Resilience-Aware Elastic Scaling for Cloud-Native Online DL Training on Multi-Tenant GPU Clusters

Qianhao Wu
Tencent Inc.
Shenzhen, China
polarwu@tencent.com

Jiazhi Jiang
Beijing Normal University
Zhuhai, China
jiangjz@bnu.edu.cn

Guohui Ling
Tencent Inc.
Shenzhen, China
randyling@tencent.com

Yue Pang
Tencent Inc.
Shenzhen, China
pangyue@tencent.com

ABSTRACT

Online deep learning (DL) training has become pivotal in powering real-time applications. Yet tidal workload fluctuations leave GPU clusters significantly underutilized during off-peak periods. This imbalance not only wastes GPU capacity but also exacerbates scarcity for other GPU-intensive jobs on cloud-native GPU cluster. Cluster-wide resource leasing across different tenants enabled by elastic scaling offers a promising opportunity to enhance GPU utilization for cloud-native online DL training deployments in multi-tenant GPU clusters. Existing solutions do not address the unique challenges associated with maintaining system stability during elastic scaling for online DL training jobs, including prolonged disruptions due to job reconstruction, failures arising from dependency-unaware operation triggering, and the unreliable reclamation of high-availability GPU resources.

In this paper, we introduce WeFlex, a resilience-aware elastic scaling solution engineered for cloud-native online deep learning jobs in multi-tenant GPU clusters. WeFlex enables online training jobs to lease idle GPUs for other GPU-intensive jobs during low-demand periods while ensuring rapid reclamation as demand surges. It significantly reduces the duration of training disruptions through constructing a interruption mitigation pipeline, prevents dependency-unaware operation failures via topology-aware pod orchestration, and ensure reclamation of high-availability GPU resources through right-of-return GPU leasing. Evaluations on 10,000-plus scale GPU clusters in production demonstrate that WeFlex significantly enhances GPU utilization while reliably maintaining continuous training performance.

KEYWORDS

Online Training, Scaling, Stability, GPU, Cloud

PVLDB Reference Format:
Qianhao Wu, Jiazhi Jiang, Guohui Ling, and Yue Pang. Resilience-Aware Elastic Scaling for Cloud-Native Online DL Training on Multi-Tenant GPU Clusters. PVLDB, 19(3): 375 - 387, 2025.
doi:10.14778/3778092.3778099

Proceedings of the VLDB Endowment, Vol. 19, No. 3 ISSN 2150-8097.
doi:10.14778/3778092.3778099

Jiazhi Jiang is the corresponding author. Jiazhi Jiang and Qianhao Wu contribute equally.

1 INTRODUCTION

Online deep learning (DL) training has fundamentally transformed online services, powering applications from personalized recommendations to live video analysis. Unlike traditional offline DL training, online DL training systems continuously update models with streaming data, ensuring immediate responsiveness to evolving user behavior and content trends [25, 27]. This approach is now essential for major internet platforms, with leading applications like WeChat Video processing billions of training samples every day to keep models fresh. Such real-time online training jobs are typically deployed on large-scale GPU clusters to meet their intensive computational requirements [26].

In production environments, GPU clusters are typically orchestrated using cloud-native platforms like Kubernetes (K8s)[14] in multi-tenant settings, enabling resource allocation across various services and user groups. Each DL job consists of multiple workers or instances, with each worker deployed to a pod managed by the cloud platform. In these systems, individual jobs submit independent resource requests and are allocated dedicated GPU pools, each exclusively reserved for a specific DL training job.

While these cloud-native multi-tenant platforms ensure robust resource isolation and simplify management, it leads to significant inefficiencies. A key characteristic of online training jobs is their highly volatile traffic patterns. Such workload fluctuations and multi-tenant resource demands are also prevalent in many data management environments, including stream processing engines and distributed databases[4, 15, 20]. Driven by regular user activity, these jobs experience extreme daily fluctuations. Peak demand can exceed off-peak levels by more than four times. For example, in production systems like WeChat Video, a traditional "request-and-keep" approach tailored for peak hours results in GPU clusters operating at only around 40% utilization during off-peak hours, leading to tens of millions of dollars in wasted capital annually. Meanwhile, the scarcity of GPU resources forces GPU-intensive jobs of other tenants (e.g., offline LLM training[24] or finetuning[28]) into prolonged queuing delays and prevents them from securing sufficient GPU resources.

In order to mitigate the inherent challenge of low GPU utilization in online DL training, cluster-wide resource leasing enabled by elastic scaling can be explored. Online training jobs dynamically scale up or down in response to a fluctuating GPU resource pool. Idle GPU capacity, typically available during off-peak periods, is temporarily allocated to other GPU-intensive jobs and reclaimed when demand increases. However, current elastic scaling solutions for deep learning workloads are either framework-specific or designed primarily for stateless tasks like deep learning inference services. They lack the capability to support widely diverse deep

learning training frameworks on multi-tenant GPU clusters while ensuring the continuous stability and service level objects (SLO) of online training workloads.

In this paper, we present WeFlex, a resilience-aware elastic scaling system tailored for cloud-native online deep learning training on multi-tenant GPU clusters. In contrast to existing solutions that are confined to specific DL training frameworks and stateless jobs, WeFlex facilitates unified resource leasing and elastic scaling of GPU capacity across multiple tenants' GPU-intensive deep learning workloads to enhance GPU utilization at cloud platform level. It incorporates techniques such as interruption mitigation pipeline, topology-aware pod orchestration, and right-of-return GPU leasing to ensure the stability and robustness for the elastic scaling of online training process.

The interruption mitigation pipeline is essential for minimizing training disruptions during elastic scaling. Elastic scaling often causes lengthy interruptions due to resource reclamation, checkpoint downloads, and model weight redistribution. When resources are reclaimed or reallocated, training instances typically require state reconstruction and data redistribution, which can take tens of minutes. During this downtime, real-time streaming data cannot be processed, leading to stale model updates and reduced service quality. WeFlex addresses this by performing in-situ resource adjustment, bypassing Kubernetes' default release-and-reclaim process. With local loading and standby prelaunch, WeFlex avoids costly steps such as image pulling and checkpoint downloading, ensuring faster and smoother scaling.

To prevent failures from ignoring framework dependencies, WeFlex introduces topology-aware pod orchestration. Existing cloud platforms lack awareness of the internal topology and dependencies in distributed frameworks like MPI, PyTorch, and Ray [17], which complicates elastic scaling. When resources are adjusted, the orchestrator may execute commands in the wrong order, often causing job restarts to fail and leading to long interruptions. WeFlex addresses this by parsing each framework's dependency rules and building a topology-aware orchestrator, ensuring that scaling commands are executed in the correct order for training jobs.

To enable precise and reliable GPU reclamation, WeFlex introduces right-of-return GPU leasing, combining resource tagging with priority-based scheduling for deterministic allocation and recovery. Traditional cloud management treats GPUs as interchangeable and lacks fine-grained device tracking, making it hard to ensure that the original job receives the same number and quality of GPUs after leasing. This can result in missing or lower-quality devices, causing instability and degraded service quality. With right-of-return leasing, WeFlex accurately tracks each GPU, ensuring timely and quality-preserving reclamation during elastic scaling. While WeFlex is motivated by the demands of large-scale online deep learning training, its resilience-aware elastic scaling techniques are also applicable to a broader range of data-intensive systems. Cloud-native databases, large-scale stream processing engines, and real-time analytics platforms face similar elasticity and SLO challenges [4, 15, 20], and can benefit from WeFlex's platform-level mechanisms for fine-grained resource leasing, dependency-aware coordination, and stability preservation. Our contributions are as follows.

- Leveraging production traces, we expose the inefficiencies inherent in separately managing cloud-native online DL training and other GPU-intensive jobs on multi-tenant GPU clusters. In particular, we observe a tidal traffic pattern that results in low utilization of online training resources, while GPU-intensive jobs of other tenants suffer from resource scarcity.
- Cluster-wide resource leasing enabled by elastic scaling is proposed to enhance GPU utilization. Further investigation reveals scaling-induced obstacles to sustaining stability and robustness in online training, including training interruptions caused by job reconstruction, dependency-unaware operation triggering failures, and unreliable reclamation of GPU resources.
- We present WeFlex, an elastic scaling solution featuring sophisticated stability-preservation mechanisms for cloud-native online training. In detail, WeFlex incorporates interruption mitigation pipeline, topology-aware pod orchestration, and right-of-return GPU leasing to collectively mitigate stability challenges inherent in elastic scaling.
- WeFlex is validated and deployed on large-scale production clusters comprising 10,000-plus GPUs, demonstrating its effectiveness in improving resource utilization and ensuring the robustness of online training jobs.

2 BACKGROUND AND MOTIVATION

In this section, we introduce online DL training in corporations like Tencent and WeChat, and motivate the design of WeFlex for elastic scaling of cloud-native online DL training.

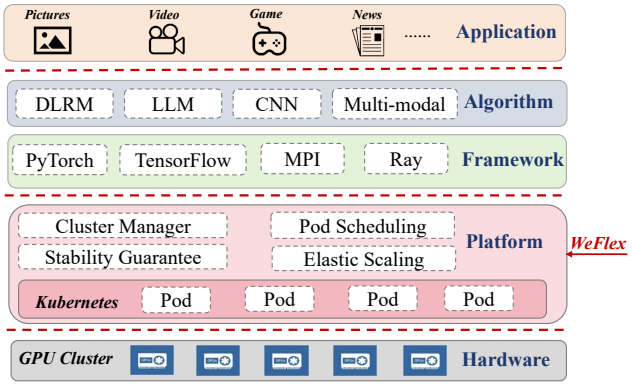

Figure 1: The summarized software stack of cloud-native DL training cluster in production, highlighting WeFlex's position in software stack.

2.1 Software Stack of Cloud-Native DL Training

As depicted in Figure 1, the software-hardware stack for cloud-native DL training is reshaping organizational structures and workflows across enterprises.

Application Layer. Breakthroughs in DL algorithms have driven rapid advances in recommendation, video/picture processing and natural language processing systems. At corporations like Tencent, applications span from news recommendations to video processing. For instance, customer service teams leverage real-time online

training with DLRM[18] on streaming user data to dynamically improve the effect of video and news recommendations in WeChat.

Algorithm and Framework Layer. To support efficient and scalable training, algorithm teams develop business-specific DL models using frameworks such as PyTorch. Large-scale distributed training is enabled by communication and synchronization libraries like Horovod, Ray, and MPI, facilitating seamless scaling across extensive GPU clusters.

Cloud Platform Layer. The platform layer, typically built on Kubernetes, orchestrates GPU clusters at scale, supporting diverse frameworks and algorithmic needs across teams. It manages resource allocation, robust job scheduling, and load balancing while ensuring reliable, stable resource provisioning to upper layers. This maximizes hardware utilization, reduces operational complexity, and guarantees continuous, real-time DL service delivery.

2.2 Why Elastic Scaling for Online DL Training?

DL training jobs in production environments differ significantly from jobs on testbed, introducing unique resource management challenges in multi-tenant GPU clusters.

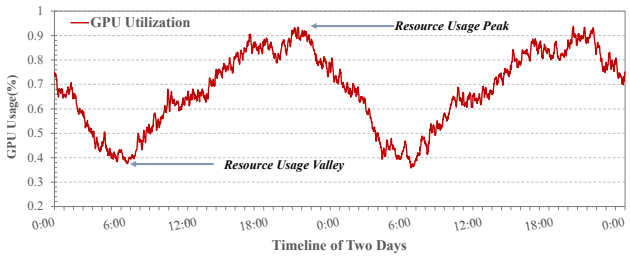

Figure 2: GPU utilization of online training jobs, based on the average usage of 1000 GPUs allocated to four online training jobs in WeChat, is measured over a 48-hour period from March 24 to 25, 2025. Utilization fluctuates between 39% during off-peak hours and 92% during peak hours.

Workload fluctuation in online DL training. Online training workloads, similar to other web or DL inference services, are highly dynamic and closely mirror user activity. Streaming data volumes for training can vary dramatically throughout the day, resulting in sharply defined peak and off-peak periods. To accommodate peak traffic, GPU resources must be overprovisioned, which in turn leads to considerable underutilization during off-peak periods. We plot the GPU utilization in one of our online training jobs with 2-minute intervals for two days time as shown in Figure 2.

Resource scarcity for other GPU-intensive workloads. GPU clusters used for deep learning training in production environments typically support multiple tenants, including online and other GPU-intensive training workloads, simultaneously within a cloud-native platform. A notable observation is that many of these GPU-intensive jobs encounter insufficient resource request fulfillment. As depicted in Figure 3, the hourly average resource request fulfillment ratio for these jobs, corresponding to the same day as in Figure 2, reveals that a significant proportion of jobs (up to 100%) fail to fully satisfy their resource requirements. The overall resource request fulfillment ratio averages only about 40%, which has a detrimental effect on the completion times of other GPU-intensive jobs. This challenge is not solely attributable to cluster-wide resource scarcity. In fact, during off-peak periods, the average GPU utilization for online training is only 40%, indicating a significant amount of idle GPU capacity across the entire cluster. This under-utilization creates an opportunity for resource leasing between workloads from different tenants.

Resource leasing across tenants enabled by elastic scaling. To better accommodate the dynamic fluctuations in cluster capacity and optimize the utilization of loaned online training resources, WeFlex employs elastic scaling. While elastic scaling has been incorporated into certain ML frameworks, relying solely on specific frameworks for resource leasing across diverse business units and algorithm teams is impractical. Thus, elastic scaling should be uniformly initiated and managed on the cloud platform. Elastic scaling enhances the efficiency of resource leasing by enabling online training jobs to scale in during off-peak periods, thereby releasing idle GPU resources, rather than adhering to a "request-and-keep" approach. With the temporarily freed resources, other GPU-intensive jobs can dynamically scale up, utilizing additional workers and GPUs to expedite job completion. This allows online training jobs to avoid holding excess GPUs during low-demand periods, reducing capital expenditures. Simultaneously, other jobs can temporarily lease these GPUs at a lower cost and priority.

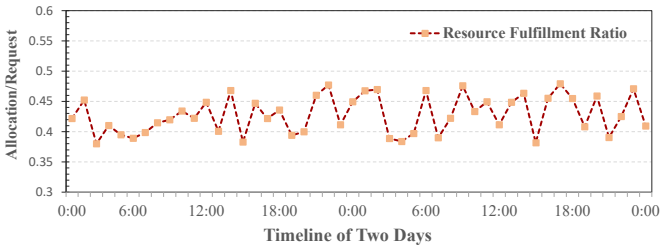

Figure 3: Resource fulfillment ratio over a 48-hour period for all newly submitted jobs on the cloud platform in WeChat. The y-axis shows, on an hourly basis, the ratio of GPUs actually allocated by the cloud platform to those requested by applications. Data are aggregated per hour across all tenants and span two consecutive days. A low fulfillment ratio indicates that submitted jobs were unable to acquire their full requested capacity, resulting in either queuing under gang scheduling until additional GPUs became available [11] or running with fewer resources than requested. Both situations lead to increased overall completion time.

2.3 Associated Challenges of Resource Leasing

While resource leasing enabled by elastic scaling can significantly boost resource utilization for online training, concurrently shortening queuing delays and expediting job completion for other GPU-intensive workloads, it also introduces potential scalability challenges in distributed training systems. With our production traces, there are three associated issues affecting the stability of online training jobs.

C_1 : Elastic scaling of GPU resources would significantly disrupt ongoing online training jobs. In online DL training

scenarios, performing GPU resource leasing through elastic scaling frequently necessitates restarting or reconfiguring training processes, causing significant training disruptions. For instance, distributed training frameworks like MPI require consistent cluster configurations. Changes in GPU capacity during scaling affect critical parameters such as rank assignments and world size of MPI-based solution. TorchElastic automatically detect resource capacity changes, immediately restarting all instances to rebuild collective communication groups and reloading model weights from checkpoints. These inherently necessitate interrupting and restarting the ongoing training processes.

In production, the cloud-native platform uniformly manages the restart procedure across diverse training frameworks. This process encompasses several time-intensive steps, including Kubernetes' GPU resource release-and-reclaim mechanism, re-downloading checkpoints (often tens to hundreds of gigabytes), pulling container images, and redistributing model parameters. Even frameworks designed to avoid full restarts must pause training requests and synchronize model weights, a procedure nearly as costly. Empirically, the duration of this scaling operation typically ranges from 10 to over 40 minutes, depending on workload scale, model complexity, and storage throughput. As shown in Figure 4, adjusting GPU allocations in two jobs within WeChat results in streaming-data training interruptions lasting approximately 20–30 minutes. Such prolonged interruptions can lead to substantial losses of real-time streaming data, potentially affecting tens of millions of user-generated data points. The consequent data loss significantly impacts model convergence, recommendation accuracy, and overall service effectiveness, presenting an unacceptable risk for production-scale online training workloads.

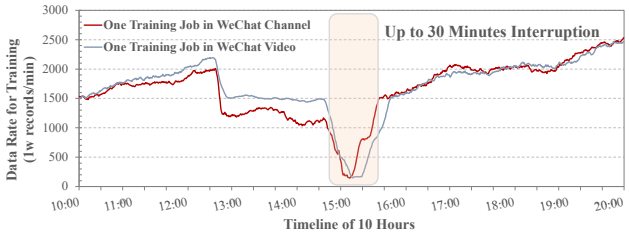

Figure 4: Interruption of online training jobs caused by restarting jobs. The y-axis indicates the number of training records processed per minute. The monitored training data flow spans a duration of 10 hours. Both jobs experienced restarts at approximately 15:00, resulting in interruptions lasting around 20–30 minutes. During interruptions, streaming data could not be consumed by training jobs. Given the buffer queue's capacity limit of 20 million records (approximately two minutes' worth of data), any interruption exceeding this threshold leads to data loss due to buffer overflow.

C_2 : **Effective elastic scaling for online training in cloud platforms requires topology-aware coordination to prevent premature command execution.** Distributed DL training frameworks typically deploy multiple training pods, each with distinct roles (e.g., coordinator and workers), and enforce stringent synchronization requirements. For example, MPI-based training jobs necessitate that the coordinator pod waits until all worker pods are fully initialized before issuing collective commands (e.g., *mpirun*). In contrast, frameworks such as PyTorch and Ray[17] require the coordinator pod to be launched first, followed by worker pods that actively register. Without topology awareness, cloud platforms may inadvertently trigger commands prematurely, resulting in operation failures. Additionally, during resource reclamation or elastic scaling, cloud platforms may mistakenly reclaim critical pods, such as a coordinator pod. Premature removal of these essential instances can immediately halt the entire training process, leaving allocated resources unusable and necessitating manual intervention or costly restarts. To address this issue, it is crucial for cloud platforms to integrate framework-specific dependencies and topology awareness into their orchestration, ensuring stable and efficient elastic scaling while maintaining stability.

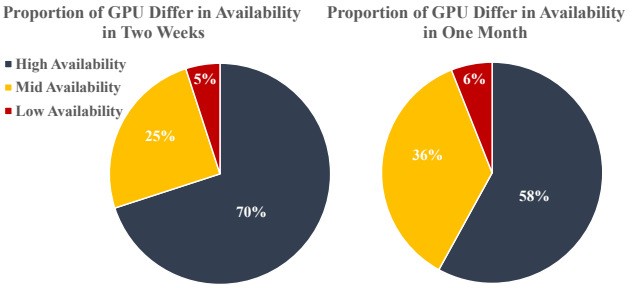

Figure 5: Production traces of GPU availability over two weeks and one month. High-Availability refers to GPUs remaining fully available (100%) throughout the monitored intervals, mid-availability refers to GPUs maintaining availability duration between 97% and 100%, while low-availability GPUs were available less than 97% of monitoring duration.

C_3 : **Reliable reclamation of GPUs after temporary leasing.** Effective GPU leasing in production environments imposes two stringent requirements. First, all GPUs loaned to other tenants must be promptly and fully reclaimed by the originating job during scale-up, thereby preventing scenarios where resources become unavailable due to further preemption or reassignment. Second, the quality of reclaimed resources must be preserved: high-availability GPUs originally allocated to online training jobs must be returned in kind, rather than substituted with lower-availability devices. As evidenced by Figure 5, although 70% of GPUs maintain perfect availability over two weeks and 58% over a month, GPU failure rates remain substantially higher than those of CPUs. To mitigate this risk, production and algorithm teams have traditionally relied on manual whitelists of high-availability GPUs to safeguard the stability of online training jobs. These factors underscore the critical need for precise and dependable reclamation of both the quantity and quality of GPUs throughout dynamic leasing and scaling cycles.

However, native Kubernetes scheduling mechanisms lack the ability to distinguish device provenance and treat all resources of equivalent priority as interchangeable, providing no assurance that the originally entrusted and equivalently provisioned GPUs will be returned to their initial owners. Addressing this challenge requires

a more sophisticated GPU tracking and resource management system capable of enforcing robust right-of-return semantics, thereby ensuring stable and predictable resource availability for critical online training workloads.

2.4 Position and Distinctive Features of WeFlex

WeFlex can be distinguished from other solutions in three key aspects. 1) First, WeFlex is a **cloud platform-level solution** that operates independently of particular DL framework for improving GPU utilization, which supports diverse business units and algorithm teams that typically use their own customized DL frameworks. In contrast, systems like Horovod Elastic[9] and TorchElastic[22], despite offering elastic scaling mechanisms, are tightly integrated with particular DL frameworks. This inherent coupling restricts users by confining them to associated software stacks. 2) Second, WeFlex introduces **cluster-wide resource leasing** as a novel dimension and aims to loan GPU resources between online training and other jobs in multi-tenant GPU cluster, an aspect rarely explored in existing systems. Recent solutions exploit shared DL infrastructure such as Mudi [3], twine[29], Gandiva[32], Pollux[23] and AntMan[33], their scope typically remains confined to cluster scheduling and co-locating varied workloads rather than resource elasticity or leasing across tenants. Instead, WeFlex does not focus on job scheduling or packing strategies to minimize job completion time. 3) Third, WeFlex focuses on **resilience-aware elastic scaling** solution for online training stability. Recent studies such as Lyra [13], FaPES [34], Faro[10] and Primus [2] successfully achieved elastic scaling for stateless DL inference workloads through dynamic replica adjustment and proxy-based service management. However, their solutions can't be employed directly in online training, where on-the-fly scaling involves challenges such as prolonged training disruptions due to job reconfiguration or reconstruction, failures caused by dependency-unaware operation, and unreliable reclamation of high-availability GPU resources.

While WeFlex is motivated by the requirements of online deep learning training, its underlying design principles are broadly applicable to a wide spectrum of systems including distributed data-intensive systems. WeFlex delivers resilience-aware elastic resource provisioning, characterized by sophisticated GPU resource management, minimized service interruptions, and dependency-aware orchestration. These features are critical to multi-tenant environments including cloud-native databases, data lakes, and real-time analytics platforms, and are applicable to a broad range of systems facing challenges in elastic scaling, tenant fairness, and distributed orchestration.

3 THE DESIGN OF WEFLEX

In this section, we present an overview that highlights the novel components and workflow in WeFlex for ensuring stability when employing elastic scaling for online training.

3.1 Overall Architecture.

WeFlex is deployed on top of a cloud-native GPU cluster managed by Kubernetes where each physical GPU device is a fundamental lending unit. As depicted in Figure 6, WeFlex's architecture integrates three principal modules to realize resilience-aware elastic scaling. Specifically, the right-of-return GPU leasing module orchestrates GPU allocation and pod placement during scaling events, ensuring timely reclamation of high-availability GPUs and sufficient transient resources after temporary loans. Complementing this, the interruption mitigation pipeline proactively manages online training job launches through mechanisms such as in-situ pod restarts, which reclaim original nodes and leverage cached data, as well as prelaunch operations on standby nodes to minimize startup delays when original nodes are unavailable. Topology-aware pod orchestrator governs high-level control operations (e.g., start, terminate) on DL training pods, enforcing dependency-driven readiness checks so commands never execute prematurely and risk failure.

3.2 Elastic Scaling Workflow.

As shown in Figure 6, WeFlex's design is centered around two key workflows: a lightweight control-plane process for pod scheduling and resource allocation, and a heavyweight execution-plane process for job launch and reconstruction.

Upon detecting changes in online training workload, the scheduler at the cluster level consumes lease/reclaim directives from the cloud platform that specify both the quantity and identities of GPUs to be adjusted for each online training job (step a), determine GPUs to be occupied/returned of other workloads (step b) and issues virtual reassignment commands to the underlying resource manager, seamlessly moving the chosen servers across tenant boundaries (step c). If not enough devices can be reclaimed, scheduler supplements from standby resource pool to ensure one-to-one matching of leased and returned GPUs (step d). Once the virtual reassignment is complete, WeFlex notifies the affected jobs to perform reconfiguration or restart, thereby applying the updated GPU assignments (step e).

Online training traffic follows highly regular tidal cycles, interruption mitigation pipeline employs in-situ resource adjust and prelaunch mechanisms, enabling reclaimed or newly allocated nodes to reuse existing environments and reduce job restart overhead (step ①). Upon receiving the actual scaling command, WeFlex instructs its topology-aware orchestrator to poll each pod's status and confirm readiness for reconstruction (step ②). Once all pods are ready, dispatches rebuild or reconfiguration commands to the online training containers (step ③) and simultaneously signals the workloads that had loaned the reclaimed GPUs to resume under their new resource assignments (step ④).

3.3 Interruption Mitigation Pipeline.

As shown in Figure 7, WeFlex constructs an interruption mitigation pipeline to proactively mitigate the duration of training interruptions with elastic scaling of containerized DL training jobs. Take scaling up operation as an instance, restarting DL training job involves multiple latency-sensitive operations, including GPU release and reclaim, container image pulls, checkpoint downloads. By orchestrating in-place GPU capacity upgrades, proactive prelaunch, and short-duration streaming data staging, WeFlex compresses interruptions induced by the elastic-scaling window to seconds.

Transforming K8s for In-Situ Restart. In standard Kubernetes workflows, restarting a workload involves terminating all associated pods. This action compels jobs to release their current

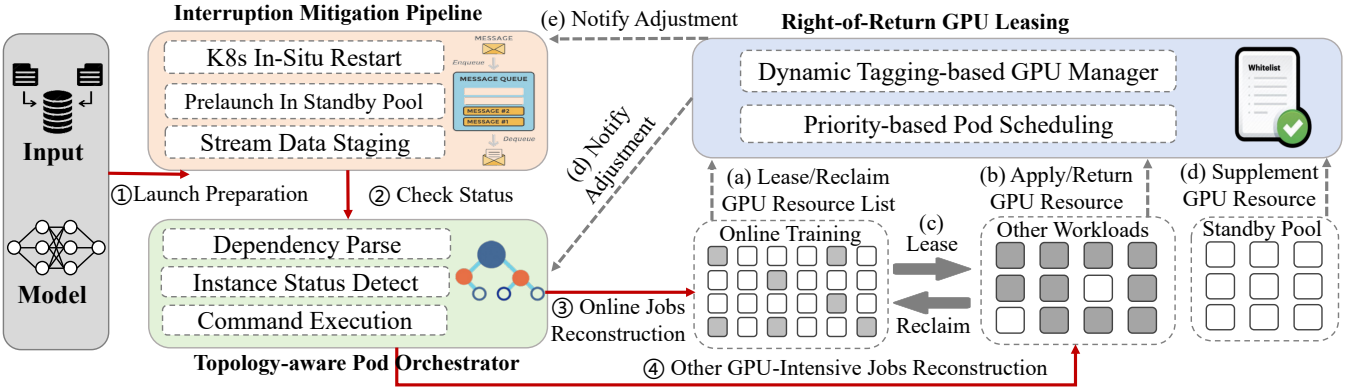

Figure 6: WeFlex system architecture. Solid red lines indicate job execution workflow. Gray dashed lines represent resource leasing and pod scheduling workflow. Each square represents a GPU server; the gray ones are in use.

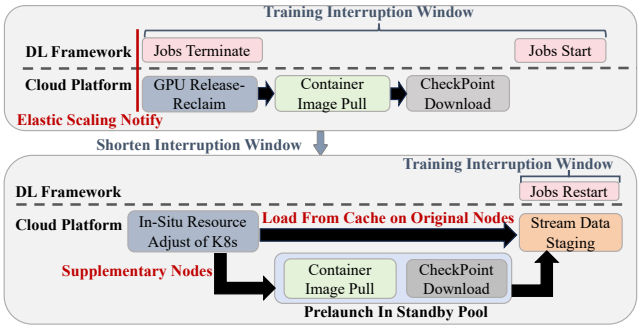

Figure 7: Online training interruption mitigation pipeline, incorporating in-place job upgrades, proactive prelaunch procedures, and short-duration streaming data staging.

GPU allocations back into the public shared resource pool, after which they must reacquire resources. Consequently, GPUs previously occupied by the restarted workload might quickly become allocated to other tenants, leading to unpredictable delays as jobs wait for new GPU assignments, thereby extending interruptions.

To overcome these inefficiencies, WeFlex augments Kubernetes with a custom control loop and node-level agent to deliver true in-situ restarts. At the control-plane tier, a WeFlex operator watches for targeted PodSpec edits(`template.spec.containers[x].image`), and issues a strategic "hot-restart" patch via the standard Kubernetes API. This patch carries a lightweight annotation that flags the affected container for in-place update rather than full pod recreation. On each node, the WeFlex agent integrates with the container runtime and the native Kubelet sync loop. When it detects the hot-restart annotation, the agent gracefully terminates only the specified container process and immediately spawns a new instance with the updated image, all within the existing pod sandbox. Crucially, it preserves: the UID of pod, network namespace (IP address), existing volume mounts and ephemeral storage, shared-memory regions (e.g., IPC namespaces), GPU device bindings, and cgroup allocations. By sidestepping the default delete-then-create cycle, WeFlex avoids relinquishing GPU allocations back to the cluster pool and eliminates the overhead of re-mounting volumes or re-initializing shared memory. The result is genuine hot-restart semantics, characterized by in-place updates of container images, persistent GPU-node bindings, and the elimination of the conventional Kubernetes resource recycling and reacquisition procedures associated with job restarts.

Local Loading Complemented by Standby Prelaunch. To further shrink interruption windows caused by container image pulling and checkpoint downloading, WeFlex first tries to bring exactly the same GPU nodes back to the job that leased them. Section 25 employs resource leasing strategies, tracking and recording leased resources to ensure that reclaimed original GPU nodes preferentially return to the online training jobs. These nodes still hold the required container images and the most recent checkpoints on their local disk, restarting jobs on them avoids massive network transfer of image and checkpoint, thus incurring virtually zero restart latency.

When some on-loaned GPU cannot be reclaimed, owing to hardware failures or other unexpected issue, WeFlex falls back to a public resource standby pool comprising dozens of spare nodes. These spare nodes are proactively pre-warmed well before being allocated to online training jobs. WeFlex launches a background pipeline that pulls the relevant container images, downloads the correct checkpoints, and redistributes models among GPUs well ahead of anticipated scaling events. The duration required for these prelaunch operations, defined as the prelaunch window (W), is carefully computed through the following formulas:

$$T_{\text{img}}^{\text{pre}} = (1 - \eta_{\text{cache}}) \times \frac{S_{\text{img}}}{\min(B_{\text{net},}, B_{\text{reg}})} \tag{1}$$

$$T_{\text{ckpt}}^{\text{pre}} = (1 - \eta_{\text{ckpt}}) \times \frac{S_{\text{ckpt}}}{\min(B_{\text{net},}, B_{\text{stor}})} + \frac{S_{\text{ckpt}}}{B_{\text{gpu}}} \tag{2}$$

$$W = T_{\text{gpu}_{\text{adj}}} + \max(T_{\text{img}}^{\text{pre}}, T_{\text{ckpt}}^{\text{pre}}) \tag{3}$$

Here, η_{cache} and η_{ckpt} represent local cache hit ratios for images and checkpoints, respectively, derived from node-level metrics. S_{img} and S_{ckpt} denote data sizes, while B_{net}, B_{reg}, and B_{stor} correspond

to network, registry, and storage throughputs, respectively. B_{gpu} refers to GPU interconnect bandwidth. By scheduling prelaunch operations sufficiently ahead of anticipated scaling events (within window W), WeFlex ensures full preparation of all nodes. Regardless of whether resources are reclaimed from returned leases or provisioned from the standby pool, each node arrives ready for immediate use. As a result, interruption windows are both minimized and predictable.

Short-Duration Streaming-Data Staging. As shown in Figure 7, even with in-situ restarts and pre-launching, a brief interruption (tens of seconds) caused by pod restart remains unavoidable, potentially causing loss of streaming inputs. To prevent data gaps, WeFlex employs a lightweight staging buffer via message queue (e.g., pulsar[1]). Incoming data for training job is replicated to a dedicated staging topic for a duration W_{stage}, chosen to cover the expected restart window plus a safety margin. Upon job relaunch, the data pipeline drains the staging queue before resuming live ingestion, replaying buffered records in original order. The queue's offset-tracking ensures exactly-once processing semantics, preserving training accuracy. By staging streaming data for the short restart interval, WeFlex eliminates data loss and maintains end-to-end training continuity.

3.4 Topology-Aware Instance Orchestration.

Elastic scaling is launched by the cloud platform. However, the native platform lacks awareness of topology relationships among instances of higher-level DL frameworks, potentially causing issued commands to fail. To robustly handle startup and teardown semantics specific to training frameworks during elastic scaling, WeFlex introduces a topology-aware instance orchestrator at the cloud platform level as shown in Figure 8. Its orchestration logic comprises three clearly delineated stages: Directed Acyclic Graph (DAG) construction from framework-provided dependency definitions, continuous instance state monitoring, and condition-driven command execution guided by DAG queries.

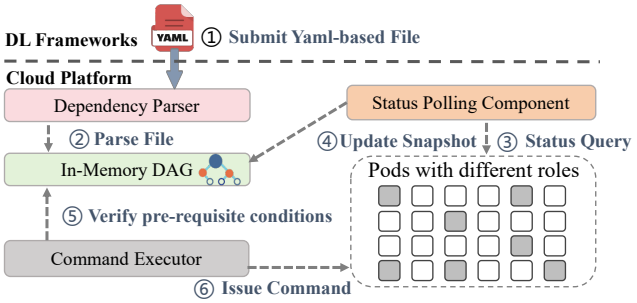

Figure 8: Topology-aware pod orchestration and framework command execution.

Initially, WeFlex ingests a YAML-based dependency file provided by each distributed DL training framework. This file explicitly enumerates operational commands, the associated node roles (e.g., coordinator, worker), and the execution preconditions for each command. Specifically, the dependency file includes: i) *Commands*: discrete operational actions (e.g., running *mpirun* on the coordinator node). ii) *Roles*: categories of nodes (e.g., coordinator, worker) designated to execute the commands. iii) *Preconditions*: conditions that must be satisfied prior to command execution (e.g., worker pods achieving *ready* state in MPI-based frameworks, PyTorch and Ray require coordinator pod to be launched before worker pods).

Upon ingestion, WeFlex validates the YAML schema and translates it into an in-memory DAG. Each node in the DAG represents an operational command. The *Roles* are associated metadata attached to these command nodes, explicitly identifying which nodes in the cluster should execute each command. Preconditions determine the edges between DAG nodes, representing dependencies where one command must precede another. An edge from command A to command B indicates that command B can only be executed once all preconditions represented by command A's completion are fulfilled. For example, the command to run *mpirun* on the coordinator node would have an edge from the worker node initialization commands, indicating that the coordinator node's *mpirun* command can only be executed after all worker nodes have successfully completed their startup commands.

Following DAG construction, WeFlex deploys a continuous polling component responsible for querying and tracking preconditions. This component actively monitors live instance statuses via Kubernetes API queries, filtering by specific role-based labels (e.g., role=worker, role=coordinator). Readiness and liveness states of pods are efficiently aggregated into a regularly updated snapshot. Any status transitions, such as pods moving from *pending* to *ready*, prompt immediate updates to this snapshot.

When a command execution request is submitted, the polling component consults the DAG to verify that all prerequisite conditions (upstream dependencies) are satisfied. This verification leverages both the dependency edges in the DAG and the latest status snapshot, ensuring conditions such as "all worker nodes are ready" are explicitly checked before command initiation. Command execution occurs strictly after DAG validation and status confirmation. Commands are issued through idempotent Kubernetes exec operations or Kubernetes jobs, depending on the specific command semantics (e.g., *mpirun*). Each execution step is meticulously logged and incorporates retry mechanisms governed by a controlled backoff policy to handle transient failures gracefully.

3.5 Right-of-Return GPU Leasing.

In WeChat's production environment, workloads are classified as either high-priority (HP) or low-priority (LP) jobs. Resources allocated to HP jobs are strictly protected and cannot be preempted by any other jobs, whereas GPU resources assigned to LP jobs remain preemptible and may be reclaimed as needed to satisfy HP demands. Elastic scaling introduces additional complexity to this resource model. When an online training job (classified as HP) scales down, its released high-availability GPUs can be temporarily leased to jobs from other tenants, including both HP and LP jobs. To ensure resource guarantees for the original owner, the leasing process is subject to two critical requirements: first, all leased GPUs must be promptly and completely reclaimed when the original HP job scales up; second, the set of reclaimed devices must maintain both the original hardware types and the proportion of HA GPUs before

elastic scaling. To realize these guarantees, WeFlex integrates a dynamic tagging-based GPU manager with a priority-aware scheduler, enabling efficient utilization of idle GPUs by low-priority jobs while preserving resource stability.

Algorithm 1: Priority-based Pod Scheduling

Input: $\mathcal{G}$: set of candidate GPUs;
Need: number of GPUs requested by the pod;
Prio $\in$ {High, Low}: pod priority;
ReqRel $\in$ {HA, MA, LA}: required reliability class;
IsReclaim: whether the scheduling is triggered by an online training resource reclamation event;
Match(*Pod*, *Owner*): true iff pod belongs to *Owner*;
PickGPUs(n): prefer GPUs with cache image/checkpoint on node;
FetchStandby(k, r): fetch k GPUs with reliability $\geq r$
Output: *Chosen*: set of *Need* GPUs, or current available GPUs

1 **Step 1 – Load GPU Metadata**;
2 (*RelMap*, *TagMap*) $\leftarrow$ LoadConfigMaps();
3 **Step 2 – Collect Usable GPUs**;
4 *Usable* $\leftarrow \emptyset$;
5 **foreach** $g \in \mathcal{G}$ **do**
6 **if** *RelMap*[g] < *ReqRel* **then**
7 **continue;**
8 **if** $g \notin$ dom(*TagMap*) **then**
9 *Usable* $\leftarrow$ *Usable* $\cup \{g\}$;
10 **else if** *Prio* = *Low* **then**
11 *Usable* $\leftarrow$ *Usable* $\cup \{g\}$;
12 **else if** *Prio* = *High* $\wedge$ Match(*Pod TagMap*[g]) **then**
13 *Usable* $\leftarrow$ *Usable* $\cup \{g\}$;
14 **Step 3 – Allocate with Cache Priority or Supplement**;
15 **if** $|$*Usable*$| \geq$ *Need* **then**
16 *Chosen* $\leftarrow$ PickGPUs(*Usable Need*);
17 **return** *Chosen*;
18 **else**
19 *Shortfall* $\leftarrow$ *Need* $-$ $|$*Usable*$|$;
20 **if** *IsReclaim* **then**
21 *Supplemented* $\leftarrow$ FetchStandby(*Shortfall ReqRel*);
22 *Chosen* $\leftarrow$ *Usable* $\cup$ *Supplemented*;
23 **return** *Chosen*;
24 **else**
25 **return** *Usable*;

Dynamic Tagging-based GPU Manager. WeFlex dynamically classifies GPUs at the granularity of individual devices rather than whole nodes. Each GPU is uniquely identified by the tuple *(nodeIP, gpu_idx, gpu_type)*, capturing the device's location and specific hardware type (e.g., H20). The reliability tier classification, High-Availability (HA, 100% uptime), Medium-Availability (MA, 97–100% uptime), and Low-Availability (LA, below 97% uptime), is refreshed every two weeks, ensuring an accurate reflection of the current hardware condition. When an online training job scales down, WeFlex tags HA and MA GPUs owned by the online training jobs. Each tagged GPU receives a transient ownership identifier corresponding to the releasing online training job, explicitly signaling future reclamation intent. Untagged GPUs remain open for general allocation. If a tagged GPU cannot be reclaimed due to hardware failures or other unexpected issues, WeFlex supplements these with replacement GPUs from the standby pool, matching the required availability tier and GPU type. This ensures the stable proportion of HA devices within online training jobs post elastic scaling. To maintain the standby resource pool's stability during routine allocation and reclamation, GPUs released to the platform are periodically integrated back into standby pool.

Priority-based Pod Scheduling. The priority-based scheduler leverages unified and dynamic metadata from the tagging-based GPU manager to enforce both reliability and ownership constraints within a single, cohesive scheduling operation, as outlined in Algorithm 1. The scheduling process proceeds in several integrated stages: It begins by loading the tagging map along with real-time GPU availability data(line 1). Candidate GPUs are then selected through a process that considers both the reliability tier, ensuring only devices that satisfy the job's availability requirements are included, and ownership constraints. For lower-priority jobs, allocation can utilize both unallocated GPUs and tagged GPUs that are temporarily released by online training jobs, while these tagged GPUs remain subject to immediate reclamation by their original owners if necessary. In contrast, high-priority jobs are only permitted to reclaim tagged GPUs if these devices were previously released by themselves, which ensures the recoverability of reserved resources for critical online training during scale-up(lines 5–13). Subsequently, the scheduler determines the final allocation based on the operational context. When the number of available GPUs exceeds the job's request, preference is given to devices residing on nodes that still cache the job's checkpoint or image data, thereby minimizing restart latency as discussed in Section 3.3 (lines 15–17). If available GPUs are insufficient and the scheduling action is triggered by an online training resource reclamation event, the scheduler supplements the allocation from the standby resource pool, selecting devices that match the required reliability and hardware type. For ordinary resource requests, the scheduler simply returns the set of currently available GPUs. GPUs in the standby pool are reserved exclusively to backfill scale-up reclaim operations when the normally reclaimable devices are insufficient(lines 19–25).

Preemption Handling for LP Tasks. When HP online training scales up, WeFlex performs *graceful reclaim* to bound impact on LP jobs. WeFlex issues a preemption notice with a short grace period τ_{grace} (30–60 s) via the Kubernetes API. During τ_{grace}, LP jobs flush lightweight state and, if supported, persist checkpoints/offsets. After the timeout, only the selected LP pods are terminated and their GPUs reclaimed, preserving device provenance (type and availability tier) for right-of-return. Since LP jobs are offline/non-real-time, brief interruptions are tolerated under QoS policy.

4 EVALUATION

This section presents a comprehensive evaluation of WeFlex's performance in WeChat's production environment. Through the experiments detailed here, we aim to address the following critical

questions: i) To what extent does WeFlex's resource leasing capability, enabled by elastic scaling, improve GPU utilization efficiency? (Section 4.3) ii) Compared to native elastic scaling of K8s, how effectively does WeFlex preserve the stability and robustness of online training workloads? (Section 4.4) iii) What mechanisms enable WeFlex to sustain such stability and robustness during elastic scaling? (Ablation studies in Section 4.5–4.7)

4.1 Experiment Setup

WeFlex is developed with about 50,000 lines of Go language. It has been deployed into production for over one year to work with Kubernetes in internal GPU cluster of Wechat. The production environment comprises 10,000-plus GPUs, including Nvidia H20 GPUs, etc. By 2025, we have used WeFlex to support a wide range of online training jobs, including real-time short video recommendations, searching, and advertisement. More than one billion users are using these services monthly. The number of GPU used in the evaluation varies depending on the specific experiment and service type, which are detailed in the context of each experiment. We also implemented a YARN* baseline on Kubernetes in Section 4.4, Section 4.5 that reproduces YARN's[31] core mechanisms, including priority and fairness preemption, node and rack aware placement, node health awareness, and framework-level checkpoint and restart, while do not include WeFlex-specific features such as interruption mitigation pipeline.

4.2 Workloads Deployed on GPU Cluster

In our production GPU cluster, online training workloads predominantly consist of real-time training of recommendation models[12, 18, 30], for example, training a large-scale ranking model that is routinely deployed in our live environment or training the Wide & Deep model [5] on the Criteo Terabyte Click Logs in chronological order. Other GPU-intensive workloads are also diverse but are chiefly composed of offline training and fine-tuning jobs for various models, such as large language models and other deep learning models. The selections of distributed training frameworks are guided by the unique requirements of each tenant or team, spanning options such as PyTorch and TensorFlow, orchestration platforms like Ray, or MPI-based solutions.

4.3 Improvement of Resource Usage

In this section, we assess WeFlex's impact on resource utilization from two perspectives: first, by measuring uplift in GPU utilization for online training jobs after elastic scaling; and second, by quantifying improvement in resource-request fulfillment ratio for other workloads, demonstrating the GPUs released by online training jobs are being fully leveraged.

4.3.1 Improvement of Cluster-Wide Resource Usage. As shown in Figure 9, we quantify the effectiveness of WeFlex in addressing workload imbalances and enhancing GPU utilization within a large-scale production environment. Our evaluation spans GPU clusters comprising approximately 10,000 GPUs, with cloud-native online training jobs occupying roughly 48% of these GPUs, while other GPU-intensive workloads (e.g., offline LLM inference and fine-tuning tasks) utilize the remaining resources. We collected GPU utilization data separately over a 24-hour period, comparing scenarios with and without WeFlex integration.

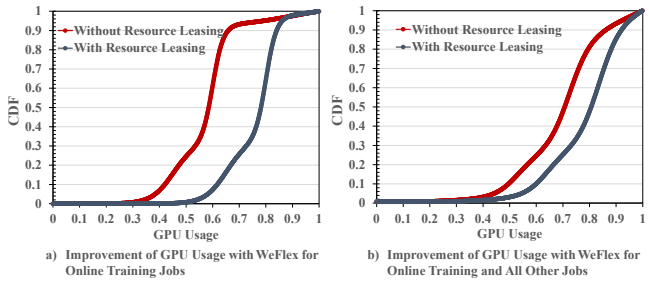

Figure 9: Improvement GPU usage for cloud-native online training jobs and all other GPU-intensive jobs in WeChat by employing resource leasing enabled by elastic scaling.

Figure 9(a) breaks down the utilization of GPUs serving online training jobs prior to any resource reallocation. Nearly 30% of these GPUs ran at low efficiency (utilization between 0.30 and 0.55), 65% operated in the mid-range (0.55–0.80), and only 5% exceeded 0.80. After enabling WeFlex's resource-leasing mechanism, we observe a roughly 25% uplift in average GPU utilization for online training, over 30% of those GPUs now reach or exceed 80% utilization. Figure 9(b) shows the aggregate utilization across all workloads including both online training and other workloads: since other GPU-intensive generally sustain higher GPU occupancy, the cluster-wide CDF shifts upward, about 15% of GPUs fall in the 0.30–0.55 band, 65% in the 0.55–0.80 band, and the remaining 20% above 0.80. After enabling WeFlex's resource-leasing mechanism, overall cluster utilization climbs by about 10%, demonstrating that GPUs vacated during training troughs were promptly reallocated to other GPU-intensive workloads rather than left idle. This confirms that WeFlex both levels out utilization across workloads and maximizes efficiency of the entire GPU fleet.

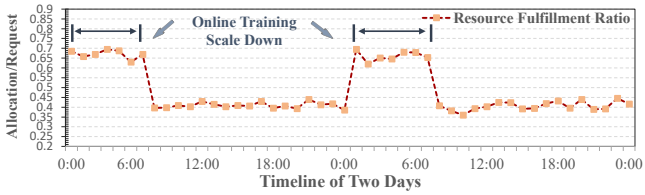

Figure 10: Resource fulfillment ratio over a 48-hour period for all submitted jobs on the cloud platform with resource leasing of online training jobs.

4.3.2 Improvement of Cluster-Wide Resource Request Fulfillment Ratio. Figure 10 presents a 48-hour record across all the jobs that lease GPUs from online training workloads, in which we captured every job's GPU requests and quantified the corresponding allocations granted. It shows that GPU resources are in short supply: for all these tenants on the cloud platform, the average fulfillment rate for GPU requests lingers at only about 40%. In other words, if the tenant requests 100 GPUs, only roughly 40 are granted immediately.

During the trough period for online training jobs, typically from 00:00 to 06:00, training traffic falls and many GPUs remain idle. WeFlex therefore elastically scales down those training jobs and leases the freed GPUs to other workloads. As a result, over the 00:00–06:00 window, request-to-allocation rate for other GPU-intensive jobs jumps by roughly 25 percentage points, to around 65%, alleviating resource shortages and improving utilization of GPU fleet.

4.4 Evaluations on Stability Preservation

In this section, we validate WeFlex's effectiveness in maintaining training stability during elastic scaling through two complementary analyses. First, we dissect the breakdown of GPU hours over time to quantify the duration of continuous training intervals unaffected by scaling operations. Second, we assess the tenant perspective by examining critical service metrics to confirm that scaling actions introduce no adverse effects on application performance or model quality. To empirically substantiate these findings, we conducted small-scale experiments on WeChat's video recommendation service using a cluster of 50 servers, each equipped with eight H20 GPUs. Each scale-down operation releases half of the GPUs, while a scale-up operation reclaims all of them.

4.4.1 GPU Hour Breakdown of Online Training. The breakdown of GPU hours is shown in Figure 11. This evaluation spans 10 days, amounting to approximately 4,000 GPU hours (50 servers × 8 GPUs/server × 10 days). GPU idle time is calculated based on utilization: for instance, if training runs for 5 hours at 60% average GPU utilization, we attribute 3 hours to active training and 2 hours to idle time. Interruption time captures training pauses due to scaling events (both up and down) or user-initiated job restarts. GPU downtime specifically denotes periods when GPUs become unavailable due to hardware issues, including overheating, device failures, or NVLink disruptions. For the online training workload of this service, tidal fluctuations in demand drive an average GPU utilization of 61%. Accordingly, without any elastic scaling, normal training occupies just 58% of total GPU time, while idle periods account for roughly 37%. When we introduce Kubernetes' native scale-up/scale-down mechanism, overall utilization does improve, but at a cost: each daily scaling event forces a job restart, producing interruption periods that consume nearly 9% of GPU time. Moreover, because high-availability GPUs released during scale-down are not always reclaimed, GPU downtime creeps up by about 1.5%. Even so, the dramatic drop in idle time delivers an around 13% increase in effective training time. While YARN* also reduces idle time, its reliance on checkpoint/restart forces a full restart path, leaving interruption time still high. Its node-health checks slightly lower GPU downtime by avoiding faulty nodes, but without device-level right of return, HA GPUs are not preserved across scaling cycles. By contrast, integrating WeFlex to orchestrate elastic scaling preserves nearly all of that utilization gain while dramatically enhancing stability. Interruption time grows by only 1% over the no-scaling baseline, and GPU downtime remains essentially flat. As a result, "normal" training time soars to 81% of total GPU hours, an improvement of 23% versus no scaling and 9% over native scaling, demonstrating that WeFlex can reconcile high utilization with minimal disruption.

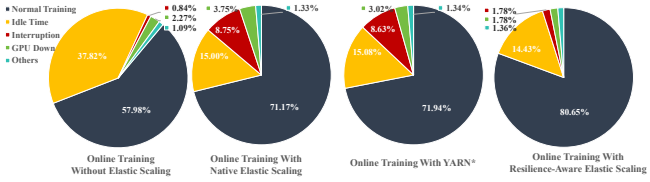

Figure 11: GPU hour breakdown for online training jobs.

4.4.2 Impact on Service Quality and Accuracy of Online Training Model. As shown in Figure 12, we diverted a small portion of traffic from WeChat Video's recommendation service to an online training workload employing elastic scaling, aiming to evaluate the impact of scaling operations on model performance and service quality. The baseline was an online training model without elastic scaling. We primarily assessed three key metrics: Overall Click-Through Rate (CTR), indicating content attractiveness; User CTR(U-CTR), reflecting individual user engagement; and Conversion Rate (CR), measuring effectiveness of user behavioral guidance. Under native elastic scaling and YARN*, frequent job restarts during scaling-up and scaling-down periods led to interruptions lasting approximately 30 minutes each time. These disruptions prevented real-time training on fresh user data, resulting in significant degradations, U-CTR dropped by roughly 5%, CTR by 7.5%, and CR by 5.5%. Compared with naive scaling, YARN* shows no substantial improvement, as slow checkpoint recovery and lack of GPU reusability remain unsolved. In contrast, when employing WeFlex's elastic scaling solution, its interruption mitigation pipeline substantially shortened training interruptions brought by elastic scaling. Consequently, the adverse impact on service metrics is negligible, showing minimal deviation from the non-scaling baseline. This highlights WeFlex's capability to maintain robust service quality and model performance even under dynamic scaling conditions.

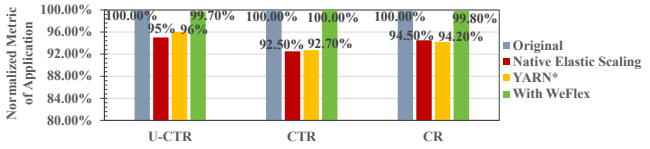

Figure 12: Impact on service metrics and model quality when employing elastic scaling for online training.

4.5 Reduction of Interruption Duration

To evaluate the reduction in interruption duration, we monitored the streaming training data throughput on WeChat's video recommendation service for 12 hours. The experiment was conducted on a cluster of 64 servers, each equipped with eight H20 GPUs. During each scale-down, half of the GPUs were released, and during each scale-up, all loaned GPUs were reclaimed. As illustrated in Figure 13, without elastic scaling (red line), the online training workload exhibits predictable tidal fluctuations while the data ingestion rate remains stable. Under native elastic scaling (blue line) and YARN* (green line), with a scale-down at 1:30 and a scale-up at 6:30, the input throughput collapses to nearly zero for roughly 30 minutes each time, halting the streaming training process. YARN* still follows a full restart path, leading to long interruption windows. In

practice its behavior is close to the naive baseline. By contrast, when employing the interruption-mitigation pipeline (orange line), each scaling event induces only a brief disruption of a few tens of seconds, during which incoming data can be buffered in message queue. Once the restart completes, the accumulated backlog is replayed, producing a short burst in the input rate during recovery. This approach dramatically reduces the disturbance of elastic scaling on online training and underscores the interruption-mitigation pipeline's role in preserving continuous service.

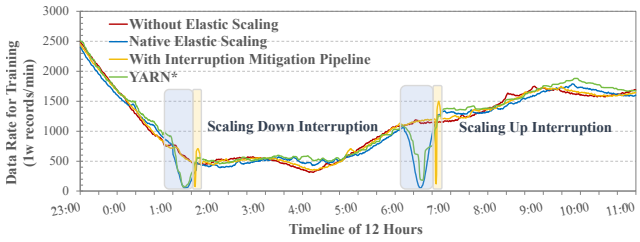

Figure 13: Reducion of interruption duration.

4.6 Effect of Topology-aware Instance Orchestration

To evaluate the impact of topology-aware job orchestration, we conducted a command-execution test of the most widely used distributed training frameworks, such as MPI-based solution, PyTorch Distributed Libray, Ray, and TensorFlow Distributed Libray, on 64 GPU servers (8 H20 GPUs per node) as shown in Table 1.

Table 1: Failure rate of command issued from cloud platform.

Distributed Framework	Command	Original Failure Rate	Topology-aware Failure Rate
MPI	mpirun	95%	0%
Ray	ray start	92%	0%
PyTorch Distributed	torchrun	97%	0%
Tensorflow Distributed	start script	0%	0%

Issuing launch commands directly from the cloud management plane is inherently blind to the actual state of each worker pod: if an MPI *mpirun* command or PyTorch/Ray launcher is invoked before all remote pods are up and registered, the job invariably fails. In our measurements, over 90% of attempts with MPI-based frameworks, PyTorch Distributed, and Ray were unsuccessful when driven naively by the cloud platform. TensorFlow Distributed embeds its own readiness checks and coordination logic, so even commands dispatched from the cloud plane consistently succeed. WeFlex's topology-aware instance orchestration performs comprehensive preflight validation, verifying pod readiness and resource availability, before issuing any launch command. With these guarantees in place, every cloud-initiated command across all four frameworks was completed successfully, demonstrating that topology-aware orchestration eliminates the unpredictability of command execution.

4.7 Effect of Right-of-Return GPU Leasing

To validate the effectiveness of the Right-of-Return GPU leasing policy, we conducted a two-month production trace of the streaming training workload for WeChat's video recommendation service on a 64-server cluster, each provisioned with eight H20 GPUs. We tracked the share of high-availability GPUs assigned to the job under three configurations: no elastic scaling, GPU leasing without Right-of-Return, and GPU leasing with Right-of-Return. As shown in Figure 14, without elastic scaling, the job maintained a 74% share of high availability GPUs, 4 percentage points above the average for two weeks of the cluster shown in Figure 5. When leasing GPUs without a Right-of-Return guarantee, the share of high-availability GPUs gradually diminished until it aligned with the cluster average, failing to meet the tenant's reliable reclamation requirements. In contrast, with Right-of-Return leasing policy enforced, the high-availability share remained firmly at 74% throughout the trace, demonstrating that the Right-of-Return GPU leasing mechanism consistently preserves access to a stable portion of reliable GPU resources with elastic scaling.

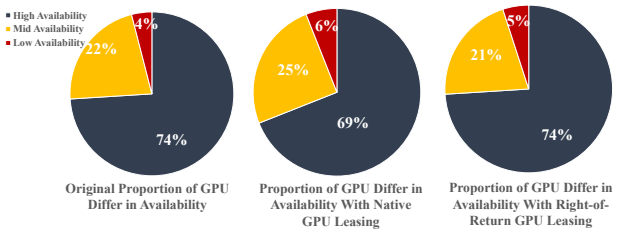

Figure 14: Production trace in WeChat for ratio of GPUs differ in availability.

4.8 Portability and Limitations of WeFlex

Although WeFlex is deployed within WeChat's ecosystem, its underlying techniques are broadly applicable to key challenges in large-scale online services, such as mitigating disruptions caused by scaling and enabling topology-aware instance orchestration. Our solution is built upon widely adopted de facto frameworks Kubernetes, ensuring seamless adaptability across diverse infrastructure environments. Moreover, the framework operates at the cloud platform layer for resource provisioning, making it agnostic to upper-layer deep learning training frameworks. This allows WeFlex to support distributed training across various DL frameworks without imposing restrictions on the application layer, further enhancing its flexibility and broad applicability.

WeFlex still offers considerable potential for further refinement. First, although current analyses show that the remaining short-lived interruptions exert minimal impact on overall service quality, and most online training tenants accept this trade-off in exchange for significantly improved GPU utilization and the economic gains from leasing idle GPUs. These brief stalls have not yet been fully eliminated, leaving room for further optimization. Second, to guarantee sufficient capacity for timely resource reclamation, WeFlex maintains an idle standby resource pool. The capacity of this pool is currently determined by empirical estimation, typically encompassing several dozen servers of a variety of types and dynamically adjusted as necessary. Moving forward, WeFlex will incorporate more detailed statistics on unreclaimed devices observed during each scale-up event, enabling more accurate prediction and adaptive adjustment of standby pool requirements.

5 RELATED WORK

DL-Framework coupling elastic scaling. Distributed DL frameworks such as Elastic Horovod [9], PyTorch Elastic [22] and ElasticDL [35] provide built-in mechanisms for elastic scaling within individual training jobs. AntMan[33] deeply co-designs cluster schedulers and DL frameworks to address GPU sharing challenges. It offers fine-grained scaling by micromanaging computation and GPU memory during training. However, these framework-specific implementations typically restrict elasticity to single-framework scenarios and lack effective support for multi-tenant GPU environments as depicted in Figure 1. The core limitation is that different business and algorithm teams often adopt their customized distributed training frameworks. Achieving elasticity at the DL framework level would require extensive and impractical modifications across all frameworks. Therefore, providing elasticity at the platform level, rather than relying on individual framework adaptations, is essential for enabling efficient resource lending in multi-tenant production clusters.

Resource platform support for elastic scaling. Existing solutions such as Lyra [13], Primus [2], and native Kubernetes offer elastic scaling mechanisms, primarily targeting stateless services like inference tasks, whose replicas are independent and easily scaled. [19] proposes an auto-scaling policy by considering both cost and scaling efficiency. These approaches, however, do not explicitly address the challenges posed by online training workloads, particularly the stringent stability requirements. Online training jobs face substantially different challenges during resource scaling, including prolonged interruptions, complex state synchronization, and the risk of significant streaming data loss. To the best of our knowledge, WeFlex is the first platform-layer solution that explicitly addresses elastic scaling for online DL training in multi-tenant production environments.

Dynamic resource allocation on GPU cluster. Existing approaches extensively explore dynamic GPU allocation and scheduling policies to improve overall resource utilization and reduce average job completion time (JCT). Tiresias [8] leverages a least-attained-service approach to minimize average JCT, though it does not explicitly support elastic scaling. Optimus [21] predicts training durations by modeling loss convergence and employs heuristics to minimize average JCT. Graphene [6] dynamically adjusts resource allocations to match time-varying job demands, aiming to enhance cluster utilization. Synergy[16] performs multi-resource workload-aware assignments across a set of jobs scheduled on multi-tenant clusters. However, these methods primarily focus on scheduling optimization rather than true elastic resource scaling in multi-tenant GPU clusters. Specifically, they do not support scenarios involving resource lending or resource reclaiming among tenants or workloads characterized by pronounced tidal patterns. In contrast, WeFlex explicitly addresses these gaps by enabling elastic, resilience-aware resource sharing tailored specifically for continuous online DL training workloads.

Efforts of SLO Guarantee for GPU Sharing. Prior works such as Mudi[3], FaPES[34], and AntMan[33] have also considered SLO-related issues, but they primarily focus on mitigating interference among multiple workloads sharing the same GPU device. ElasticFlow[7] is an elastic serverless computing platform for distributed training. It performs admission control based on a minimum satisfactory share to guarantee deadlines. In contrast, WeFlex specifically targets distinct challenges arising during the scaling of online training workloads in production, such as scaling-induced interruptions and reliable reclamation of high-availability GPUs. Thus, these prior works are orthogonal to WeFlex and address fundamentally different problems.

Elastic scaling in data management systems. Modern cloud-native data management systems such as distributed databases, stream processing engines, and real-time analytics platforms also face challenges analogous to those in online deep learning training. Systems like Oceanus [4], StreamOps [15], and MagicScaler [20] report temporal and spatial workload imbalances, stringent SLO requirements, and complex inter-component dependencies during scaling. While existing elasticity mechanisms in these systems focus on CPU and memory provisioning, the underlying principles (predictive scaling, fine-grained resource allocation, dependency-aware orchestration) are directly relevant to GPU-intensive workloads. WeFlex extends these concepts to the GPU domain, introducing device-level right-of-return leasing, topology-aware orchestration across distributed training frameworks, and interruption mitigation pipelines. These techniques can be applied to a broader class of data-intensive systems to enable stable, SLO-preserving elasticity in multi-tenant environments.

6 CONCLUSION

We have presented WeFlex, a resilience-aware resource leasing system that enables elastic scaling for cloud-native online training in multi-tenant GPU clusters. The key idea is to exploit cluster-wide elasticity, by loaning idle GPU capacity to other workloads during off-peak hours, and job-level elasticity, by preserving the stability of online training jobs across scale-up and scale-down events. WeFlex addresses three core challenges of production online training: prolonged training interruptions, dependency-unaware orchestration, and unreliable GPU reclamation. To meet these challenges, we introduced an interruption-mitigation pipeline, a topology-aware pod orchestrator, and a right-of-return GPU leasing mechanism that ensures reliable and latency-bounded resource reclamation for high-priority jobs.

Our evaluation on production clusters with over 10,000 GPUs demonstrates that WeFlex not only improves the GPU clusters utilization by about 25%, but also maintains stable convergence and introduces no additional downtime or failure rate during dynamic scaling. Meanwhile, the right-of-return mechanism enhances the resource-fulfillment ratio for other GPU-intensive workloads by around 25% during the scale-down phases of online training jobs. Overall, WeFlex shows that carefully designed elastic scaling mechanisms can deliver substantial efficiency gains while preserving robustness required for online training at scale, offering a practical path toward more agile and resource-efficient cloud-native GPU clusters in production.

ACKNOWLEDGMENTS

Thanks to anonymous reviewers for the feedback. This research is supported by the Start-up Fund of Beijing Normal University (NO.28700-312200502503).

REFERENCES

[1] Apache. 2025. Pulsar. https://pulsar.apache.org/

[2] ByteDance. 2023. Primus. https://github.com/bytedance/primus

[3] Wenyan Chen, Chengzhi Lu, Huanle Xu, Kejiang Ye, and Chengzhong Xu. 2025. Multiplexing Dynamic Deep Learning Workloads with SLO-awareness in GPU Clusters. In *Proceedings of the Twentieth European Conference on Computer Systems*. 589–604.

[4] Zihao Chen, Jiazhi Jiang, Jiangang Liu, Chao Zhang, Yuqi Diao, Yang Li, Hanmei Luo, and Peng Chen. 2025. Oceanus: Enable SLO-Aware Vertical Autoscaling for Cloud-Native Streaming Services in Tencent. In *Companion of the 2025 International Conference on Management of Data*. 323–335.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[6] Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. 2016. GRAPHENE: Packing and Dependency-Aware scheduling for Data-Parallel clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 81–97.

[7] Diandian Gu, Yihao Zhao, Yinmin Zhong, Yifan Xiong, Zhenhua Han, Peng Cheng, Fan Yang, Gang Huang, Xin Jin, and Xuanzhe Liu. 2023. Elasticflow: An elastic serverless training platform for distributed deep learning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 266–280.

[8] Juncheng Gu, Mosharaf Chowdhury, Kang G Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. 2019. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 485–500.

[9] Horovod. 2021. Elastic Horovod. https://horovod.readthedocs.io/en/latest/elastic_include.html

[10] Beomyeol Jeon, Chen Wang, Diana Arroyo, Alaa Youssef, and Indranil Gupta. 2025. A House United Within Itself: SLO-Awareness for On-Premises Containerized ML Inference Clusters via Faro. In *Proceedings of the Twentieth European Conference on Computer Systems*. 524–540.

[11] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU clusters for DNN training workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 947–960.

[12] Jiazhi Jiang, Rui Tian, Jiangsu Du, Dan Huang, and Yutong Lu. 2023. MixRec: Orchestrating Concurrent Recommendation Model Training on CPU-GPU platform. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 366–374.

[13] Jiamin Li, Hong Xu, Yibo Zhu, Zherui Liu, Chuanxiong Guo, and Cong Wang. 2023. Lyra: Elastic scheduling for deep learning clusters. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 835–850.

[14] Marko Luksa. 2017. *Kubernetes in action*. Simon and Schuster.

[15] Yancan Mao, Zhanghao Chen, Yifan Zhang, Meng Wang, Yong Fang, Guanghui Zhang, Rui Shi, and Richard TB Ma. 2023. StreamOps: Cloud-native runtime management for streaming services in bytedance. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3501–3514.

[16] Jayashree Mohan, Amar Phanishayee, Janardhan Kulkarni, and Vijay Chidambaram. 2022. Looking beyond GPUs for DNN scheduling on Multi-Tenant clusters. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 579–596.

[17] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging AI applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*. 561–577.

[18] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[19] Andrew Or, Haoyu Zhang, and Michael Freedman. 2020. Resource elasticity in distributed deep learning. *Proceedings of Machine Learning and Systems* 2 (2020), 400–411.

[20] Zhicheng Pan, Yihang Wang, Yingying Zhang, Sean Bin Yang, Yunyao Cheng, Peng Chen, Chenjuan Guo, Qingsong Wen, Xiduo Tian, Yunliang Dou, et al. 2023. Magicscaler: Uncertainty-aware, predictive autoscaling. *Proceedings of the VLDB Endowment* 16, 12 (2023), 3808–3821.

[21] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the Thirteenth EuroSys Conference*. 1–14.

[22] PyTorch. 2021. PyTorch Elastic. https://pytorch.org/elastic/0.2.0rc1/distributed.html#module-torchelastic.distributed.launch.

[23] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. 2021. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*.

[24] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training. (2018).

[25] Doyen Sahoo, Quang Pham, Jing Lu, and Steven CH Hoi. 2017. Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705* (2017).

[26] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 322–337.

[27] Chijun Sima, Yao Fu, Man-Kit Sit, Liyi Guo, Xuri Gong, Feng Lin, Junyu Wu, Yongsheng Li, Haidong Rong, Pierre-Louis Aublin, et al. 2022. Ekko: A Large-Scale deep learning recommender system with Low-Latency model update. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 821–839.

[28] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification?. In *China national conference on Chinese computational linguistics*. Springer, 194–206.

[29] Chunqiang Tang, Kenny Yu, Kaushik Veeraraghavan, Jonathan Kaldor, Scott Michelson, Thawan Kooburat, Aravind Anbudurai, Matthew Clark, Kabir Gogia, Long Cheng, et al. 2020. Twine: A unified cluster management system for shared infrastructure. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 787–803.

[30] Rui Tian, Jiazhi Jiang, Jiangsu Du, Dan Huang, and Yutong Lu. 2024. Sophisticated Orchestrating Concurrent DLRM Training on CPU/GPU Platform. *IEEE Transactions on Parallel and Distributed Systems* (2024).

[31] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 1–16.

[32] Wencong Xiao, Romil Bhardwaj, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kwatra, Zhenhua Han, Pratyush Patel, Xuan Peng, Hanyu Zhao, Quanlu Zhang, et al. 2018. Gandiva: Introspective cluster scheduling for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 595–610.

[33] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. AntMan: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 533–548.

[34] Xiaoyang Zhao, Siran Yang, Jiamang Wang, Lansong Diao, Lin Qu, and Chuan Wu. 2024. FaPES: Enabling Efficient Elastic Scaling for Serverless Machine Learning Platforms. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 443–459.

[35] Jun Zhou, Ke Zhang, Feng Zhu, Qitao Shi, Wenjing Fang, Lin Wang, and Yi Wang. 2023. ElasticDL: A Kubernetes-native deep learning framework with fault-tolerance and elastic scheduling. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*. 1148–1151.