, corresponding to a 22.5% reduction, and the 95th-percentile decreases from 15.03 to 11.39 h, corresponding to a 24.2% reduction. Moreover, the maximum observed JCT decreases from 17.23 to 14.52 h, corresponding to a 15.7% reduction, indicating that Entropy tightens the extreme tail under contention. Overall, the CDF indicates that Entropy improves completion times for a large fraction of jobs while also reducing tail risk, which aligns with the percentile summary in Table 5.

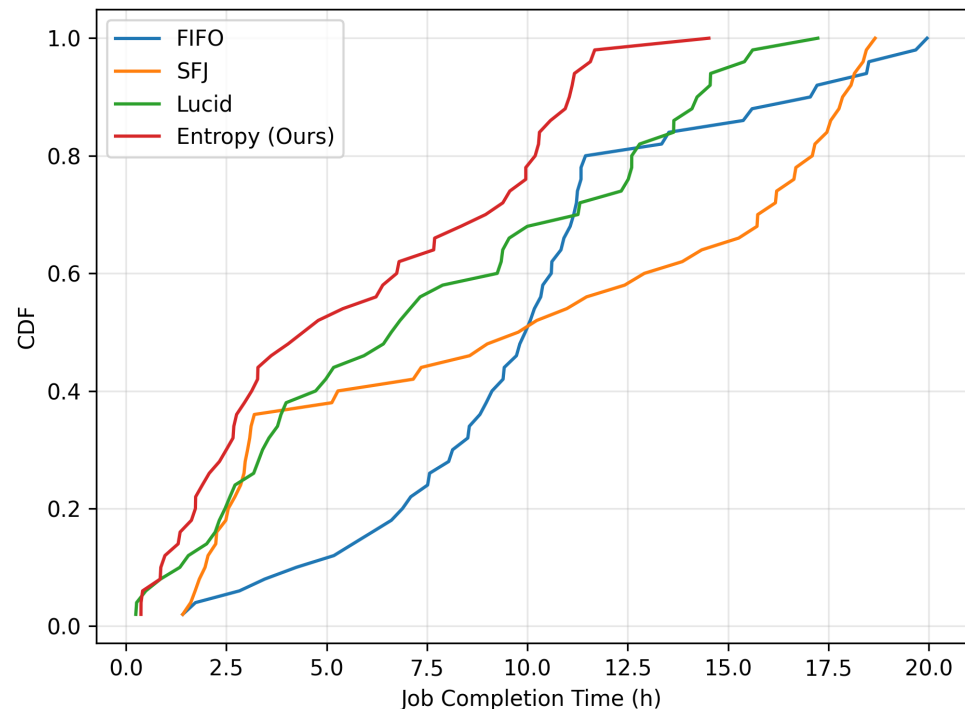


Figure 2. Empirical CDF of JCT (hours) across 50 jobs.

Figure 3 decomposes the average JCT into queueing delay from submission to start and service time from start to completion. The main improvement of Entropy comes from a substantial reduction in queueing delay. Compared to Lucid, Entropy lowers average queueing delay from 6.80 to 4.71 h, a 30.7% reduction, which accounts for most of the 1.77 h

reduction in average JCT from 7.46 to 5.69 h. In contrast, the average service time under Entropy is slightly higher than Lucid, increasing from 0.66 to 0.98 h. This suggests that the benefit of the proposed method mainly comes from improved admission and GPU-quota decisions under contention, rather than from uniformly shortening execution time after a job starts. A plausible reason is that micro-level adaptation prioritizes learning-aware redistribution and stability, which can introduce conservative batch-size or learning-rate adjustments during some phases of training. As a result, the method may improve overall time to accuracy at the cluster level while not always minimizing raw per job service time. Potential optimizations include less frequent control updates, threshold-based adaptation, and more conservative tuning of the micro-level control parameters.

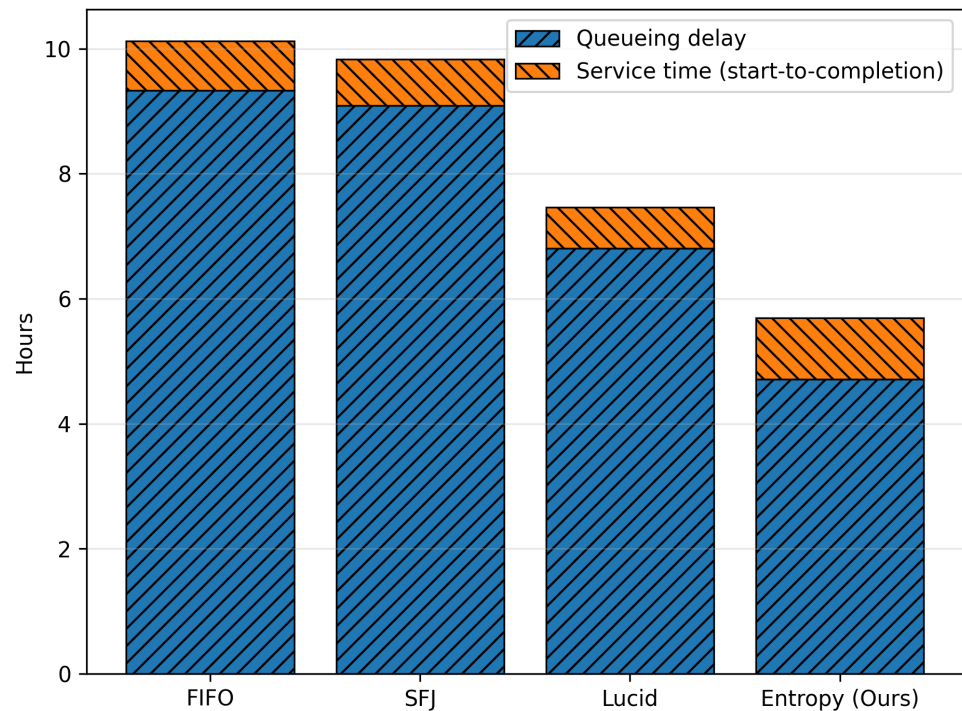


Figure 3. Decomposition of average JCT (hours) into queueing delay and service time.

This result also helps clarify the computational overhead and runtime cost of the proposed method relative to the baseline schedulers. Although the entropy-guided control logic introduces additional adaptation decisions during training, its runtime effect appears as a moderate increase in service time rather than a large penalty in end-to-end latency. In our results, this overhead is outweighed by the larger reduction in queueing delay, so the net impact on overall JCT remains favorable. Therefore, the proposed method should be understood as introducing a lightweight runtime trade-off in exchange for improved cluster-level scheduling efficiency under contention.

5.1.2. Training Accuracy and Convergence

In addition to system-level latency, we examine whether different scheduling policies preserve training accuracy and convergence behavior. Table 6 reports the average best validation accuracy achieved by each job type under each scheduler, computed as the maximum validation accuracy attained during each run.

Entropy achieves the highest average accuracy in 7 out of 8 job types. Compared to Lucid, Entropy improves average best accuracy by 0.10–2.45 percentage points in seven job types, with an average gain of approximately 0.88 percentage points across all eight job types. Compared to FIFO, Entropy improves average best accuracy by 0.21–3.61 percentage

points in seven job types, and remains within 0.96 percentage points in the remaining job type. Taken together, these results show that Entropy maintains competitive model quality across job types and often achieves higher accuracy than the baselines. Alongside the improvements in JCT and tail latency, this indicates that the proposed scheduler improves cluster responsiveness while preserving learning outcomes under multi-tenant contention.

Table 6. Average best accuracy (%) by job type. For each job type, the best average accuracy across schedulers is highlighted in bold.

Type	FIFO	SFJ	Lucid	Entropy (Ours)
J1	99.47	99.44	99.49	99.68
J2	92.09	92.09	92.10	92.20
J3	84.60	84.61	85.28	86.55
J4	31.44	30.95	31.52	32.80
J5	64.34	64.57	66.16	67.92
J6	25.32	25.04	24.64	24.36
J7	46.53	47.20	47.72	49.01
J8	66.85	66.82	68.01	70.46

5.1.3. Ablation Study

To isolate the contribution of the macro-level and micro-level components, we conduct an ablation study on a workload of 25 jobs drawn from job types J1 to J9. We compare three variants. The macro-only variant enables inter-job scheduling and quota updates only. The micro-only variant enables intra-job adaptation only while keeping inter-job scheduling fixed. The full method enables both components.

Table 7 summarizes the results. The full method achieves the best performance across all system-level metrics. Compared to macro-only, the full method reduces average JCT from 3.32 to 2.39 h, a 28.0% reduction, and reduces makespan from 8.61 to 6.77 h, a 21.4% reduction. Compared to micro-only, the full method reduces average JCT from 4.17 to 2.39 h, a 42.7% reduction, and reduces makespan from 10.04 to 6.77 h, a 32.6% reduction. These results indicate that combining macro-level allocation with micro-level adaptation yields complementary benefits under multi-tenant contention.

Table 7. Ablation study results on 25 jobs (J1–J9). All metrics are measured in hours.

Metric	Macro-Only	Micro-Only	Full Entropy
Average JCT (h)	3.32	4.17	2.39
Avg Queueing Delay (h)	2.03	2.91	1.77
Makespan (h)	8.61	10.04	6.77

Macro-level control reduces queueing delay by directly shaping admission and quota assignment, while micro-only adaptation has limited leverage over waiting time because the macro-level policy is fixed. In contrast, micro-level adaptation primarily improves per job training efficiency after GPUs are allocated, which helps translate allocated resources into faster progress. The gap between macro-only and the full method suggests that micro-level adaptation contributes additional gains beyond improved admission and quota decisions. Conversely, the gap between micro-only and the full method highlights the importance of learning-aware resource arbitration for improving cluster-level throughput under contention. Taken together, these ablation results support the practical rationale of the design. Although the macro-level and micro-level components contribute differently, the largest gains are obtained when both are coordinated through the shared uncertainty signal.

5.1.4. Scalability Under Large-Scale Simulation

We evaluate scalability using a calibrated simulator that models per step training time and communication overhead as a function of allocated GPUs. The simulator is parameterized to reflect the target homogeneous A5000 cluster setting, including throughput-related scaling behavior, communication efficiency under multi-GPU allocation, and scheduler-side overhead. Job templates are also configured using representative workload characteristics such as batch size, GPU demand, and training-progress parameters, so that the simulator provides a controlled approximation of large-scale distributed training behavior. The simulator is intended to extend the small-scale real-cluster evaluation to larger node counts under a consistent workload model, using the same scheduler logic and representative job characteristics. Because the simulator abstracts training into a throughput model, the absolute times in Table 8 should be interpreted primarily as relative comparisons across schedulers under controlled scaling, rather than as exact wall-clock durations of specific real-world training runs or production deployments.

Table 8. Scalability results from simulation. Average JCT and makespan are measured in seconds for 2000 jobs drawn from nine job types.

Nodes	Scheduler	Average JCT (s)	Makespan (s)
64	FIFO	16,832.2	27,682.8
64	SFJ	8262.8	21,993.3
64	Lucid	8022.6	18,759.0
64	Entropy (Ours)	6678.6	15,035.4
128	FIFO	8358.1	13,889.7
128	SFJ	4464.9	11,130.0
128	Lucid	4110.9	9535.0
128	Entropy (Ours)	3355.2	7637.4
256	FIFO	4171.7	7101.2
256	SFJ	2566.6	5620.6
256	Lucid	2144.3	4837.0
256	Entropy (Ours)	1685.5	3806.1
512	FIFO	2101.6	3751.3
512	SFJ	1448.2	2885.4
512	Lucid	1110.2	2588.0
512	Entropy (Ours)	854.0	1938.6
1024	FIFO	1082.9	2087.2
1024	SFJ	773.6	1638.0
1024	Lucid	593.3	1324.0
1024	Entropy (Ours)	444.0	1084.5

Table 8 reports average JCT and makespan in seconds for workloads of 2000 jobs as the cluster scales from 64 to 1024 nodes. As the number of nodes increases, average JCT and makespan decrease across all schedulers, reflecting increased parallel capacity. Entropy achieves the lowest average JCT and makespan at every scale in this simulated setting. At 1024 nodes, Entropy reduces average JCT from 593.3 s under Lucid to 444.0 s, a 25.2% reduction, and reduces makespan from 1324.0 s to 1084.5 s, an 18.1% reduction. These results suggest that the proposed policy preserves its advantage as the cluster size grows, consistently improving both per job latency and end-to-end workload completion time under the simulator model.

5.1.5. Practical Deployment Discussion

From a practical deployment perspective, the proposed framework is designed to operate as a lightweight control layer on top of existing shared-cluster scheduling and synchronous data-parallel training workflows. Because the control decisions are applied at epoch boundaries, the method can be integrated without intrusive changes to the standard per step training path. In production GPU clusters, however, deployment would still need to account for additional factors such as workload heterogeneity, varying contention patterns, scheduler policy constraints, and the trade-off between adaptation responsiveness and operational stability. These considerations suggest that deployment-oriented tuning of control frequency and parameter sensitivity will be important for robust operation in real multi-tenant environments.

6. Conclusions

In this study, we presented Entropy, an entropy-guided hierarchical scheduling approach that couples micro-level adaptation and macro-level allocation through a shared uncertainty signal computed from logits and aggregated at epoch boundaries. Across our multi-tenant A5000 testbed, Entropy substantially improves cluster responsiveness and tail latency. In particular, Entropy reduces the 95th-percentile JCT from 15.03 h under Lucid to 11.39 h, corresponding to a 24.2% reduction, and lowers the median JCT from 6.72 to 4.60 h, corresponding to a 31.6% reduction. These gains are driven primarily by shorter queueing delay, indicating that uncertainty-guided admission and quota decisions mitigate contention-induced waiting while maintaining competitive learning outcomes across job types.

The current study also has several limitations. First, the empirical evaluation is centered on controlled multi-tenant settings and primarily vision-oriented workloads, which do not fully represent the diversity of emerging distributed training scenarios, such as NLP, multimodal learning, and large-model fine-tuning. Second, although the proposed framework is designed to operate with lightweight epoch-level control, the current evaluation does not cover all practical systems effects that may arise in real production deployments, including stronger heterogeneity in hardware, workload interference, and broader failure or straggler conditions. Third, the large-scale results are based on a simulator-driven evaluation, which is useful for controlled comparative analysis but does not replace full validation in a production-scale environment.

In future work, we will extend the approach to a broader set of NLP and multimodal workloads, including large language model fine-tuning and other long-context tasks where learning dynamics and resource sensitivity differ from vision benchmarks. We will also evaluate Entropy on more diverse tasks and larger-scale distributed datasets under mixed batch sizes, mixed sequence lengths, and more heterogeneous training regimes. Finally, we will study deployment in heterogeneous clusters that combine different GPU generations and interconnects, and we will explore robustness to practical systems effects, such as interference-aware placement, checkpointing overheads, and straggler mitigation in large-scale distributed training.

Author Contributions: Conceptualization, T.-J.S.; Methodology, T.-J.S.; Software, T.-J.S.; Validation, T.-J.S.; Formal analysis, T.-J.S. and E.-N.H.; Investigation, T.-J.S. and E.-N.H.; Resources, E.-N.H.; Data curation, T.-J.S.; Writing—original draft, T.-J.S.; Writing—review & editing, T.-J.S. and E.-N.H.; Visualization, T.-J.S.; Supervision, E.-N.H.; Project administration, E.-N.H.; Funding acquisition, E.-N.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2026-RS-2024-00438239, 70%) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation) and supported by Global—Learning & Academic Research Institution for Master’s PhD students, and the Postdocs (G-LAMP) Program of the National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (No. RS-2025-25442355, 30%).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The source code developed in this study is not publicly available due to institutional restrictions. The experiments were conducted on publicly available datasets, which can be accessed from their respective sources.

Conflicts of 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.

References

1. Ye, Z.; Gao, W.; Hu, Q.; Sun, P.; Wang, X.; Luo, Y.; Zhang, T.; Wen, Y. Deep learning workload scheduling in gpu datacenters: A survey. *ACM Comput. Surv.* **2024**, *56*, 1–38. [\[CrossRef\]](#)
2. Peng, Y.; Bao, Y.; Chen, Y.; Wu, C.; Guo, C. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In Proceedings of the Thirteenth EuroSys Conference, Porto, Portugal, 23–26 April 2018.
3. Mahajan, K.; Balasubramanian, A.; Singhvi, A.; Venkataraman, S.; Akella, A.; Phanishayee, A.; Chawla, S. Themis: Fair and efficient GPU cluster scheduling. In Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20), Santa Clara, CA, USA, 25–27 February 2020.
4. Xiao, W.; Bhardwaj, R.; Ramjee, R.; Sivathanu, M.; Kwatra, N.; Han, Z.; Patel, P.; Peng, X.; Zhao, H.; Zhang, Q.; et al. Gandiva: Introspective cluster scheduling for deep learning. In Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), Carlsbad, CA, USA, 8–10 October 2018.
5. Gu, J.; Chowdhury, M.; Shin, K.G.; Zhu, Y.; Jeon, M.; Qian, J.; Liu, H.; Guo, C. Tiresias: A GPU cluster manager for distributed deep learning. In Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), Boston, MA, USA, 26–28 February 2019.
6. Zheng, P.; Pan, R.; Khan, T.; Venkataraman, S.; Akella, A. Shockwave: Fair and efficient cluster scheduling for dynamic adaptation in machine learning. In Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), Boston, MA, USA, 17–19 April 2023.
7. Zheng, P.; Pan, R.; Khan, T.; Venkataraman, S.; Akella, A. Astraea: A fair deep learning scheduler for multi-tenant gpu clusters. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 2781–2793. [\[CrossRef\]](#)
8. Kaur, R.; Asad, A.; Al Abdul Wahid, S.; Mohammadi, F. A Survey of Advancements in Scheduling Techniques for Efficient Deep Learning Computations on GPUs. *Electronics* **2025**, *14*, 1048. [\[CrossRef\]](#)
9. Hu, Q.; Ye, Z.; Wang, Z.; Wang, G.; Zhang, M.; Chen, Q.; Sun, P.; Lin, D.; Wang, X.; Luo, Y.; et al. Characterization of large language model development in the datacenter. In Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24), Santa Clara, CA, USA, 16–18 April 2024.
10. Qiao, A.; Choe, S.K.; Subramanya, S.J.; Neiswanger, W.; Ho, Q.; Zhang, H.; Ganger, G.R.; Xing, E.P. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21), Virtual, 14–16 July 2021.
11. Li, J.; Xu, H.; Zhu, Y.; Liu, Z.; Guo, C.; Wang, C. Lyra: Elastic scheduling for deep learning clusters. In Proceedings of the Eighteenth European Conference on Computer Systems, Rome, Italy, 8–11 May 2023.
12. Sharma, A.; Bhasi, V.M.; Singh, S.; Kesidis, G.; Kandemir, M.T.; Das, C.R. Gpu cluster scheduling for network-sensitive deep learning. *arXiv* **2024**, arXiv:2401.16492. [\[CrossRef\]](#)
13. Strati, F.; Ma, X.; Klimovic, A. Orion: Interference-aware, fine-grained gpu sharing for ml applications. In Proceedings of the Nineteenth European Conference on Computer Systems, Athens, Greece, 23–25 April 2024.
14. Weng, Q.; Yang, L.; Yu, Y.; Wang, W.; Tang, X.; Yang, G.; Zhang, L. Beware of fragmentation: Scheduling GPU-Sharing workloads with fragmentation gradient descent. In Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC 23), Santa Clara, CA, USA, 10–12 July 2023.
15. Yang, Z.Y.; Xia, W.K.; Chu, H.Q.; Su, W.H.; Wang, R.F.; Wang, H. A comprehensive review of deep learning applications in cotton industry: From field monitoring to smart processing. *Plants* **2025**, *14*, 1481. [\[CrossRef\]](#)

16. Wu, A.Q.; Li, K.L.; Song, Z.Y.; Lou, X.; Hu, P.; Yang, W.; Wang, R.F. Deep learning for sustainable aquaculture: Opportunities and challenges. *Sustainability* **2025**, *17*, 5084. [\[CrossRef\]](#)
17. Chen, J.; Pan, X.; Monga, R.; Bengio, S.; Jozefowicz, R. Revisiting distributed synchronous SGD. *arXiv* **2016**, arXiv:1604.00981.
18. Sergeev, A.; Del Balso, M. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv* **2018**, arXiv:1802.05799. [\[CrossRef\]](#)
19. Zhang, H.; Zheng, Z.; Xu, S.; Dai, W.; Ho, Q.; Liang, X.; Hu, Z.; Wei, J.; Xie, P.; Xing, E.P. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC 17), Santa Clara, CA, USA, 12–14 July 2017.
20. Jiang, Y.; Zhu, Y.; Lan, C.; Yi, B.; Cui, Y.; Guo, C. A unified architecture for accelerating distributed DNN training in heterogeneous GPU/CPU clusters. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), Virtual, 4–6 November 2020.
21. Provatas, N.; Konstantinou, I.; Koziris, N. A Survey on Parameter Server Architecture: Approaches for Optimizing Distributed Centralized Learning. *IEEE Access* **2025**, *13*, 30993–31015. [\[CrossRef\]](#)
22. Liang, F.; Zhang, Z.; Lu, H.; Leung, V.; Guo, Y.; Hu, X. Communication-efficient large-scale distributed deep learning: A comprehensive survey. *arXiv* **2024**, arXiv:2404.06114.
23. Goyal, P.; Dollár, P.; Girshick, R.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv* **2017**, arXiv:1706.02677.
24. McCandlish, S.; Kaplan, J.; Amodei, D.; OAID Team. An empirical model of large-batch training. *arXiv* **2018**, arXiv:1812.06162. [\[CrossRef\]](#)
25. Smith, S.L.; Kindermans, P.J.; Ying, C.; Le, Q.V. Don't decay the learning rate, increase the batch size. *arXiv* **2017**, arXiv:1711.00489.
26. You, Y.; Li, J.; Reddi, S.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv* **2019**, arXiv:1904.00962.
27. Settles, B. *Active Learning Literature Survey*; University of Wisconsin-Madison: Madison, WI, USA, 2009.
28. Gal, Y.; Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016.
29. Şahin, E.; Arslan, N.N.; Özdemir, D. Unlocking the black box: An in-depth review on interpretability, explainability, and reliability in deep learning. *Neural Comput. Appl.* **2025**, *37*, 859–965. [\[CrossRef\]](#)
30. Fort, S.; Hu, H.; Lakshminarayanan, B. Deep ensembles: A loss landscape perspective. *arXiv* **2019**, arXiv:1912.02757.
31. Ganaie, M.A.; Hu, M.; Malik, A.K.; Tanveer, M.; Suganthan, P.N. Ensemble deep learning: A review. *Eng. Appl. Artif. Intell.* **2022**, *115*, 105151. [\[CrossRef\]](#)
32. Khan, A.A.; Chaudhari, O.; Chandra, R. A review of ensemble learning and data augmentation models for class imbalanced problems: Combination, implementation and evaluation. *Expert Syst. Appl.* **2024**, *244*, 122778. [\[CrossRef\]](#)
33. Kulichenko, M.; Nebgen, B.; Lubbers, N.; Smith, J.S.; Barros, K.; Allen, A.E.A.; Habib, A.; Shinkle, E.; Fedik, N.; Li, Y.W.; et al. Data generation for machine learning interatomic potentials and beyond. *Chem. Rev.* **2024**, *124*, 13681–13714. [\[CrossRef\]](#)
34. Tang, S.; Yu, Y.; Wang, H.; Wang, G.; Chen, W.; Xu, Z.; Guo, S.; Gao, W. A survey on scheduling techniques in computing and network convergence. *IEEE Commun. Surv. Tutor.* **2023**, *26*, 160–195. [\[CrossRef\]](#)
35. Liang, F.; Zhang, Z.; Lu, H.; Li, C.; Leung, V.; Guo, Y.; Hu, X. Resource allocation and workload scheduling for large-scale distributed deep learning: A survey. *arXiv* **2024**, arXiv:2406.08115. [\[CrossRef\]](#)
36. Wu, B.; Zhong, Y.; Zhang, Z.; Liu, S.; Liu, F.; Sun, Y.; Huang, G.; Liu, X.; Jin, X. Fast distributed inference serving for large language models. *arXiv* **2023**, arXiv:2305.05920. [\[CrossRef\]](#)
37. Zhang, Y.K.; Zhan, D.C.; Ye, H.J. Capability Instruction Tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*; AAAI Press: Palo Alto, CA, USA, 2025; Volume 39.
38. Choudhury, A.; Wang, Y.; Pelkonen, T.; Srinivasan, K.; Jain, A.; Lin, S.; David, D.; Soleimanifard, S.; Chen, M.; Yadav, A.; et al. MAST: Global scheduling of ML training across Geo-Distributed datacenters at hyperscale. In Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24), Santa Clara, CA, USA, 10–12 July 2024.
39. Shen, L.; Sun, Y.; Yu, Z.; Ding, L.; Tian, X.; Tao, D. On efficient training of large-scale deep learning models. *ACM Comput. Surv.* **2024**, *57*, 1–36. [\[CrossRef\]](#)
40. Yang, G.; Hu, E.J.; Babuschkin, I.; Sidor, S.; Liu, X.; Farhi, D.; Lyon, N.; Hernandez, D.; Joshua, Z.; Gao, J.; et al. Tuning large neural networks via zero-shot hyperparameter transfer. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2021; Volume 34, pp. 17084–17097.
41. Balles, L.; Romero, J.; Hennig, P. Coupling Adaptive Batch Sizes with Learning Rates. *arXiv* **2016**, arXiv:1612.05086.
42. He, F.; Liu, T.; Tao, D. Control batch size and learning rate to generalize well: Theoretical and empirical evidence. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
43. Yu, G.; Tan, G.; Huang, H.; Zhang, Z.; Chen, P.; Natella, R.; Zheng, Z.; Lyu, M.R. A survey on failure analysis and fault injection in AI systems. *ACM Trans. Softw. Eng. Methodol.* **2026**, *35*, 1–42. [\[CrossRef\]](#)

44. Ovadia, Y.; Fertig, E.; Ren, J.; Nado, Z.; Sculley, D.; Nowozin, S.; Dillon, J.; Lakshminarayanan, B.; Snoek, J. Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift. In *Proceedings of the Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
45. Aru, J.; Labash, A.; Corcoll, O.; Vicente, R. Mind the gap: Challenges of deep learning approaches to theory of mind. *Artif. Intell. Rev.* **2023**, *56*, 9141–9156. [[CrossRef](#)]
46. Gal, Y.; Islam, R.; Ghahramani, Z. Deep Bayesian Active Learning with Image Data. In *Proceedings of the International Conference on Machine Learning (ICML)*; PMLR: Sydney, Australia, 2017; Volume 70, pp. 1183–1192.
47. Fakour, F.; Mosleh, A.; Ramezani, R. A structured review of literature on uncertainty in machine learning & deep learning. *arXiv* **2024**, arXiv:2406.00332. [[CrossRef](#)]
48. He, W.; Jiang, Z.; Xiao, T.; Xu, Z.; Li, Y. A survey on uncertainty quantification methods for deep learning. *ACM Comput. Surv.* **2025**, *58*, 179. [[CrossRef](#)]
49. Lakshminarayanan, B.; Pritzel, A.; Blundell, C. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
50. Gawlikowski, J.; Tassi, C.R.N.; Ali, M.; Lee, J.; Humt, M.; Feng, J.; Kruspe, A.; Triebel, R.; Jung, P.; Roscher, R.; et al. A Survey of Uncertainty in Deep Neural Networks. *Artif. Intell. Rev.* **2023**, *56*, 1513–1589. [[CrossRef](#)]
51. Sun, Y.; Li, X.; Li, L.; Feng, T.; Zhao, Y.; Yin, S. PHH-FL: Perceptual Hashing Hypernetwork Personalized Federated Learning for Heterogeneous Medical Image Analysis Tasks. *IEEE Internet Things J.* **2025**, *13*, 8712–8724. [[CrossRef](#)]
52. Li, X.; Li, L.; Li, M.; Yan, P.; Feng, T.; Luo, H.; Zhao, Y.; Yin, S. Knowledge distillation and teacher-student learning in medical imaging: Comprehensive overview, pivotal role, and future directions. *Med. Image Anal.* **2025**, *101*, 103819. [[CrossRef](#)]
53. Li, X.; Li, L.; Jiang, Y.; Wang, H.; Qiao, X.; Feng, T.; Luo, H.; Zhao, Y. Vision-Language Models in medical image analysis: From simple fusion to general large models. *Inf. Fusion* **2025**, *118*, 102995. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.