A Design of Cost-Efficient GPU-based Training Workloads on Preemptible Instances via Adaptive Live Migration

Huan Le, Phuong Bac Ta, Vitumbiko Mafeni, and Younghan Kim*
Department of Electronic Engineering, *Soongsil University*, Seoul, Korea
{huanle, bactp, vitumafeni}@dcn.ssu.ac.kr, younghak@ssu.ac.kr

***Abstract*—GPU-based training workloads are increasingly deployed on cloud infrastructure platforms due to their scalability and resource elasticity. To reduce costs, many users leverage preemptible (spot) GPU instances, which are significantly cheaper than on-demand resources. However, spot instances do not provide native mechanisms to checkpoint or restore GPU workloads upon preemption. Consequently, ongoing training processes are abruptly terminated, resulting in the loss of intermediate model states and wasted computation. This limitation significantly hinders the use of spot instances for long-running, stateful AI workloads. This paper proposes an adaptive live migration framework to ensure the continuity of GPU-based training workloads on preemptible instances. To address the lack of native checkpointing in spot instances, the framework leverages CRIUgpu to consistently capture and restore GPU execution state. In addition, it incorporates an adaptive job replication strategy across multiple preemptible instances, proactively coordinating checkpointing and migration to maximize workload continuity under instance preemptions. Initial evaluation results suggest that the approach can achieve substantial cost savings compared to on-demand GPU instances while preserving training continuity and reducing interruptions caused by preemptions.**

***Index Terms*—Spot Instance, AI Training, Multi-cloud, Cloud Computing**

I. Introduction

The rapid expansion of deep learning has increasingly shifted model training to cloud platforms, where elastic resource pools and pay-as-you-go pricing facilitate large-scale experimentation. Among available options, preemptible (spot) GPUs are especially attractive: empirical evidence (e.g., SpotLake) shows typical prices at 8–50% of on-demand [1], enabling more runs or larger models within fixed budgets. However, these savings come with volatility: spot capacity may be reclaimed at short notice, and current platforms lack GPU-native checkpoint/restore for user workloads. Consequently, long-running training jobs are vulnerable to abrupt termination, which discards in-flight GPU state, erases progress, and undermines the very cost benefits that motivate spot adoption.

This paper examines that core tension—cost efficiency vs. reliability—for training workloads that retain substantial state in device memory and driver contexts. Existing mitigations are insufficient. Framework-level periodic checkpointing operates at coarse granularity, incurs I/O and synchronization overheads, and still loses work since the last save. Elastic/asynchronous training can tolerate worker churn but typically requires non-trivial code changes and still restarts from the last durable checkpoint, without preserving in-flight GPU execution (e.g., kernel queues, CUDA streams). Cloud interruption signals are provider-specific and often too brief for safe capture, limiting portability. What is missing is a transparent, GPU-aware migration mechanism that allows unmodified training workloads to survive preemptions with minimal loss of progress.

*Corresponding Author: Younghan Kim (younghak@ssu.ac.kr)

We address this gap with a Kubernetes-native framework for cost-efficient training on preemptible instances via adaptive live migration. At its core is a preemptible-aware controller (a custom K8s operator) that manages a spot node pool and orchestrates GPU training pods. The controller is reactive: it continuously watches termination notices and cluster health; upon detecting a preemption, it immediately binds a replacement pod to another running instance (or a small on-demand fallback, if configured) and restores the training container from the most recent checkpoint. We leverage the container runtime's native CRIU integration [2] together with CRIUgpu plugin [3] to checkpoint and restore the container that hosts the trainer, capturing CUDA contexts, device memory, and driver state coherently with host process state. Checkpoint artifacts are persisted to shared storage and used to resume the training pod, thereby preserving continuity while retaining spot-level costs.

We evaluate the framework on representative training tasks under realistic spot churn. Compared to on-demand, our system preserves the majority of spot–cost savings while maintaining continuity under preemptions. Compared to a periodic-only checkpoint baseline, it shortens time-to-finish by resuming more quickly after revocations and losing less training progress, at the expense of a small additional cost from maintaining a second preemptible instance. This paper makes two contributions:

- **GPU-aware live migration for training jobs:** We enable live migration *without losing job progress* by integrating CRIU with CRIUgpu plugin to checkpoint and restore the training container, capturing GPU/device state coherently with host process state.
- **Adaptive replication and migration policy:** Rather than predicting revocations, the system reacts to a detected preemption by using another running instance to spawn

a replica (or promote a sibling) and continue from the latest checkpoint.

The remainder of this paper is organized as follows: Section II reviews related work; Section III details the proposed system; Section IV presents preliminary experiments and evaluation results; and Section V outlines future directions and concludes.

II. Related Work

This section reviews prior studies most relevant to our goal—cost-efficient, reliable training on preemptible GPUs—emphasizing their objectives, system levels, and differences from our approach.

Inference and service continuity on spot. *SkyServe* [4] targets *inference* across on-demand and spot GPUs, optimizing latency and cost to satisfy service SLOs rather than preserving long-running training state. *Tributary* and *Cocktail* [5], [6] likewise address elastic services/model serving, focusing on SLO adherence and throughput under transient capacity. These systems improve serving-time robustness but do not maintain device-resident training state. Our focus is *training* continuity with explicit preservation of GPU contexts.

Training on transient resources. *Varuna* [7] scales massive models on commodity/spot VMs via parallelism strategies and *elastic job morphing*, adapting to bandwidth and capacity variability through reconfiguration. *Bamboo* [8] improves fault tolerance for pipeline-parallel training by injecting *redundant computation* into pipeline bubbles so neighboring stages can continue during worker loss. *Spotnik* [9] advocates designing distributed ML for transient resources, proposing abstractions and schedulers that tolerate revocations using lightweight checkpointing and reconfiguration on ephemeral workers. Collectively, these operate at the *training runtime/algorithmic* layer—trading reconfiguration or redundancy for resilience. In contrast, we provide *process-level, GPU-aware live migration*: we checkpoint and restore CUDA/device state coherently with host state and *reactively* resume on another running instance after a detected preemption, enabling continuity without model or pipeline changes.

Infrastructure-level cost–availability policies. *SNAPE* [10], [11] models the mix of spot and on-demand VMs to achieve high availability at lower cost using production traces and constrained RL. This line of work is workload-agnostic and focuses on VM orchestration rather than preserving application-level GPU state. Our approach is complementary: it can sit atop such provisioning policies to further reduce lost work when revocations occur by migrating containerized training state.

III. System Description

The system (Fig. 1) runs inside a Kubernetes (K8s) cluster and provides reactive continuity for GPU training workloads on preemptible (spot) instances by combining a control-plane controller with node-local container checkpoint/restore using CRIU and CRIUgpu plugin. The SpotInstance Controller is a leader-elected controller that continuously reconciles GPU jobs against live cluster state and cloud revocation feeds. The GPU-Job Agent executes on each GPU spot node and interacts with the node's container runtime to issue checkpoint and restore operations. Checkpoint artifacts are written to a shared object store; artifacts are immutable, content-addressed, and versioned by (job_id,replica_id,t). The migration boundary is the entire training container, including process tree, memory image, file-descriptor table, and GPU contexts/driver state captured by CRIUgpu.

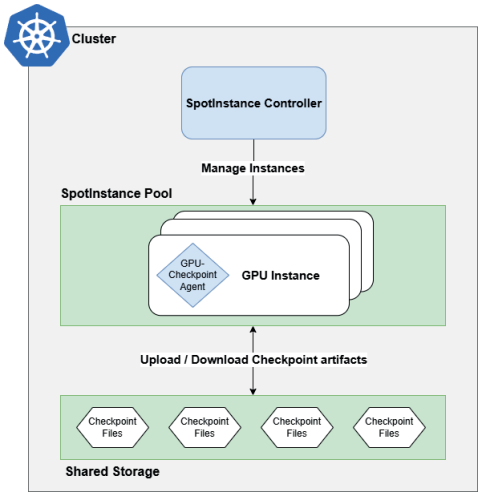


Fig. 1: Overview of the high-level architecture. The SpotInstance Controller manages the lifecycle of instances within the SpotInstance Pool. Each instance utilizes a local GPU-checkpoint agent to save artifacts to the Shared Storage backend.

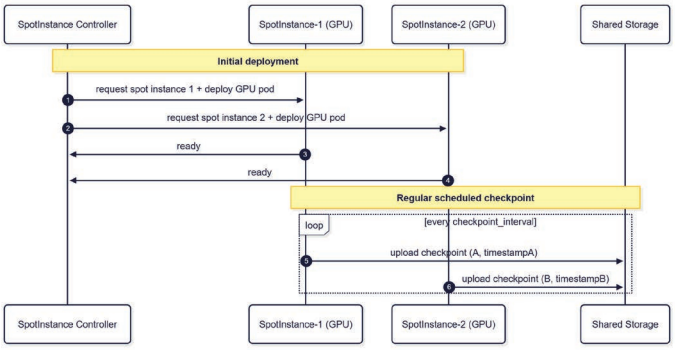


Fig. 2: Periodic checkpointing timeline.

In steady state, the controller enforces a periodic checkpoint cadence (Fig. 2). At each interval, it instructs the GPU-Job Agent on every active replica to invoke the container runtime's CRIU/CRIUgpu pipeline to checkpoint the training container. The resulting artifacts, along with integrity metadata (hash, size, timestamp), are uploaded to shared storage. The controller tracks per-replica placement and the last successful checkpoint, maintaining the freshest consistent recovery point for subsequent restores.

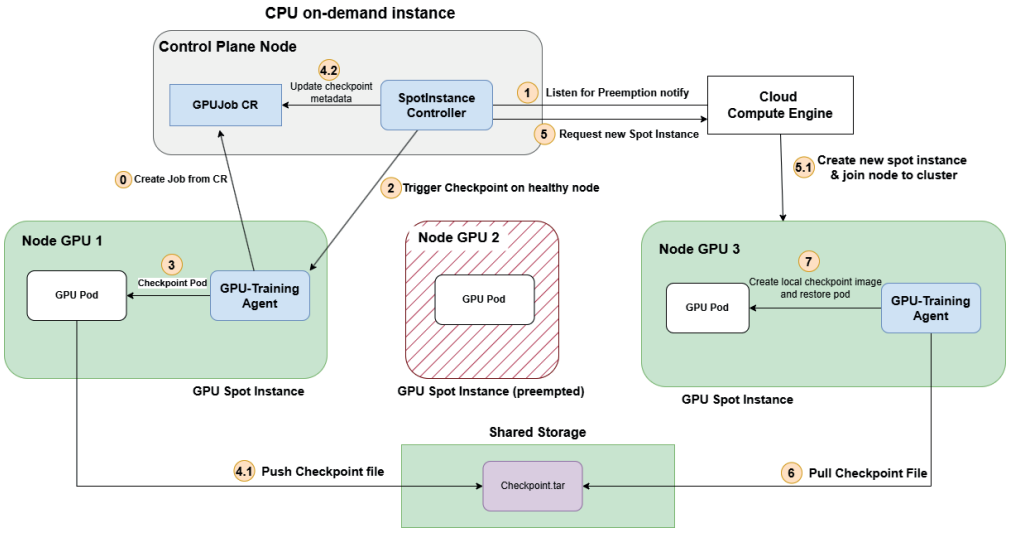


Fig. 3: System architecture and recovery workflow.

As shown in Fig. 1, when a preemption is detected—via cloud termination/eviction notices, node `NotReady`/`Unknown` conditions, or pod loss—the controller reacts immediately. If a peer replica remains available, it first tightens the recovery point by triggering an on-demand checkpoint on the survivor, then allocates a replacement spot node (or a small on-demand fallback if configured), schedules a new pod, and directs the target node's agent to fetch the freshest artifact and execute CRIU/CRIUgpu restore. If multiple replicas are revoked nearly simultaneously, the controller provisions replacements and restores from the latest completed periodic checkpoint. By preserving device-resident state and re-homing containers rather than recomputing work, the design maintains training throughput on volatile spot pools while bounding both lost work and recovery latency.

IV. Preliminary Experiment and Results

A. Spot Instance Setup

TABLE I: Per-hour GPU prices (USD) used.

GPU Model	On-Demand Instance ($ / h)	Spot Instance ($ / h)
P100	1.29	0.22
V100	1.91	0.65

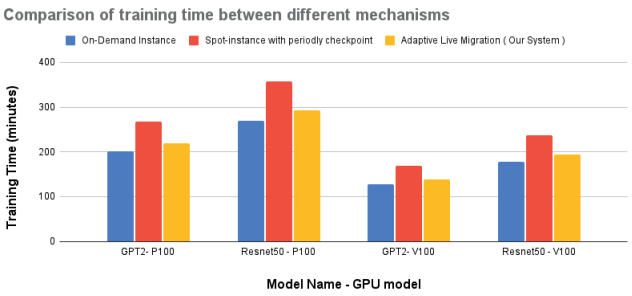
We evaluate the system on Google Cloud Platform (GCP) in `asia-northeast3` (Seoul) using preemptible GPU nodes provisioned with NVIDIA V100 and P100 accelerators; the on-demand and spot prices used in our cost calculations are summarized in Table I. We compare two baselines against our approach: (i) a periodic-only checkpoint baseline that runs on a single preemptible GPU instance and saves framework-level checkpoints at a fixed interval (no GPU-state migration), and (ii) an on-demand baseline that runs on a single on-demand GPU instance (no preemptions). Our system executes the same workload on two preemptible GPU instances, enabling reactive recovery by restoring the container (including GPU state) on surviving or newly provisioned capacity. All variants use identical container images, CUDA/cuDNN stacks, and training hyperparameters. Checkpoints for our system are created at a fixed interval t and stored in shared object storage mounted to the cluster; the periodic-only baseline uses the same t at the framework level for comparability.

We consider two workloads. First, GPT-2 [12] finetuning on the US Financial News Articles dataset (Kaggle) [13], trained for 5 epochs. Second, ResNet-50 [14] on ILSVRC2012 (ImageNet) [15], trained for 5 epochs with batch_size = 128. Data pipelines, augmentations, and optimizer settings are held constant across baselines and our system. Unless otherwise stated, results are reported per GPU family (V100/P100) and averaged over repeated trials.

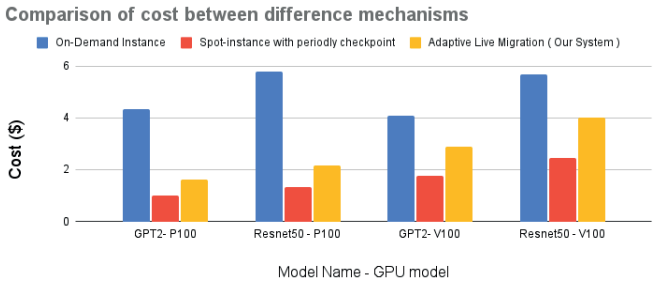
B. Experiment and Preliminary Results

For each workload (GPT-2, ResNet-50) and GPU type (V100, P100), we run the three configurations end-to-end and measure (i) Time-to-Finish (wall-clock from the first training step to completion of the target epochs, including any preemption handling) and (ii) Cost, computed as VM billable time multiplied by the corresponding price class (spot/on-demand) and aggregated over all instances used. To exercise failure handling on preemptible nodes, we inject revocations during training. In the periodic-only baseline, revocation forces a restart from the last framework checkpoint. In our system, revocation triggers a reactive container restore (including GPU state) on a healthy or replacement node; if a peer replica is alive, we first issue an on-demand checkpoint to minimize progress loss. The on-demand baseline provides a no-preemption reference for time but represents the highest cost class.

As shown in Fig. 4a, the periodic-only spot baseline incurs substantial recomputation after revocations and therefore lengthens completion time by +32.8% on average relative to on-demand. In contrast, Adaptive Live Migration resumes



(a) Training time.



(b) Training Cost.

Fig. 4: Evaluation result.

from a near-current container snapshot that includes GPU state, reducing replay and avoiding CUDA cold-start overheads; it finishes 18.1% faster than periodic-only on average and remains within +8.7% of on-demand time across models and GPUs. Dual-revocation trials narrow the gap because both spot configurations fall back to the latest periodic checkpoint, yet container-level restore still shortens wall-clock by avoiding framework re-initialization.

Cost results (Fig. 4b) mirror this trade-off. Despite running two spot instances, Adaptive Live Migration retains most of the monetary advantage of spot, averaging 0.54× the on-demand cost (i.e., 46.2% savings). Relative to the single-node periodic-only baseline, our configuration is 1.64× more expensive on average—an expected premium for redundancy—but delivers a systematic reduction in time-to-finish (-18.1%), yielding better time-cost efficiency under realistic churn. Overall, the data indicate that GPU-aware container restore is an effective mechanism to bound interruption overheads while operating in the low-price regime of preemptible capacity.

V. Conclusion and Future Directions

In this study, We presented a Kubernetes-native framework for running GPU training on preemptible instances using a preemptible-aware controller and container-level checkpoint/restore (CRIU + CRIUgpu). In experiments on GPT-2 and ResNet-50 with P100/V100 GPUs, the system consistently reduced time-to-finish versus a periodic-only spot baseline (18% on average) while retaining most of the cost advantage over on-demand runs, at the expense of a modest premium relative to single-node spot.

Future work will extend the design to federated multi-GPU training jobs and incorporate a predictive preemption subsystem to pre-stage minimal standby capacity, further shrinking recovery time without sacrificing spot-level cost.

Acknowledgment

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP)-ITRC(Information Technology Research Center) grant funded by the Korea government(MSIT)(IITP-2025-RS-2023-00258649, 50) and IITP grant funded by the Korea government(MSIT)(RS-2024-00398379, 50).

References

[1] S. Lee, J. Hwang and K. Lee, ”SpotLake: Diverse Spot Instance Dataset Archive Service,” 2022 IEEE International Symposium on Workload Characterization (IISWC), Austin, TX, USA, 2022, pp. 242-255, doi: 10.1109/IISWC55918.2022.00029.

[2] CRIU, “Main Page,” CheckpointRestore in Userspace, [Online]. Available: https://criu.org/Main_Page. [Accessed: Oct. 24, 2025].

[3] R. Stoyanov, V. Spišaková, J. Ramos, S. Gurfinkel, A. Vagin, A. Reber, et al., “CRIUgpu: Transparent checkpointing of GPU-accelerated workloads,” arXiv preprint arXiv:2502.16631, 2025.

[4] Z. Mao, T. Xia, Z. Wu, W. L. Chiang, T. Griggs, R. Bhardwaj, et al., “SkyServe: Serving AI models across regions and clouds with spot instances,” in Proc. 20th European Conf. on Computer Systems (EuroSys ’25), Mar. 2025, pp. 159–175.

[5] A. Harlap, A. Chung, A. Tumanov, G. R. Ganger, and P. B. Gibbons, “Tributary: Spot-dancing for elastic services with latency SLOs,” in Proc. 2018 USENIX Annual Technical Conference (USENIX ATC ’18), 2018, pp. 1–14.

[6] J. R. Gunasekaran, C. S. Mishra, P. Thinakaran, B. Sharma, M. T. Kandemir, and C. R. Das, “Cocktail: A multidimensional optimization for model serving in cloud,” in Proc. 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’22), 2022, pp. 1041–1057.

[7] S. Athlur, N. Saran, M. Sivathanu, R. Ramjee, and N. Kwatra, “Varuna: Scalable, low-cost training of massive deep learning models,” in Proc. 17th European Conf. on Computer Systems (EuroSys ’22), Mar. 2022, pp. 472–487.

[8] J. Thorpe, P. Zhao, J. Eyolfson, Y. Qiao, Z. Jia, M. Zhang, et al., “Bamboo: Making preemptible instances resilient for affordable training of large DNNs,” in Proc. 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI ’23), 2023, pp. 497–513.

[9] M. Wagenländer, L. Mai, G. Li, and P. Pietzuch, “Spotnik: Designing distributed machine learning for transient cloud resources,” in Proc. 12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud ’20), 2020.

[10] F. Yang, L. Wang, Z. Xu, J. Zhang, L. Li, B. Qiao, et al., “SNAPE: Reliable and low-cost computing with mixture of spot and on-demand VMs,” in Proc. 28th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’23), Vol. 3, Mar. 2023, pp. 631–643.

[11] F. Yang, L. Wang, Z. Xu, J. Zhang, L. Li, B. Qiao, et al., “SNAPE: Reliable and low-cost computing with mixture of spot and on-demand VMs,” in Proc. 28th ACM Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’23), Vol. 3, Mar. 2023, pp. 631–643.

[12] Hugging Face, “openai-community/gpt2,” Hugging Face Model Hub. [Online]. Available: https://huggingface.co/openai-community/gpt2. [Accessed: Oct. 25, 2025].

[13] Jeet2016, “US Financial News Articles,” Kaggle Dataset. [Online]. Available: https://www.kaggle.com/datasets/jeet2016/us-financial-news-articles. [Accessed: Oct. 25, 2025].

[14] Microsoft, “ResNet-50,” Hugging Face Model Hub. [Online]. Available: https://huggingface.co/microsoft/resnet-50. [Accessed: Oct. 25, 2025].

[15] Thbdh5765, “ILSVRC2012,” Kaggle Dataset. [Online]. Available: https://www.kaggle.com/datasets/thbdh5765/ilsvrc2012. [Accessed: Oct. 25, 2025].