ALLPASS: ADAPTIVE, COST-EFFICIENT, SERVERLESS SCHEDULING FOR LLM FINE-TUNING ACROSS HETEROGENEOUS GPU CLOUDS

Yuqi Fu [1] **Ruizhe Shi** [2] **Tingfeng Lan** [1] **Zhaoyuan Su** [1] **Rui Yang** [1] **Haoliang Wang** [3] **Songqing Chen** [2] **Yue Cheng** [1]

ABSTRACT

Fine-tuning large language models (LLMs) has emerged as the dominant paradigm for adapting base models to specialized domains. Real-world fine-tuning workloads are highly skewed: a significant portion of jobs are short and exploratory, while long-running training jobs dominate GPU resources. Meanwhile, emerging GPU providers—spanning serverless, marketplace, and conventional clouds—offer complementary tradeoffs in startup time, price, and resource availability. To address workload skew and cloud heterogeneity, we present Allpass, an adaptive scheduler that unifies heterogeneous GPU clouds under a serverless abstraction. Allpass launches jobs on elastic serverless GPUs for instant startup and seamlessly migrates longer jobs to low-cost, marketplace, serverful GPUs, dynamically scaling resource allocation on both tiers to balance responsiveness and $ cost. We evaluate Allpass extensively using production workloads, and results show that Allpass significantly reduces average job completion time with lowest total $ cost across all baselines.

1 INTRODUCTION

Fine-tuning large language models (LLMs) has become a widely adopted approach for customizing general-purpose models to specific domains and tasks. This trend has led to the rise of commercial Fine-Tuning-as-a-Service (FTaaS) platforms such as Hugging Face AutoTrain (HuggingFace), OpenAI fine-tuning API (OpenAI, b), Google Vertex AI (Google), Amazon SageMaker (AWS, e), and Unsloth.ai (Unsloth.ai), which abstract away infrastructure and provide turnkey solutions for model customization.

Despite recent advances in parameter-efficient fine-tuning (PEFT) methods such as LoRA (Hu et al., 2022) and QLoRA (Dettmers et al., 2023), fine-tuning billion-scale LLMs remains computationally expensive. Datacenter-grade GPUs like NVIDIA A100 and H100 are often required to accommodate model size and training throughput, yet these accelerators are both scarce—due to global supply limits (Strati et al., 2024; JIANG et al., 2025; Yang et al., 2023a)—and expensive. For instance, on-demand pricing for H100 GPUs can reach from $6.89 to $11.06 per hour on major cloud providers such as AWS (AWS, a), GCP (GCP, a), and Azure (Azure, a) (Table 2). Such prices accumulate quickly across large jobs and long training runs, creating a significant barrier for small teams and independent developers (Cottier et al., 2024).

In response, a new wave of decentralized, marketplace GPU providers has emerged. Marketplace platforms such as Vast.ai (Vast.ai) aggregate idle accelerators from individuals and smaller datacenters, offering significantly reduced prices but with resource availability volatility. In parallel, serverless GPU platforms like Modal (Modal, a) and RunPod (Runpod, 2025) offer second-level startup times and high GPU elasticity, enabling interactive fine-tuning at the cost of higher per-second rates. These heterogeneous alternatives offer flexibility and cost-efficiency not achievable with conventional cloud providers.

However, fine-tuning workloads exhibits distinctive scheduling challenges. Analysis of real-world LLM development and fine-tuning workloads (Hu et al., 2024) reveals that over 83% of jobs finish within 10 minutes—terminated early due to exploratory use, interactive debugging, or bugs/failures—while others are long-running production jobs that span hours. This long-tailed job duration distribution leads to GPU underutilization (Weng et al., 2022; Carver et al., 2025; Coppock et al., 2025) in conventional VMs and highlights the need for low-latency feedback for short jobs. Meanwhile, in current conventional cloud paradigms, users reserve entire GPUs for the duration of their sessions, regardless of whether the device is actively used. Such a coarse-grained reservation policy leaves GPU idle between training phases, further inflating costs.

[1]University of Virginia [2]George Mason University [3]Adobe Research. Correspondence to: Yue Cheng <mrz7dp@virginia.edu>.

Proceedings of the 8^{th} MLSys Conference, Santa Clara, CA, USA, 2025.

These workload patterns present a unique challenge: *How to reconcile the responsiveness needs of short, exploratory jobs with the cost-efficiency requirements of long-running, GPU-intensive jobs.* This tension is exacerbated by three factors: (1) the high cost of serverless GPUs for sustained training, (2) the contention between short and long jobs, and (3) the unstable resource availability of cheaper marketplace GPUs. Balancing these tradeoffs requires a novel scheduling approach that delivers low-latency execution for short jobs while shifting longer workloads to more cost-effective resources—without sacrificing performance.

To address these challenges, we present Allpass[1], a novel GPU cluster scheduler that unifies heterogeneous GPU clouds under a *serverless abstraction*. Allpass exposes a serverless GPU interface that allows users to submit LLM fine-tuning jobs without needing to manage or configure the underlying hybrid cloud infrastructure. Internally, Allpass dynamically orchestrates jobs across two tiers: a serverless GPU tier (Modal (Modal, a)) for immediate startup, and a serverful GPU tier that seamlessly integrates cheap marketplace clouds (Vast.ai (Vast.ai) for cheap execution) and conventional clouds (Lambda (Lambda) as a stable fallback for resource availability). Each job initially executes in the serverless tier to ensure responsiveness; if it exceeds an execution time threshold, it is checkpointed and restored to cheaper resources at the serverful tier. Allpass leverages soft deadlines—either user-specified or inferred from job metadata—for scheduling decisions. Job dispatch follows an earliest-deadline-first scheduling policy guided by the Demand Bound Function framework (Baruah et al., 1990). Allpass employs two feedback-driven scalers to adjust serverless execution time threshold and serverful GPU cluster capacity in real-time.

This paper makes the following contributions:

- We conduct a comprehensive characterization study analyzing multiple production workloads and representative GPU clouds. Our study reveals novel challenges and opportunities for fine-tuning workload scheduling.
- We design Allpass, the ***first*** GPU scheduler that exposes a *serverless GPU interface* through a real-world serverless GPU platform. Internally, Allpass orchestrates jobs across a low-latency, elastic serverless GPU tier and a cost-efficient serverful GPU tier spanning marketplace and conventional GPU clouds.
- We implement an Allpass prototype[2] atop real-world commercial serverless and marketplace GPU clouds.
- We evaluate Allpass using production LLM fine-tuning traces. Results show Allpass reduces average job completion time by up to 70% with 66% total $ cost savings.

[1]Adaptive, Low-Latency, Price-Aware Serverless Scheduler.

[2]Source code and collected Wandb, ABC, and Vast.ai trace datasets will be released upon paper acceptance.

2 Background and Motivation

2.1 Overview of LLM Fine-Tuning Platforms

Fine-Tuning-as-a-Service (FTaaS) platforms have become the primary means for adapting LLMs to domain-specific tasks. Platforms like Unsloth (Unsloth.ai), Hugging Face AutoTrain (HuggingFace), and OpenAI fine-tuning API (OpenAI, b) simplify the distributed-training complexity by unified training APIs.

A typical fine-tuning pipeline comprises two stages: (1) **Data ingestion and preprocessing.** Users upload datasets in structured formats such as JSONL or CSV. The platform tokenizes and formulates text, generates prompt-response pairs, and stores processed data in cloud object storage for reuse. To reduce repeated I/O overhead during subsequent training runs, modern FTaaS platforms employ caching mechanisms that colocate frequently accessed datasets near compute resources. For example, cloud providers often mount network-attached volumes (e.g., Amazon EBS (AWS, b), Modal persistent storage (Modal, c)) or ephemeral NVMe caches on GPU nodes. This caching layer minimizes cold-start delays when reusing preprocessed data across multiple training jobs or parameter configurations, improving startup latency and data throughput.

(2) **GPU job scheduling and execution.** After the data preprocessing stage, preprocessed data are streamed to reserved GPU VMs or containers. Jobs usually run on long-lived cloud instances billed per second or minute and the GPU execution dominates cost and latency, especially for billion-parameter models that require datacenter-grade GPUs such as A100 and H100. To achieve more cost-effective fine-tuning, training often uses parameter-efficient methods such as LoRA (Hu et al., 2022), QLoRA (Dettmers et al., 2023), or adapters (Pfeiffer et al., 2021), which apply low-rank updates to a frozen base model (Han et al., 2024). These techniques greatly reduce memory usage, enabling large-scale model tuning on limited hardware and with reduced budget requirements.

While recent research has substantially optimized training data preprocessing, the cluster GPU scheduling and compute phase remains the dominant source of cost and performance (Graur et al., 2024; Audibert et al., 2023). This phase consumes the majority of runtime and infrastructure spent, particularly for large-scale fine-tuning tasks on multi-billion parameter models. Moreover, access to datacenter-grade GPUs (e.g., A100 and H100 listed in Table 2) continues to be constrained due to global supply limits (Strati et al., 2024; JIANG et al., 2025; Yang et al., 2023a) and surging AI demand (AWS, c).

From the perspective of both service providers and users, optimizing this stage is critical. For platform providers, inefficient scheduling leads to GPU underutilization, inflated

Table 1: Statistics of GPU training workload traces.

Trace	# Users	# Jobs	Job Runtime (avg/P50/P90)	Covered Period
Wandb	13	6,021	630s/32s/2025s	10/20-06/25 (57 months)
Seren	131	818,326	1,208s/22s/765s	03/23-08/23 (6 months)
Kalos	12	62,413	482s/28s/260s	05/23-08/23 (3 months)
ABC	3,647	547,390	1,020s/120s/268s	06/21-08/21 (3 months)

costs, and violated service-level objectives (SLOs). For end users, prolonged startup delays and training time and high per-job cost inhibit rapid iteration. This paper focuses on the stage of FTaaS GPU scheduling as the core challenge.

2.2 Characterizing GPU Job Duration

We characterize training workloads using four representative production workload traces: Wandb (Wandb, a;b), Seren / Kalos (Hu et al., 2024), and ABC. Table 1 lists their basic statistics. Wandb is a fine-tuning trace dataset that we collected from Wandb-hosted projects. Specifically, we collected the Wandb trace from the Wandb public platform (Wandb, a; CoreWeave), where individual users publicly share projects, dashboards, and experiments. Each project corresponds to a fine-tuning job created by a user, and multiple repeated runs under different configurations. We aggregated these project-level and run-level metadata through the Wandb public API, preserving key attributes such as job submission time and durations. All user identifiers and textual content were anonymized before analysis.

Seren and Kalos are published LLM development and training traces (Hu et al., 2024), spanning a total of 4,704 A100 GPUs. The traces include scheduler logs and execution metadata of fine-tuning workloads collected over a six-month period from March to August 2023. The Seren cluster records 818K jobs from 131 users, while Kalos contains 62K jobs from 12 users.

ABC is a large-scale production Jupyter Notebook workload trace that we collected from ABC (platform name anonymized), an industry AI service platform. The ABC trace includes 547K *interactive* deep neural network training jobs collected over a three-month period from June to August 2021. The workload spans over 2.5K EC2 GPU VMs and over 166 million GPU hours. The sample granularity of this trace is 1 minute. We analyze the ABC trace to reflect emerging fine-tuning trends, which are increasingly interactive (Wandb, a) and predominantly conducted through computational notebooks (Unsloth.ai; Colab, b;a).

Figure 1 plots the distribution of training job duration for all four workloads. The distributions show several interesting, common trends: (1) Fine-tuning workloads are highly skewed toward short, transient jobs—more than 80% of runs complete within only 10 minutes for all four workloads, and in the WandB trace, more than 40% of jobs start training but terminate almost immediately after initialization. (2) All traces exhibit heavy-tailed distributions, where the top 10% longest training jobs contribute to over 84% of total GPU time. In the Seren LLM development trace, for example,

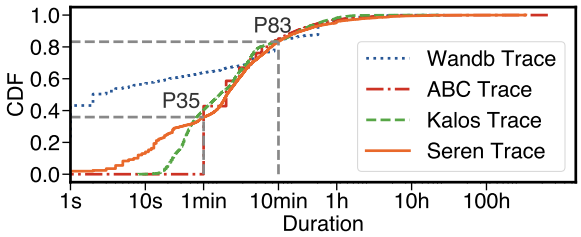


Figure 1: CDF distribution of GPU training job duration.

we observe two distinct job types: approximately 83% jobs complete within 10 minutes—often due to early termination, interactive debugging, or exploration runs—while the remaining 17% are long-running training jobs.

Implications

- *Most enterprise fine-tuning platforms continue to rely on VM-based or reservation-based provisioning, where each user session monopolizes its assigned GPUs until the session ends. Once allocated, the GPUs remain dedicated—even during idle or paused phases—because current systems lack mechanisms to share, reclaim, or reassign resources reserved by a user (Cao et al., 2025; Carver et al., 2025).*
- *The bimodal nature of real-world LLM fine-tuning workloads introduces a fundamental scheduling challenge for FTaaS platforms: short jobs demand ephemeral, low-latency access to GPU resources, whereas long-running jobs dominate total GPU consumption and drive cost. Efficiently serving both classes of jobs demands a scheduling strategy that can balance responsiveness and cost-efficiency under a shared infrastructure.*

2.3 Characterizing GPU Cloud Heterogeneity

Recent years have witnessed a rapid proliferation of commercial GPU providers beyond traditional public clouds such as AWS, Azure, and Google Cloud. To better understand the cloud heterogeneity, we classify cloud GPU providers into three categories—*conventional, serverless, and marketplace*—each representing a distinct resource provisioning and pricing tradeoff. Conventional clouds resemble legacy Infrastructure-as-a-Service (IaaS) offerings with virtually infinite capacity and stable pricing. In contrast, serverless and marketplace platforms introduce new tradeoffs in resource availability, elasticity, and cost efficiency. In this section, we characterize the GPU cloud heterogeneity in pricing and provisioning behaviors.

Serverless GPU clouds, such as Modal (Modal, a), represent an emerging class of "GPU-as-a-Service" platforms that abstract server provisioning and lifecycle management away from users and expose a cloud-function-like programming interface similar to Function-as-a-Service (FaaS) (AWS, d). Unlike conventional clouds that bill users with a minimum of 60 seconds of VM usage, Modal charges on a per-second basis (Modal, b), making it more cost-efficient

Table 2: GPU price ($ per hour) for representative cloud providers. For Vast.ai (Vast.ai) and DataCrunch.io (datacrunch.io), which employ market-driven pricing, values denote the *median (min, max)* prices observed across all registered GPU providers from September 26, 2025 to October 25, 2025. AWS, GCP, and Azure offer spot instances. The values denote the *on-demand (spot min, spot max)* prices observed between October 30, 2024, and October 29, 2025. On-demand prices are recorded as of October 25, 2025. Prices highlighted in dark green indicate the lowest observed (min) price across platforms, while those in dark red represent the highest. "–": unavailability.

GPU	Serverless Clouds		Marketplace Clouds		Conventional Clouds			
	Modal	Runpod	**Vast.ai**	DataCrunch.io	**Lambda**	AWS	GCP	Azure
Consumer Tier								
RTX4090	–	**0.34**	**0.42 (0.17, 0.92)**	–	–	–	–	–
Datacenter Inference Tier								
L4	0.8	0.44	**0.38 (0.38, 0.39)**	–	–	0.81 (0.39, 0.66)	**1.10 (0.27, 0.28)**	–
L40	**1.95**	0.69	**0.68 (0.60, 0.72)**	0.91 (0.58, 0.97)	–	1.86 (0.67, 1.65)	–	–
T4	**0.59**	–	**0.14 (0.14, 0.15)**	–	–	0.53 (0.26, 0.41)	0.38 (0.14)	0.52 (0.12, 0.28)
Datacenter Training Tier								
V100	–	–	0.21 (0.11, 0.26)	**0.13 (0.11, 0.14)**	0.55	1.53 (0.35, 0.98)	1.77 (0.99)	**2.75 (0.49, 0.61)**
A10	**1.10**	–	**0.26 (0.25, 0.26)**	–	0.75	1.01 (0.44, 0.52)	–	0.91 (0.37, 0.48)
A100	2.10	1.39	**1.08 (0.38, 1.66)**	0.71 (0.64, 0.86)	1.29	2.74 (0.32, 2.04)	3.67 (0.47, 1.46)	**3.67 (0.61, 1.08)**
H100	3.96	2.69	**2.05 (1.53, 2.62)**	1.99 (1.83, 2.12)	2.49	6.88 (1.65, 6.88)	**11.06 (1.52, 4.39)**	6.98 (4.29, 8.38)

Table 3: Startup time (avg ± std of 20 tests) on A100 instances across three representative types of GPU cloud providers.

Platform	Modal	Vast.ai	Lambda
Startup time (s)	3.88 ± 1.84	36.04 ± 7.35	255.59 ± 26.26

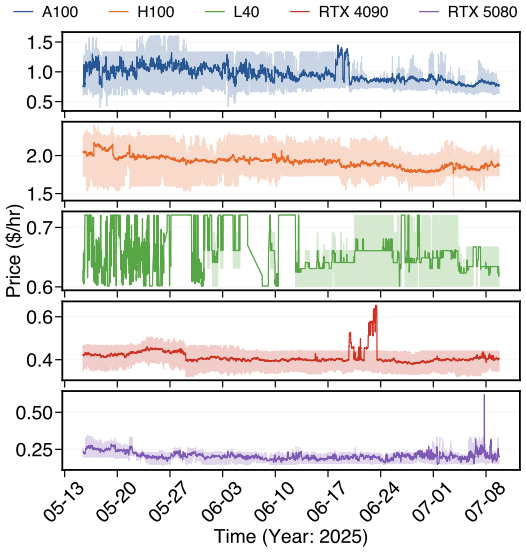


Figure 2: Vast.ai's GPU price dynamics. Curve shows the average price across all available registered GPU providers at a given time; shaded region represents the P25-P75 price range among all.

for short, interactive AI workloads. As shown in Table 2, Modal's on-demand GPU price is substantially higher than that of marketplace or conventional clouds—often exceeding 2-4× for datacenter-grade GPUs. This premium cost reflects its optimization for interactive, low-latency workloads that demand near-instant provisioning. Empirical measurement on `bert-large-uncased` (Bert) fine-tuning (Table 3) shows that Modal achieves an average startup latency of only 3.88s, compared to 36.04s on Vast.ai and 255.59s on Lambda (Lambda). These results indicate that *serverless GPU providers trade higher unit cost for dramatically improved resource elasticity and responsiveness.*

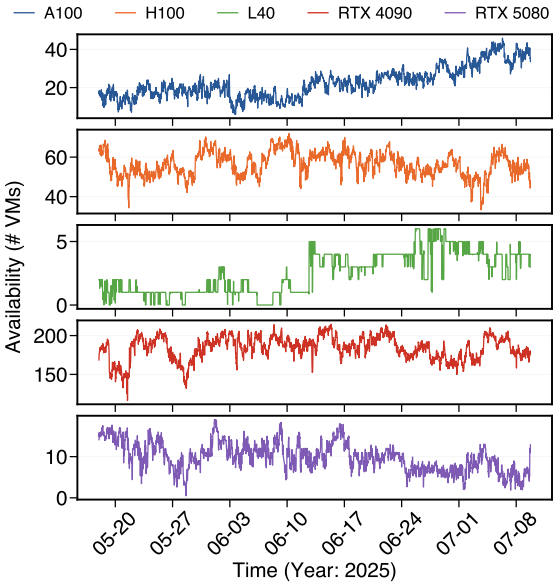


Figure 3: Vast.ai's GPU availability dynamics. Each curve represents the number of available VM instances for a specific GPU type over a seven-week period.

Marketplace GPU clouds introduce a *decentralized model* for user-driven GPU provisioning, where individuals and small datacenters can register as GPU providers by leasing idle resources through *open trading* platforms. Their main appeal lies in cost efficiency: anyone can register as a provider and list available GPUs with custom pricing. To attract demand, providers often compete by lowering prices, effectively forming a dynamic supply-demand marketplace. As a result, these platforms offer mainstream datacenter training GPUs such as A100 and H100 at only 14-22% of AWS's price—based on Vast.ai's minimum observed prices for these GPU types (Table 2).

However, this flexibility comes at the *expense of price stability*. Figure 2 illustrates the temporal dynamics of GPU pricing on Vast.ai over a two-month period. A100 and L40 prices average around $1.0 and $0.6 per GPU hour,

respectively, but fluctuate substantially over time driven by changing supply and demand conditions. By contrast, RTX 4090 and RTX 5080 GPUs remain relatively stable around $0.4 and $0.3 per GPU hour. Such volatility exposes a clear *cost-stability tradeoff*: *users can exploit lower-priced offers to achieve up to 5-8× savings, but the volatility of marketplace prices complicates resource planning and sustained fine-tuning at scale.*

Unlike conventional GPU clouds that offer virtually unlimited resources, marketplace GPU providers exhibit highly dynamic resource availability driven by user supply and demand. Figure 3 shows the time-varying GPU inventory on Vast.ai over a seven-week period. Consumer GPUs such as RTX 4090 and RTX 5080 remain relatively abundant, sustaining over 200 available GPUs throughout the trace. In contrast, datacenter-grade tiers offer tighter supply: H100 and A100 inventories fluctuate around 40-60, while L40 availability remains scarce—nearly depleted during the first three weeks of the trace. Consequently, *while marketplaces provide a large pool of inexpensive consumer GPUs, their limited and volatile supply of datacenter-grade GPUs makes sustained LLM fine-tuning unreliable without overprovisioning or migration support.*

Another notable category of GPU resources comes from spot instances (Table 2), which offer substantially lower prices compared to on-demand instances. However, spot instances (AWS, g; GCP, b; Azure, c) can be preempted by the provider at any time, with only a short grace period (e.g., 30s on GCP Spot VMs (GCP, b)). In contrast, marketplace platforms such as Vast.ai allow users to define a maximum lease duration, while users can rent GPUs for flexible periods within that limit, without risk of preemption.

Implications

- *Serverless GPU clouds provide near-instant startup and highly elastic scaling, making them ideal for short-lived or interactive fine-tuning jobs commonly seen in real-world workloads.*
- *Marketplace GPU clouds offer significant price advantages—up to 5-8× cheaper than major public clouds—but their resource availability is highly volatile. To support real-world fine-tuning workloads with highly variable job durations, a hybrid scheduling strategy is needed: utilizing marketplace GPUs for low-cost execution, serverless GPUs for rapid responsiveness, and conventional clouds as stable fallbacks.*

3 DESIGN

3.1 Design Overview

Modern GPU clouds have become increasingly heterogeneous, each offering distinct tradeoffs among startup latency, cost efficiency, and resource availability (inventory stability). This heterogeneity motivates the need for a heterogeneity-aware scheduler that can combine the complementary strengths of these GPU providers to balance latency, cost, and resource availability.

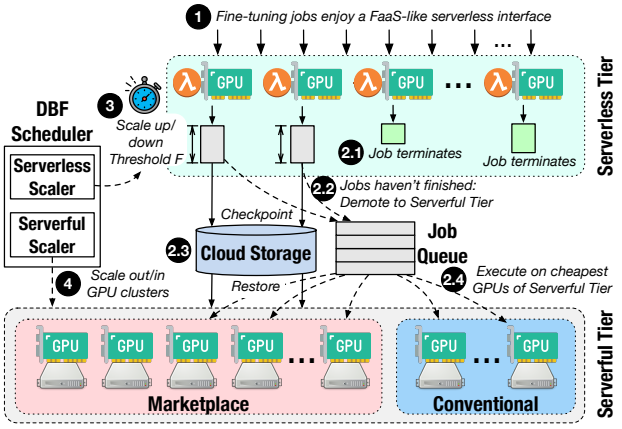


Figure 4: Allpass architecture overview.

To this end, we design Allpass, an online, multi-tier GPU scheduler that orchestrates LLM fine-tuning workloads across three types of GPU clouds—serverless, marketplace, and conventional—to jointly optimize responsiveness and cost efficiency. As illustrated in Figure 4, Allpass manages two tiers of resource pools: (1) **Serverless GPU Tier:** A dynamic fleet of *serverless GPU workers*, which enables immediate job execution upon arrival, allowing short jobs to complete entirely within the serverless tier; and (2) **Serverful GPU Tier:** A *job queue* that manages long-running jobs migrating out of the serverless tier. It dispatches these jobs across a shared cluster of GPU workers spanning both marketplace and conventional cloud providers. Jobs in this tier are ordered and scheduled using an Earliest Deadline First (EDF) algorithm (Liu & Layland, 1973).

Serverless interface and soft deadlines. Each job is submitted as a serverless GPU function (AWS, d; Modal, a) and is associated with a soft relative deadline that indicates the user's expected completion time. By exposing a *direct-user-facing serverless GPU interface*, Allpass offers an easy-to-use and fully managed FTaaS platform that abstracts away the complexities of underlying hybrid cloud infrastructure. User-submitted GPU functions comprise job code (in Python), structured metadata such as model name, parameter count, dataset size, number of training steps, and target GPU type, along with an expected completion target (Wu et al., 2024). This expected completion target is recorded as a soft deadline[3] visible to Allpass and directly

[3]Many commercial AI platforms (e.g., OpenAI (OpenAI, a), AWS SageMaker (AWS, f), and Azure ML (Azure, b)) support training time limits or stopping conditions, but do not utilize user-specified deadlines for scheduling. In contrast, orchestrators like Kubernetes support job-level deadlines via `activeDeadlineSeconds` (Kubernetes, b). Recent research (Wu et al., 2024) demonstrates the effectiveness of deadline-aware GPU scheduling.

used by the GPU scheduler to determine job ordering. If no explicit target is given, Allpass automatically infers a soft deadline proportional to the predicted runtime, estimated from metadata and historical profiling traces. Under soft deadline constraint, the job queue is maintained in ascending order of deadlines, and available workers always select the job with the earliest deadline.

Allpass workflow. Allpass orchestrates fine-tuning jobs across serverless and serverful GPU tiers through the following steps. ❶ Users submit function-like fine-tuning jobs. Upon submission, the job is immediately placed to the serverless GPU tier, enabling near-instant startup. The *serverless scaler* assigns each job a configurable *execution time threshold*, during which the job executes speculatively on serverless GPUs. ❷.❶ If the job finishes within the threshold, it completes entirely in the serverless tier with minimal latency. ❷.❷ For longer-running jobs, partial progress is checkpointed to remote storage, and the remaining portion is enqueued into the serverful job queue. ❷.❸–❷.❹ Jobs in queue are scheduled and restored on available GPU workers across the marketplace and conventional tiers, continuing from their last checkpoint until completion. ❸–❹ Allpass uses a control plane to govern job orchestration across tiers. At its core is a *GPU resource scheduler* (§3.2), which estimates aggregate workload demand within a scheduling window using policies inspired by the Demand Bound Function (DBF) framework (Baruah et al., 1990).

At its core, the DBF scheduler continuously answers the following question: *Given all jobs and their deadlines, how much total GPU work must be completed within each upcoming scheduling window T to satisfy all deadlines?* To do so, it aggregates every unfinished job whose deadline falls within the upcoming scheduling window T and computes, for each time window T the total GPU hours required by that time. For instance, if two fine-tuning jobs together require 3 GPU hours to meet their deadlines within the next hour, then $D(1\text{h}) = 3$.

3.2 Main Scheduling Loop

The DBF scheduler serves as the analytical core of Allpass, providing a global view of system pressure and coordinating the resource scaling at both serverless and serverful tiers. Intuitively, DBF acts as a single "pressure gauge" that continuously monitors the GPU resource usage of the *serverful GPU tier* and measures whether it can finish all jobs before their respective deadlines under current capacity.

Core logic. Let $D(t, T)$ denote the total remaining execution demand of all jobs whose deadlines fall within the next scheduling window T:

$$D(t, T) = \sum_{j \in J(T)} r_j(t), \tag{1}$$

where $r_j(t)$ is the remaining runtime of job j at time t, and $J(T)$ is the set of such jobs whose respective deadline fall into the sliding window. As mentioned in §3.1, each job provides its expected fine-tuning step count through the job template that is visible to the system. Since the per-step duration remains stable, making the remaining time $r_j(t)$ predictable from scheduling progress (Wu et al., 2024). The system capacity over the same time window is $C_t \cdot T$, where C_t is the aggregate service rate (per time unit) of all active GPU workers in the serverful tier. Allpass defines a normalized *pressure ratio* as follows:

$$p(t) = \max_{T \in \mathcal{T}} \frac{D(t, T)}{C_t \cdot T}. \tag{2}$$

A value of $p > 1$ indicates overload in term of deadline requirement; $p < 1$ implies available margin. DBF monitoring thus continuously assesses *schedulability* (Baruah et al., 1990; Liu & Layland, 1973) under the soft-deadline constraint. We define "schedulability" in Allpass as follows: A fine-tuning workload is schedulable if, within any near-term scheduling time window T, the cumulative remaining execution demand of all jobs expected to finish by $t + T$ does not exceed the total compute service that the serverful tier can deliver within the same period, i.e., $D(t, T) \leq C_t \cdot T$.

Coordinated spatial-temporal GPU scaling. Whenever the pressure signal p changes, Allpass simultaneously triggers two complementary scaling actions along orthogonal dimensions of control—*temporal* and *spatial*:

- The **serverless scaler** (*temporal actuator*) adjusts the execution threshold F on the serverless tier, controlling how long each job remains in the fast path before demoting to the serverful tier. This temporal scaling reacts immediately to short-term fluctuations in demand.
- The **serverful scaler** (*spatial actuator*) adjusts the size of serverful GPU pool across marketplace and conventional clouds, thereby optimizing the aggregate compute capacity C_t. This spatial scaling is slower, expanding or shrinking the GPU VM cluster as needed.

When $p > 1$, both controllers react together: the serverless scaler increases F to quickly absorb more (near-term) demand on serverless GPUs, while the serverful scaler provisions additional workers slowly to restore capacity balance. When $p < 1$, both reduce their respective resource footprints to save cost: the serverless scaler shortens F, and scaling gradually releases excess GPUs. This coordinated spatial-temporal control loop enables Allpass to adapt stably and efficiently across heterogeneous GPU tiers.

3.3 Serverless and Serverful Scaling Policies

The DBF signal provides the global coordination target, while the two-tier scalers translate that signal into concrete resource adjustments on different timescales.

Serverless scaler. Each job begins on the serverless tier with an optimized execution time budget F, representing how long it may run before demoting to the serverful queue.

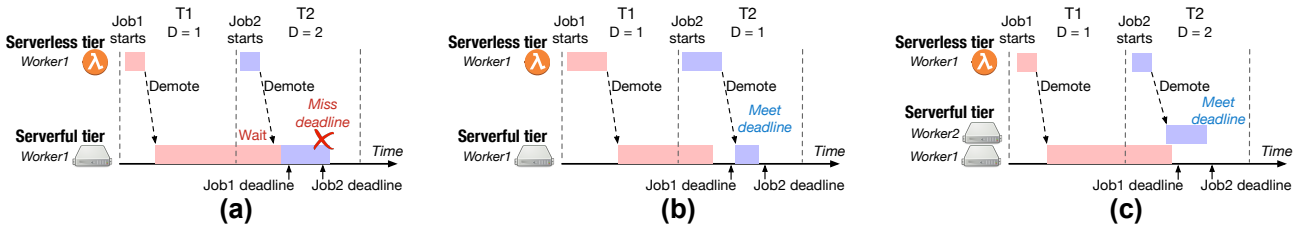


Figure 5: Example Gantt chart of Allpass's DBF schedules. In **(a)**, Job1 arrives during period T_1, where the DBF value equals 1. When Job2 arrives at T_2, the DBF value increases to 2, exceeding the capacity of the single available worker and causing a schedulability violation. To restore feasibility, Allpass applies two action in parallel: extending the execution threshold on serverless GPUs **(b)**, and scaling out additional GPU workers on serverful tier **(c)**.

Algorithm 1 Serverless Scaler

Setup: Sliding window $\mathcal{T}$, limits $G_{\text{up}}, G_{\text{dn}}, R_{\text{up}}, R_{\text{dn}}$.
State: Serverless threshold F; last-change time $\tau \leftarrow -\infty$.

1: **while** true **do** ▷ periodic control loop
2: $p \leftarrow \max_{T \in \mathcal{T}} \frac{D(t,T)}{C_t T}$ ▷ DBF pressure, Eq. (2)
3: **if** $p > 1$ **then** ▷ overload ⇒ increase F
4: $\Delta F \leftarrow \min\big(R_{\text{up}},\, G_{\text{up}}(p-1)\big)$
5: **else** ▷ underload ⇒ decrease F
6: $\Delta F \leftarrow -\min\big(R_{\text{dn}},\, G_{\text{dn}}(1-p)\big)$
7: $F \leftarrow F + \Delta F$; $\tau \leftarrow$ now ▷ commit update
8: *NextTick:* **sleep** TimeTick until next iteration
9: $\mathcal{T} \leftarrow \{T \in \mathcal{T} \mid t \geq T_{start}\}$ ▷ advance sliding window

Output: Updated per-job serverless tier threshold F.

In Algorithm 1, the serverless scaler operates as a *while loop* (line 1), re-evaluating the schedulability at each time tick (line 8). The loop maintains a sliding window $\mathcal{T}$, as defined in Equation (1), to estimate the pressure ratio p, computed as the maximum demand-to-capacity ratio across all time units in the current sliding window (line 2). This pressure ratio serves as the control signal to adjust the threshold F. The adjustment behavior is governed by four parameters: $G_{\text{up}}, G_{\text{dn}}$ (scaling aggressiveness), and $R_{\text{up}}, R_{\text{dn}}$ (rate limits). When $p > 1$ (overload), the scaler increases the threshold using $\Delta F = \min(R_{\text{up}}, G_{\text{up}}(p-1))$ (lines 3–4). When $p < 1$ (underload), it decreases the threshold similarly (lines 5–6). To ensure responsiveness, we configure the scaling-up parameters higher than the scaling-down ones, allowing fast reaction to deadline pressure while avoiding oscillation. After each loop iteration, the serverless threshold F is updated and the sliding window $\mathcal{T}$ is advanced to discard expired intervals (lines 7–9).

Serverful scaler. In parallel, the serverless scaler propagates the global DBF pressure to the serverful scaler, which adjusts the number of serverful-tier workers, as shown in Algorithm 2. Let I denote the current number of active serverful workers, and μ the service rate of a single worker. Given the predicted demand $D(t,T)$ and total capacity $C = I\mu$ over the next scheduling window T, the shortfall in capacity is computed as $\Delta(t) = \max 0, D(t,T) - C$ (line 2). The scaler estimates the capacity contributed by each worker over the window as μT, and determines the number of workers to adjust using $k^\star = \left\lceil \frac{\Delta(t)}{\mu T} \right\rceil$. When $p > 1$ (overload),

Algorithm 2 Serverful Scaler

Setup: Next sliding window T; serverful worker rate μ; current serverful worker count I; global pressure p.

1: **while** True **do**
2: $C \leftarrow I\,\mu$; $\Delta(t) \leftarrow \max\{0, D(t,T) - C\}$
3: **Scale-out trigger:**
4: **if** $p > 1$ **then** ▷ overload ⇒ add workers
5: $k_\uparrow \leftarrow \left\lceil \frac{\Delta(T)}{\mu T} \right\rceil$
6: add $k_\uparrow$ workers; $I \leftarrow I + k_\uparrow$
7: **Scale-in trigger:**
8: **if** $p < 1$ **then**
9: $k \leftarrow \left\lfloor \frac{\Delta(t)}{\mu T} \right\rfloor$
10: remove $k_\downarrow$ workers; $I \leftarrow I - k_\downarrow$
11: *NextTick:* **sleep** TimeTick until next iteration

Output: Updated worker count I.

the scaler provisions $k^\uparrow$ additional workers immediately (lines 4–6). Conversely, when $p < 1$ and the system is underutilized, it scales in by releasing $k^\downarrow$ workers (lines 8–10). The loop sleeps until the next scheduling window (line 11) before repeating.

The serverful scaler adopts a greedy policy when making scaling decisions. During scale-out, it provisions the cheapest available workers (across marketplace and conventional clouds) to fill the capacity gap, whereas during scale-in, it releases the most expensive, idle workers. When marketplace instances are unavailable, the scaler provisions conventional cloud workers as a fallback.

3.4 Putting It all Together: End-to-end Example

Figure 5 illustrates how Allpass's DBF scheduler maintains schedulability as workload intensity varies. At time window T_1, the workload is balanced with the available serverful GPU capacity ($p \approx 1$): Job1 is demoted to the serverful tier and completes execution within its deadline. However, in the next window (T_2), a burst in arrivals increases the aggregate demand to $D = 2$, exceeding the capacity of the single GPU worker (Figure 5(a)). When Job2 is demoted from the serverless tier, Worker1 is still busy executing Job1. This forces Job2 to wait in the queue, and the resulting delay causes it to miss its deadline.

To handle this, Allpass performs two actions in parallel:

1. The serverless scaler extends the execution threshold on serverless tier, allowing Job2 to make additional progress before demotion. By the time Job2 is handed off to the next tier during T_2, Worker1 has completed Job1 and can immediately admit Job2, eliminating queueing delay and avoiding a deadline miss (Figure 5(b)). This shifts transient load to the elastic serverless tier, alleviating pressure on the constrained marketplace serverful pool.
2. Upon demotion, Allpass restores the snapshot (Figure 4) while its serverful scaler provisions an additional GPU worker in serverful tier (Figure 5(c)). Once Worker2 is ready and the restore process finishes, Job2 is scheduled directly onto Worker2, further reducing contention. As this new capacity becomes available, the demand value of D gradually returns toward 1, restoring schedulability.

In summary, DBF acts as the feedback signal of the closed-loop system: when demand pressure rises, Allpass reacts by offloading near-term work to elastic serverless GPU tier and scaling out the compute capacity of serverful GPU tier; when pressure subsides, it conservatively shortens the serverless window and releases serverful tier's idle GPUs.

4 Implementation

We implemented Allpass as a GPU resource orchestrator accompanied by a lightweight web-frontend serverless GPU interface for user interaction. Allpass' scheduler has 4,800 lines of Go code and exposes REST endpoints that interface with backend APIs of Modal, Vast.ai, and Lambda. Allpass' frontend enables users to write fine-tuning jobs as Modal-compatible GPU functions (Modal, a) that specify fine-tuning environments and soft deadlines for each job. Upon submission, it serializes the job metadata into JSON and transmits it to the scheduler backend through a RESTful API. Through these APIs, Allpass provisions and releases GPU workers, transfers checkpoints via an S3-compatible object storage, and monitors runtime metrics. Each control plane module—the DBF scheduler, serverless scaler, and serverful scaler—runs as a goroutine that periodically polls workload statistics and updates shared scheduler state.

5 Evaluation

5.1 Experimental Methodology

Fine-tuning benchmarks. We built a benchmark suite that consists of two types of fine-tuning jobs: (1) LLaMA series including `LLaMA3-8B` (Dubey et al., 2024) and `LLaMA2-13B` (Touvron et al., 2023), and (2) `distilbert-base-uncased` (BERT) (Bert). For LLaMA models, fine-tuning jobs are configured using the Hugging Face PEFT framework with LoRA adapters for parameter-efficient adaptation. The implementation leverages Transformers with bitsandbytes for 4-bit quantization, using the following LoRA settings: r=16, α=32, and $dropout$=0.05. The BERT model is fine-tuned on the IMDB sentiment classification dataset (Maas et al., 2011), serving as a lightweight workload that enables precise control over job duration for short-lived tasks. In contrast, the LLaMA models are fine-tuned on instruction-tuning datasets such as Alpaca (Taori et al., 2023) and OpenOrca (Lian et al., 2023), representing long-running jobs. By varying the number of training steps, the benchmark spans a range of execution times, capturing diverse scheduling scenarios.

Workload generation. We generated fine-tuning workloads based on the Seren (Hu et al., 2024) cluster trace. To adapt the trace on our evaluation setup, we sampled 200 jobs through the trace and downscale both the job duration and inter-arrival time (IAT). Specifically, the job duration is reduced to one-fifth of its original value, and the IATs are shortened to one-fifteenth, making each test run tractable while preserving real-world workload dynamics.

Table 4: Benchmark model statistics.

Model	Per-epoch time	# epochs (avg)
`LLaMA3-8B`	476s	16.1
`LLaMA2-13B`	765s	15.8
`BERT`	20s	45.6

Each fine-tuning job corresponds to one of the three benchmark models. We pre-profile per-epoch execution times for each model—approximately 20 seconds for `BERT`, 476 seconds for `LLaMA3-8B`, and 765 seconds for `LLaMA2-13B`—as summarized in Table 4. The average number of epochs per job is 45, 16, and 15, respectively. We assume users specify a desired deadline at job submission, which is visible to Allpass. For evaluation, we generate soft deadlines proportional to each job's estimated execution time, where the deadline is randomly chosen to be between $1\times$ and $10\times$ the estimated job duration. This design reflects realistic user expectations: interactive debugging or early-terminated jobs demand quick turnaround, while long-running jobs can tolerate longer completion windows.

Baselines. We tested two deployment scenarios. (1) In the *limited capacity setup*, the serverful tier is limited by 30 concurrent A100 instances, representing a realistic execution environment where marketplace GPU availability may be scarce or fragmented (Figure 3). (2) In the unlimited capacity setup, Allpass can elastically scale serverful resources without restriction. These two conditions stress-test the scheduler's ability to balance latency and cost under constrained and abundant GPU supply.

In limited capacity, we compare Allpass with four baselines:

- **Tiresias-L (TL)** (Gu et al., 2019) is a SOTA GPU scheduler that adopts a multi-queue Least Attained Service (LAS) policy. It ranks jobs by attained serviced time and progressively demotes them to lower-priority queues as their executed time increases. We tested two variants: a preemptive version (TL-P: short jobs can preempt long-running ones), and a non-preemptive version (TL-NP).

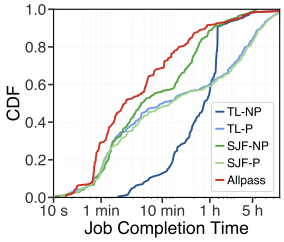


Figure 6: JCT CDF under limited serverful capacity.

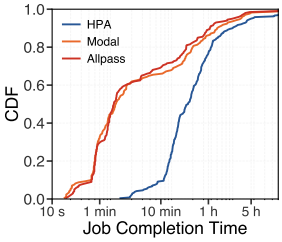


Figure 7: JCT CDF under unlimited serverful capacity.

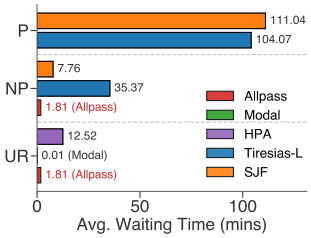


Figure 8: Comparison of average per-job waiting time.

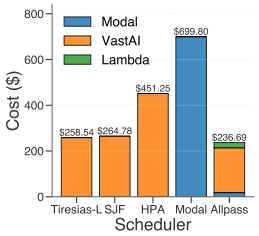


Figure 9: Scheduler cost breakdown.

- **Shortest Job First (SJF)** serves as an offline oracle that schedules jobs in ascending order of their execution times. Each job's duration is known in advance via offline profiling and supplied to the scheduler as perfect prior knowledge. Again, we tested both a preemptive variant (SJF-P) and a non-preemptive variant (SJF-NP).

In unlimited capacity, we tested two autoscaling baselines.

- **Kubernetes Horizontal Pod Autoscaling** (Kubernetes, a) (HPA), which apply autoscaling to the serverful tier by adjusting the number of GPU workers based on utilization. A new worker is added when GPU usage exceeds a defined threshold, set to 70% in our experiments.
- **Modal** (Modal, a), which runs all workloads exclusively on Modal's elastic serverless GPU tier.

Configuration and testbed. In our evaluation, Allpass uses a sliding window of $T = 30$ minutes and reevaluates scaling decisions every 1 minute. For the serverless scaler (Algorithm 1), we configure $G_{\text{up}} = 10$ and $R_{\text{up}} = 100$ to enable aggressive scaling when increasing the serverless execution budget, and set $G_{\text{dn}} = 2$ and $R_{\text{dn}} = 30$ to apply more conservative reductions. The default initial serverless execution budget F is set to 300 seconds. We assume all GPUs in the serverful tier provide uniform compute performance when estimating the aggregated service rate μT in Algorithm 2. All experiments use A100 GPUs from three clouds to ensure consistent compute performance across tiers.

5.2 End-to-End Job Completion Time

We begin by evaluating the end-to-end performance of Allpass. Figure 6 reports the CDF of job completion time (JCT) under limited serverful capacity. Allpass consistently achieves better performance across all percentiles, particularly for short jobs: 68.5% of jobs in Allpass finish within 10 minutes, compared to 12.5%, 48%, 54.5%, and 46.5% under TL-NP, TL-P, SJF-NP, and SJF-P, respectively. TL-NP exhibits the most severe performance degradation, as it adopts a LAS-style policy without preemption, leading to persistent head-of-line blocking when long jobs occupy the front of queue. SJF-P performs worse than SJF-NP, as its proportional-share (Waldspurger & Weihl, 1994) scheduling introduces queueing delays and frequent checkpoint/restore overheads. These results highlight that preemption alone is not a panacea: under bursty, heterogeneous workloads, the cost of preemption may outweigh its benefits.

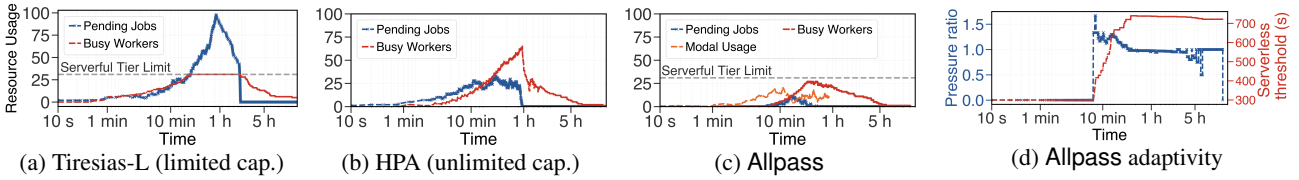
Figure 7 shows the results in the unlimited serverful capacity setting. Allpass achieves performance nearly identical to Modal, indicating its ability to avoid prolonged cold starts by leveraging serverless GPUs effectively. In contrast, HPA, although capable of dynamically scaling serverful resources, performs substantially worse—its average job duration is 2.53 times longer than that of Allpass, primarily due to the high startup cost of provisioning new serverful workers (see Table 3) and the delayed scaling response to workload bursts. These findings suggest that elasticity alone does not ensure responsiveness: serverful provisioning introduces startup delays and accumulated queueing costs before new instances become available. Meanwhile, Allpass seamlessly leverages serverless tier as a high-priority buffer to absorb bursts of early-terminated jobs immediately, maintaining serverful queue stability and overall performance. As a result, Allpass never reaches the upper capacity bound of the serverful tier, demonstrating its ability to balance short-job latency and long-job efficiency through hybrid cloud orchestration.

5.3 Waiting Time and Overhead

Figure 8 shows the average job waiting time under different baselines and decomposes Allpass's latency into queueing delay and demotion overhead. Modal achieves the lowest waiting time—just 0.1 minutes on average—by provisioning a dedicated serverless GPU worker for each job. Allpass closely follows, with an average queueing delay of 0.4 minutes and an additional 1.4 minutes of demotion overhead—primarily due to model checkpoint/restore—when transitioning jobs from the serverless to the serverful tier. This overhead is bounded, as roughly 70% of jobs complete entirely within the serverless tier and avoid demotion altogether. In contrast, preemptive schedulers (TL-P and SJF-P) suffer significantly higher waiting times compared to their non-preemptive counterparts, as frequent preemptions lead to repeated checkpoint–resume operations and prolonged queueing delays that accumulate across scheduling intervals. Overall, these results highlight that Allpass achieves near-optimal responsiveness with bounded overhead.

5.4 Cost

We next analyze the monetary efficiency of Allpass. Figure 9 compares the cost breakdown of different schedulers,



(a) Tiresias-L (limited cap.) (b) HPA (unlimited cap.) (c) Allpass (d) Allpass adaptivity

Figure 10: Scheduling dynamics over time. (a,b,c) show the number of pending jobs in queue and active serverful/serverless workers under Tiresias-L (with limited serverful capacity), HPA (with unlimited serverful capacity), and Allpass. (d) illustrates how Allpass adaptively adjusts the serverless threshold in response to workload pressure.

and Figure 12 illustrates Allpass's serverful cost over time. Allpass achieves a comparable overall cost to Tiresias-L and SJF, both of which run entirely on serverful instances—$236.69, $258.54, and $264.78, respectively. This is because Allpass strategically leverages the serverless tier—accounting for only $18.02 out of $236.69—to absorb short-job arrivals, minimizing the waiting times in queue and thereby the use of on-demand serverful instances. In contrast, HPA incurs significantly higher cost due to aggressive over-provisioning during workload spikes, resulting in an average cost that is 1.9× higher than Allpass. Modal, which executes all jobs entirely on serverless workers, incurs the highest monetary cost—2.95× that of Allpass—owing to its reliance on fully elastic but expensive serverless pricing. Allpass' serverful-tier workers save on average 64.7% to Modal workers (see Figure 12 in §A.2). Importantly, the cost from Lambda and Modal represents only a small fraction of Allpass' total cost, since Allpass uses Lambda as a fallback, infrequent operation and Allpass bounds the execution time in the serverless tier to limit the expense.

5.5 Adaptive Scheduling Dynamics

Figure 10 illustrates the dynamics of different schedulers over time. Tiresias-L suffers severe resource contention when the serverful capacity is capped at 30 workers, causing job queue length to grow rapidly as pending jobs accumulate. HPA alleviates queueing by scaling out to over 60 serverful workers. However, the high startup cost of serverful workers causes its scaling response to lag behind workload surges, resulting in backlog accumulation as pending jobs outpace provisioning. Allpass, on the other hand, demonstrates smooth and adaptive scaling: during the first 10-minute to 1-hour time period, short-lived jobs workloads are absorbed by ephemeral serverless workers on Modal, and around 70% of all jobs complete entirely within the serverless tier without ever entering the serverful job queue. As the burst subsides, Allpass provisions up to 27 serverful workers—remaining below the configured capacity limit of 30—and quickly drains the pending jobs from the queue.

Figure 10(d) shows how Allpass dynamically adjusts its serverless execution threshold F in response to workload pressure. The serverless threshold F is set to 300s initially. As the pressure ratio rises above 1 under job arrivals, Allpass rapidly scales the serverless tier to mitigate contention at the serverful tier. After roughly one hour, only long jobs remain in the system; as the pressure gradually drops below 1, Allpass conservatively scales down the serverless tier by slowly reducing the serverless threshold. F's decreasing slope is intentionally slower than its growth phase, reflecting the conservative scaling-down policy (§3.3).

6 Related Work

GPU cluster scheduling. A large body of works has focused on cluster scheduling for deep learning training. Tiresias (Gu et al., 2019), Gandiva (Xiao et al., 2018), Themis (Mahajan et al., 2020), Chronus (Gao et al., 2021), and Optimus (Peng et al., 2018) propose policies that improve fairness, responsiveness, and efficiency in multi-tenant GPU clusters. Classic frameworks such as Quincy (Isard et al., 2009), Firmament (Gog et al., 2016), TetriSched (Tumanov et al., 2016), and related multi-resource scheduling algorithms (Grandl et al., 2016a;b; Chowdhury & Stoica, 2015) provide global optimization and fair sharing strategies for heterogeneous clusters. These solutions, however, are designed for long-running training workloads, rather than the mixture of exploratory short jobs and long fine-tuning campaigns that motivate our study.

Serverless AI scheduling. Torpor (Yu et al., 2025) and ServerlessLLM (Fu et al., 2024) explore low-latency scheduling and startup time reduction for serverless inference. INFaaS (Romero et al., 2021), Clockwork (Gujarati et al., 2020), and MArk (Zhang et al., 2019) provide automated inference scheduling and SLO-aware serving in shared clusters. None of these works address scheduling for large-scale LLM training.

Spot and heterogeneity-aware scheduling. Varuna (Athlur et al., 2022), Parcae (Duan et al., 2024), and SkyPilot (Yang et al., 2023b) explore leveraging preemptible, spot resources for cost-efficient training. SkyServe (Mao et al., 2025), HetPipe (Park et al., 2020), and Sailor (Strati et al., 2025a) train models across regions and heterogeneous GPUs. Allpass differs by leveraging serverless GPUs for LLM fine-tuning.

Refer to §A.4 for a qualitative comparison.

7 Conclusion

We present Allpass, an adaptive scheduler that unifies heterogeneous GPU clouds under a serverless abstraction for cluster LLM fine-tuning scheduling. Allpass dynamically balances fast job startups for short jobs with low-cost execution for long training jobs by orchestrating across serverless,

marketplace, and conventional GPU clouds. Our prototype-based evaluation on real-world heterogeneous commercial clouds demonstrates the efficacy of Allpass.

REFERENCES

Athlur, S., Saran, N., Sivathanu, M., Ramjee, R., and Kwatra, N. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pp. 472–487, 2022.

Audibert, A., Chen, Y., Graur, D., Klimovic, A., Šimša, J., and Thekkath, C. A. tf.data service: A case for disaggregating ml input data processing. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, SoCC '23, pp. 358–375, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703874. doi: 10.1145/3620678.3624666. URL https://doi.org/10.1145/3620678.3624666.

AWS. Amazon Web Services. https://aws.amazon.com/, a.

AWS. Amazon Elastic Block Store. https://aws.amazon.com/ebs/, b.

AWS. Amazon's Secretive GPU Strategy Pays Off as AI Demand Surges. https://finance.yahoo.com/news/amazons-secretive-gpu-strategy-pays-173744973.html, c.

AWS. AWS Lambda. https://aws.amazon.com/lambda/, d.

AWS. Amazon SageMaker: Fine-Tune a Model. https://docs.aws.amazon.com/sagemaker/latest/dg/jumpstart-fine-tune.html, e.

AWS. AWS SageMaker doc: CreateTrainingJob. https://docs.aws.amazon.com/sagemaker/latest/APIReference/API_CreateTrainingJob.html, f.

AWS. Spot Instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html, g.

Azure. Azure Functions. https://azure.microsoft.com/en-us/services/functions/#overview, a.

Azure. Azure ML doc: Configure and submit training jobs. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-set-up-training-targets?view=azureml-api-1, b.

Azure. Use Azure Spot Virtual Machines. https://learn.microsoft.com/en-us/azure/virtual-machines/spot-vms, c.

Baruah, S. K., Mok, A. K., and Rosier, L. E. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pp. 182–190. IEEE, 1990.

Bert, G. bert-large-uncased. https://huggingface.co/google-bert/bert-large-uncased.

Cao, T., Arpaci-Dusseau, A. C., Arpaci-Dusseau, R. H., and Caraza-Harter, T. Making serverless Pay-For-Use a reality with leopard. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pp. 189–204, Philadelphia, PA, April 2025. USENIX Association. ISBN 978-1-939133-46-5. URL https://www.usenix.org/conference/nsdi25/presentation/cao.

Carver, B., Zhang, J., Wang, H., Mahadik, K., and Cheng, Y. Notebookos: A replicated notebook platform for interactive training with on-demand gpus, 2025. URL https://arxiv.org/abs/2503.20591.

Chowdhury, M. and Stoica, I. Efficient coflow scheduling without prior knowledge. *ACM SIGCOMM Computer Communication Review*, 45(4):393–406, 2015.

Colab, G. Colab Notebook: Fine-tuning a language model. https://colab.research.google.com/github/huggingface/notebooks/blob/master/examples/language_modeling.ipynb, a.

Colab, G. Google Colab. https://colab.research.google.com/, b.

Coppock, P. H., Zhang, B., Solomon, E. H., Kypriotis, V., Yang, L., Sharma, B., Schatzberg, D., Mowry, T. C., and Skarlatos, D. Lithos: An operating system for efficient machine learning on gpus. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, pp. 1–17, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718700. doi: 10.1145/3731569.3764818. URL https://doi.org/10.1145/3731569.3764818.

CoreWeave. Weights & Biases by CoreWeave. https://wandb.ai/site/.

Cottier, B., Rahman, R., Fattorini, L., Maslej, N., Besiroglu, T., and Owen, D. The rising costs of training frontier ai models. *arXiv preprint arXiv:2405.21015*, 2024.

datacrunch.io. DataCrunch.io. https://datacrunch.io.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Duan, J., Song, Z., Miao, X., Xi, X., Lin, D., Xu, H., Zhang, M., and Jia, Z. Parcae: Proactive,{Liveput-Optimized}{DNN} training on preemptible instances. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 1121–1139, 2024.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Fu, Y., Xue, L., Huang, Y., Brabete, A.-O., Ustiugov, D., Patel, Y., and Mai, L. ServerlessLLM: Low-Latency serverless inference for large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 135–153, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/fu.

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

GCP. Google Compute Platform. https://cloud.google.com/, a.

GCP. Spot VMs. https://cloud.google.com/compute/docs/instances/spot, b.

Gog, I., Schwarzkopf, M., Gleave, A., Watson, R. N., and Hand, S. Firmament: Fast, centralized cluster scheduling at scale. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 99–115, 2016.

Google. Google Vertex AI: Tune Gemini models by using supervised fine-tuning. https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini-use-supervised-tuning.

Grandl, R., Chowdhury, M., Akella, A., and Ananthanarayanan, G. Altruistic scheduling in {Multi-Resource} clusters. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 65–80, 2016a.

Grandl, R., Kandula, S., Rao, S., Akella, A., and Kulkarni, J. {GRAPHENE}: Packing and {Dependency-Aware} scheduling for {Data-Parallel} clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 81–97, 2016b.

Graur, D., Mraz, O., Li, M., Pourghannad, S., Thekkath, C. A., and Klimovic, A. Pecan: Cost-Efficient ML data preprocessing with automatic transformation ordering and hybrid placement. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pp. 649–665, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-41-0. URL https://www.usenix.org/conference/atc24/presentation/graur.

Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., and Guo, C. Tiresias: A {GPU} cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 485–500, 2019.

Gujarati, A., Karimi, R., Alzayat, S., Hao, W., Kaufmann, A., Vigfusson, Y., and Mace, J. Serving {DNNs} like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 443–462, 2020.

Han, Z., Gao, C., Liu, J., Zhang, J., and Zhang, S. Q. Parameter-efficient fine-tuning for large models: A comprehensive survey, 2024. URL https://arxiv.org/abs/2403.14608.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

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 *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 709–729, 2024.

HuggingFace. Hugging Face AutoTrain. https://huggingface.co/autotrain.

Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., and Goldberg, A. Quincy: fair scheduling for distributed computing clusters. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 261–276, 2009.

JIANG, Y., Fu, F., Yao, X., Wang, T., CUI, B., Klimovic, A., and Yoneki, E. Thunderserve: High-performance and cost-efficient LLM serving in cloud environments. In *Eighth Conference on Machine Learning and Systems*, 2025. URL https://openreview.net/forum?id=44PwmgOpAt.

Kubernetes. Horizontal Pod Autoscaling. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, a.

Kubernetes. Kubernetes API v1.25: Job v1 batch. https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.25/, b.

Lambda. Lambda.ai. https://lambda.ai/.

Lian, W., Goodson, B., Pentland, E., Cook, A., Vong, C., and "Teknium". Openorca: An open dataset of gpt augmented flan reasoning traces. https://https://huggingface.co/datasets/Open-Orca/OpenOrca, 2023.

Liu, C. L. and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

Maas, A., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150, 2011.

Mahajan, K., Balasubramanian, A., Singhvi, A., Venkataraman, S., Akella, A., Phanishayee, A., and Chawla, S. Themis: Fair and efficient GPU cluster scheduling. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pp. 289–304, Santa Clara, CA, February 2020. USENIX Association. ISBN 978-1-939133-13-7. URL https://www.usenix.org/conference/nsdi20/presentation/mahajan.

Mao, Z., Xia, T., Wu, Z., Chiang, W.-L., Griggs, T., Bhardwaj, R., Yang, Z., Shenker, S., and Stoica, I. Skyserve: Serving ai models across regions and clouds with spot instances. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 159–175, 2025.

Modal. Modal. https://modal.com/, a.

Modal. Modal pricing: With Modal, you always pay for what you use and nothing more. https://modal.com/pricing, b.

Modal. Modal storage volume. https://modal.com/docs/guide/volumes, c.

OpenAI. OpenAI Batch API. https://platform.openai.com/docs/guides/batch/batch-api, a.

OpenAI. OpenAI Fine-tuning API. https://platform.openai.com/docs/api-reference/fine-tuning, b.

Park, J. H., Yun, G., Chang, M. Y., Nguyen, N. T., Lee, S., Choi, J., Noh, S. H., and Choi, Y.-r. {HetPipe}: Enabling large {DNN} training on (whimpy) heterogeneous {GPU} clusters through integration of pipelined model parallelism and data parallelism. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 307–321, 2020.

Peng, Y., Bao, Y., Chen, Y., Wu, C., and Guo, C. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–14, 2018.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 487–503, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.39. URL https://aclanthology.org/2021.eacl-main.39/.

Romero, F., Li, Q., Yadwadkar, N. J., and Kozyrakis, C. {INFaaS}: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 397–411, 2021.

Runpod. Runpod. https://www.runpod.io/, 2025.

Strati, F., Elvinger, P., Kerimoglu, T., and Klimovic, A. Ml training with cloud gpu shortages: Is cross-region the answer? In *Proceedings of the 4th Workshop on Machine Learning and Systems*, EuroMLSys '24, pp. 107–116, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400705410. doi: 10.1145/3642970.3655843. URL https://doi.org/10.1145/3642970.3655843.

Strati, F., Zhang, Z., Manos, G., Périz, I. S., Hu, Q., Chen, T., Buzcu, B., Han, S., Delgado, P., and Klimovic, A. Sailor: Automating distributed training over dynamic, heterogeneous, and geo-distributed clusters. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, pp. 204–220, New York, NY, USA, 2025a. Association for Computing Machinery. ISBN 9798400718700. doi: 10.1145/3731569.3764839.

Strati, F., Zhang, Z., Manos, G., Périz, I. S., Hu, Q., Chen, T., Buzcu, B., Han, S., Delgado, P., and Klimovic, A. Sailor: Automating distributed training over dynamic, heterogeneous, and geo-distributed clusters. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 204–220, 2025b.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama

model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Tumanov, A., Zhu, T., Park, J. W., Kozuch, M. A., Harchol-Balter, M., and Ganger, G. R. Tetrisched: global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters. In *Proceedings of the Eleventh European Conference on Computer Systems*, pp. 1–16, 2016.

Unsloth.ai. Unsloth: Easily fine-tune train LLMs Get faster with unsloth. https://unsloth.ai/.

Vast.ai. Vast.ai. https://vast.ai/.

Waldspurger, C. A. and Weihl, W. E. Lottery scheduling: Flexible Proportional-Share resource management. In *First Symposium on Operating Systems Design and Implementation (OSDI 94)*, Monterey, CA, November 1994. USENIX Association.

Wandb. Wandb Fine-tuning-related projects. https://wandb.ai/fully-connected/blog/fine-tuning, a.

Wandb. Wandb Training. https://wandb.ai/site/wb-training/, b.

Weng, Q., Xiao, W., Yu, Y., Wang, W., Wang, C., He, J., Li, Y., Zhang, L., Lin, W., and Ding, Y. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pp. 945–960, Renton, WA, April 2022. USENIX Association. ISBN 978-1-939133-27-4. URL https://www.usenix.org/conference/nsdi22/presentation/weng.

Wu, Z., Chiang, W.-L., Mao, Z., Yang, Z., Friedman, E., Shenker, S., and Stoica, I. Can't be late: optimizing spot instance savings under deadlines. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pp. 185–203, 2024.

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 *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 595–610, 2018.

Yang, Z., Wu, Z., Luo, M., Chiang, W.-L., Bhardwaj, R., Kwon, W., Zhuang, S., Luan, F. S., Mittal, G., Shenker, S., and Stoica, I. SkyPilot: An intercloud broker for sky computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 437–455, Boston, MA, April 2023a. USENIX Association. ISBN 978-1-939133-33-5. URL https://www.usenix.org/conference/nsdi23/presentation/yang-zongheng.

Yang, Z., Wu, Z., Luo, M., Chiang, W.-L., Bhardwaj, R., Kwon, W., Zhuang, S., Luan, F. S., Mittal, G., Shenker, S., et al. {SkyPilot}: An intercloud broker for sky computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 437–455, 2023b.

Yu, M., Wang, A., Chen, D., Yu, H., Luo, X., Li, Z., Wang, W., Chen, R., Nie, D., Yang, H., et al. Torpor: Gpu-enabled serverless computing for low-latency, resource-efficient inference. *arXiv preprint arXiv:2306.03622*, 2025.

Zhang, C., Yu, M., Wang, W., and Yan, F. {MArk}: Exploiting cloud services for {Cost-Effective},{SLO-Aware} machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 1049–1062, 2019.

A Appendices

A.1 Sensitivity Analysis of Serverless Scaler

Figure 11 presents the sensitivity analysis of the Allpass serverless scaler, comparing it against two fixed-threshold baselines—Fixed-60s and Fixed-600s, which set the serverless-tier threshold to 60 and 600 seconds, respectively. The adaptive policy dynamically tunes the threshold based on workload pressure, as described in Algorithm 1. As a result, it achieves a median job completion time of 2.86 minutes, improving performance by 71.01% and 35.94% over Fixed-60s and Fixed-600s, respectively.

The improvement arises because the adaptive scaler is aware of workload demand and adjusts thresholds accordingly. By increasing the serverless window during bursty short-job periods, it prevents excessive queuing in the serverful tier and sustains low latency for near-term jobs. In contrast, non-adaptive policies with smaller fixed thresholds prematurely offload jobs to the serverful tier, leading to higher queuing delay and degraded short-job performance.

A.2 Cost Saving for Serverful-Tier Workers

Figure 12 presents the real-time serverful marketplace GPU prices that Allpass obtained during the experiment. By continuously probing and greedily selecting the lowest-priced instances on the Vast.ai market, Allpass dynamically acquires GPU workers at discounted prices. Throughout the experiment, Allpass maintained an average rental price of $0.74 per GPU-hour, achieving 42.6% and 64.7% cost reductions compared to Lambda and Modal, respectively.

A.3 Allpass' Serverless GPU Function API

Listing 1 shows the example of Modal serverless function. The serverless coding paradigm allows users to define a function by registering the function (line 3), configuring the runtime environment (line 4-9) and uploading finetune job code (line 10-12). Each function is packaged with its dependencies on storage of cloud providers, where providers offer instant statup optimization for users. The function has registered can be invoked remotely (line 13-15) with second-level startup cost (Table 3). Allpass builds upon this interface by integrating Modal's user-facing API, enabling users to define functions through the same programming abstraction. This design abstracts away infrastructure management while allowing Allpass to automatically handle serverless-tier resources.

```
1  """ Example of Modal function """
2  import modal
3  app = modal.App("App name")
4  image = (
5      modal.Image.from_registry("Base image")
6      .pip_install(
7          # User defined runtime
8      )
9  )
10 @app.function(image=image)
11 def train(model_name):
12     # Training code
13 @app.local_entrypoint()
14 def main():
15     train.remote()
```

Listing 1: Serverless GPU function example (Modal-API-compatible).

```
1  """ Allpass serverless API """
2  import Allpass as aps
3  app = aps.build(
4      app_name="LLM-finetune",
5      user_name="alice",
6      runtime={
7          "base_image": "pytorch:2.3.0",
8          "pip": ["transformers==4.45.0"]
9      },
10     code="train.py"
11 )
12 app.run(
13     app_name = "Fine-tuning LLaMA3-8B",
14     template = "template.json",
15     soft_deadline= "6hr",
16     parameters = [...],
17 )
```

Listing 2: Allpass serverless API.

Listing 2 illustrates Allpass APIs. The `build()` function registers an application by specifying its name, user identity, runtime configuration, and the path to the user's training code (line 3–11). The `run()` function launches the registered application with user-defined parameters and a *soft deadline* (line 12–17). Through this interface, Allpass provides a serverless abstraction that hides the complexity of underlying heterogeneous GPU infrastructures from users.

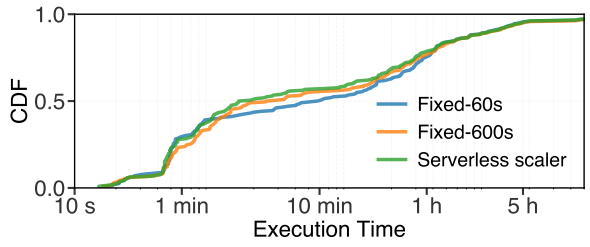


Figure 11: CDF of job execution time under different thresholds.

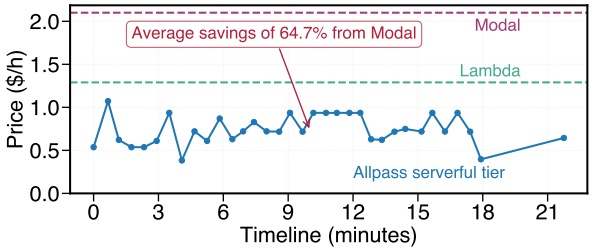


Figure 12: Cost saving by serverful workers.

System	Latency-aware	Cost-aware	Dynamic Scaling	Heterogeneous Clouds	Serverless API
Tiresias (Gu et al., 2019)	✓	✗	✗	✗	✗
SkyPilot (Yang et al., 2023b)	✗	✓	✗	✓	✗
Can't Be Late (Wu et al., 2024)	✗	✓	✗	✗	✗
Sailor (Strati et al., 2025b)	✓	✗	✓	✗	✗
Allpass (ours)	✓	✓	✓	✓	✓

Table 5: Feature comparison of Allpass and prior model training scheduling systems.

A.4 System Comparison

Table 5 compares representative GPU scheduling systems along five dimensions. Tiresias (Gu et al., 2019) focuses on reducing training turnaround using attained-service/Gittins–style queueing policies on a single cluster, without cost consideration or cross-cloud awareness. SkyPilot (Yang et al., 2023b) acts as an inter-cloud broker that provisions jobs on the cheapest available cloud and handles failover, but it is a coarse-grained orchestration layer rather than an online scheduler with per-job latency control or a serverless API. Can't Be Late (Wu et al., 2024) targets deadline-sensitive jobs on preemptible markets, formulating when to switch between spot and on-demand workers to minimize cost while meeting deadlines that lacks a serverless interface or multi-tier elasticity. Sailor (Strati et al., 2025b) enables distributed training across heterogeneous and geo-distributed GPUs. It automatically selects configurations and allocates resources to improve throughput and reduce cost. However, it focuses on intra-training optimization rather than online scheduling or serverless execution across clouds.

In contrast, Allpass offers all five capabilities shown in the table. Allpass orchestrates jobs across both serverless workers, which provide low latency for short-term or bursty jobs, and serverful workers, which deliver minimal-cost execution for long-running workloads. It manages serverless and serverful resources across multiple cloud providers—including Modal, Vast.ai, and Lambda in our prototype, and leverages adaptive scalers §3 to dynamically adjust capacity in response to workload demand. In adaption, Allpass offer serverless interface that allows users to submit fine-tuning jobs without knowledge of the underlying infrastructure.