Capsule: Yield-Point Continuation for Mixed LLM Serving and Training

Adhiraj Chhoda
Thomas Jefferson High School for Science and Technology
Alexandria, VA, USA
2027achhoda@tjhsst.edu

***Abstract*—GPU clusters cannot elastically shift accelerators between latency-critical LLM serving and background training because preemption is too expensive: kill-and-restart discards progress, while full GPU checkpointing [1] copies tens of gigabytes of mostly immutable state. We introduce *continuation capsules*, a framework-level substrate that exploits natural yield points in LLM workloads to capture only the mutable delta (key-value (KV) cache pages for serving, optimizer moments and adapter weights for training) while referencing immutable model weights by anchor. For Llama-2-7B LoRA-16 [2] finetuning, the measured capsule delta is $116\times$ smaller than a full DeepSpeed checkpoint with 29 ms capture latency. On a 2-GPU A100 mixed-workload experiment, Capsule completes 95% more training steps than static partitioning while preserving exact continuation: training loss is bitwise identical and serving output is token-exact. Trace-driven simulation shows $2.0\times$ more retained training tokens than kill-and-restart while matching its 0.28% service-level objective (SLO) miss rate.**

I. Introduction

Production GPU clusters serving LLMs operate under rigid static partitioning: interactive serving gets a fixed GPU pool sized for peak load, while finetuning and evaluation jobs get a separate pool. This wastes capacity: serving is bursty (diurnal patterns drive $10\times$ load swings [3]), so a partition sized for peak sits idle at trough, while background training jobs that could fill those GPUs are walled off. In our simulation of a production-derived 24-hour trace on 8 A100 GPUs, static partitioning achieves only 48.8% GPU utilization with a 3.3% SLO miss rate.

The obstacle to elastic sharing is preemption cost. Killing a training job discards minutes of work. Full GPU checkpointing [1] captures the entire address space (60+ GB), taking 15+ seconds. We observe, however, that LLM workloads expose natural *yield points* where mutable state is both consistent and small. At a token boundary in serving, the live state beyond model weights is KV cache pages and sampler metadata. At an optimizer step in training, it is optimizer moments and adapter weights. Table I catalogs these yield points and their delta sizes.

We introduce *continuation capsules*, a runtime abstraction exploiting this decomposition. A capsule is a triple: (1) a *weight anchor* referencing immutable model parameters already on cluster storage, (2) a *mutable delta* containing only live state at the yield point, and (3) a *replay plan* for rematerializing transient state. We implement Capsule in vLLM [4] and DeepSpeed [5], demonstrating exact continuation across serving and training with scheduler-visible costs.

We target parameter-efficient finetuning (LoRA/PEFT) and inference, where the mutable delta is a small fraction of GPU memory; for full-parameter training the optimizer state dominates either representation (Table I, $1.2\times$), so the benefit narrows to checkpoint timing rather than size. Persisting only KV cache (as serving-migration systems do [6], [7]) or only adapter weights (the standard PEFT checkpointing practice) is each individually established; our contribution is to unify them under one continuation abstraction that (i) spans both serving and training stacks, (ii) exposes preemption cost (capture, restore, memory reclaimed) as a scheduler-visible quantity, and (iii) guarantees bitwise-identical resume rather than functional approximation.

II. Design

A. Continuation Capsules

A *yield point* is a framework-level boundary where (i) no GPU kernel required for correctness is in flight, (ii) live mutable state is enumerable, and (iii) the framework signals quiescence. Capsule capture is only permitted at such boundaries.

The weight anchor is a content-addressed reference to immutable model weights (model ID, shard map, integrity hash). Weights are read-only after loading; the anchor is never copied during capture. In the common same-node case, restore incurs zero weight-loading cost.

The mutable delta maps logical state names to tensor descriptors plus metadata. For serving: active KV block IDs, block table, per-request state. For training: optimizer moments, adapter weights, LR scheduler state, data-loader position, RNG states. Transfer uses `cudaMemcpyAsync` with pinned

TABLE I
Capsule delta vs. full snapshot for Llama-2-7B (fp16). Capsule captures only mutable state at the yield point.

Workload	Yield Pt.	Full	Caps.	Ratio
Serve (32s, 512t)	token	60+ GB	8 GB	$7.5\times$
Serve + INT4	token	60+ GB	2 GB	$30\times$
Train (ZeRO-2, 4G)	opt. step	23.6/r	20.2/r	$1.2\times$
Train (LoRA-16)	opt. step	14.3 GB	0.4 GB	$36\times$
Eval	batch	14+ GB	$<$10 MB	$1400\times$

host memory on a dedicated CUDA stream, overlapping with ongoing computation.

The replay plan records how to rematerialize transient state (e.g., remaining prefill chunks or microbatch index for gradient accumulation) rather than snapshotting it.

Capsule supports two capture modes: *exact mode* (lossless fp16 KV, full-precision optimizer moments) guarantees token-identical serving output and bitwise-identical training loss; *compact mode* applies INT4 quantization for $4\times$ smaller deltas but is lossy.

B. Framework Integration

In vLLM, we hook into `EngineCore.step()` at the scheduler iteration boundary. PagedAttention [4] already organizes KV cache as discrete blocks; our callback snapshots the block table and physical KV data at active block IDs. vLLM v1 removed CPU swap in favor of free-and-recompute preemption; Capsule restores KV pages directly, avoiding $O(\text{context_length})$ recomputation.

In DeepSpeed, we hook into `engine.step()` after the optimizer step. At this point, gradients are zeroed and all NVIDIA Collective Communications Library (NCCL) collectives have completed. Only optimizer moments and adapter weights are captured.

C. Capsule Store

The capsule store provides tiered storage with automatic tier selection. The HBM handoff tier ($\sim$1.5 TB/s) keeps deltas in GPU memory for co-located workloads (e.g., swapping LoRA adapters), completing in sub-millisecond latency via CUDA IPC. The host DRAM tier ($\sim$25 GB/s PCIe Gen4) is the default for preemption; an asynchronous pinned-memory copy on a dedicated CUDA stream overlaps with ongoing computation, managed by an LRU eviction policy with a configurable memory budget. The NVMe tier ($\sim$3–6 GB/s) handles longer-term suspension with optional zstd compression (level 1); KV cache pages compress at $\sim 2\times$ losslessly, and combined with INT4 quantization an 8 GB serving delta reduces to under 1 GB on disk. Restore uses the fastest available path: same-GPU HBM $\rightarrow$ same-node host DRAM $\rightarrow$ same-node NVMe $\rightarrow$ cross-node remote fetch over InfiniBand.

D. Group Capsules

Distributed training requires consistent group-level preemption. A *group capsule* bundles per-rank capsules from the same logical optimizer step. On a preemption request, each rank sets a pending-capture flag and continues until the next completed optimizer step. At that boundary, all ranks enter a barrier, capture their local deltas, and report completion to a coordinator, which commits the group capsule only when all ranks have reported. The group capsule additionally records a rank-to-node mapping for potential cross-node restore.

E. Scheduling

The scheduler manages heterogeneous jobs on a shared GPU pool. Each job exposes four metadata values: estimated capture cost, restore cost, memory reclaimed, and SLO slack. Jobs are assigned priorities: serving under SLO pressure is CRITICAL; serving with slack and urgent evals are HIGH; finetuning is NORMAL; background evals are LOW. When a higher-priority job cannot be placed, the scheduler selects a preemption victim set V by maximizing a benefit score:

$$\text{score}(V, r) = \frac{1}{\text{slack}(r) + \epsilon} \, \frac{\sum_{v \in V} m(v)}{\sum_{v \in V} (c(v) + \rho(v)) + \epsilon}$$

where $m(v)$ is memory reclaimed, $c(v)$ capture cost, $\rho(v)$ restore cost, and $\text{slack}(r)$ the requester's remaining SLO slack. Only candidate sets freeing enough memory for requester r are considered. The selected victims are captured to capsules, their GPUs freed, and the requester placed; victims are re-queued with capsule IDs to resume from their last yield points once GPUs become available.

TABLE II
CAPSULE PERFORMANCE ACROSS GPUS FOR LORA-16 TRAINING (4 ATTENTION PROJECTION MODULES). RATIO = DEEPSPEED CHECKPOINT / CAPSULE DELTA.

GPU	Model	Delta	Capture	Restore	Ckpt	Ratio
A100-80GB	7B	235 MB	29 ms	29 ms	27.2 GB	116×
H100-80GB	13B	367 MB	27 ms	14 ms	52.4 GB	143×
B200-180GB	13B	367 MB	26 ms	14 ms	52.4 GB	143×

III. EVALUATION

We evaluate on NVIDIA A100-80GB, H100-80GB, and B200-180GB GPUs with Llama-2-7B and Llama-2-13B models. Microbenchmarks are repeated 9 times (medians reported); the mixed-workload experiment is repeated 3 times with 200 requests per run.

A. Capsule Size and Latency

Table II shows capsule performance across GPU architectures for LoRA-16 training. The capsule delta is 116–143$\times$ smaller than a DeepSpeed checkpoint, with capture latency under 30 ms regardless of GPU generation. Training loss is bitwise identical in all configurations.

Across 5 independent seeds with 200 steps and 10 preemptions each, capture averages 22.7 ± 0.1 ms and restore 15.2 ± 0.4 ms, with all seeds producing bitwise-identical loss curves (max absolute difference 0.0). A 1,000-step stress test with capture every 10 steps (100 preemptions) confirms exact continuation under sustained preemption; loss remains bitwise identical over all 1,000 steps.

For serving, the measured vLLM capture mean is 366 ms (variable-occupancy active set) with 52 ms restore. We validate serving correctness against the natural recomputation baseline, vLLM v1's free-and-recompute preemption, on 200 ShareGPT requests: Capsule achieves 100% token-exact output (12,628/12,628 tokens), whereas recomputation is nondeterministic and diverges on 29 tokens across 2 requests, at $O(\text{context})$ recompute cost. Exactness holds under temperature (T=0.7), top-p (p=0.9), and beam-search decoding.

TABLE III
REAL MIXED-WORKLOAD RESULTS ON 2×A100-80GB (MEAN ± STD OVER 3 RUNS, 200 REQUESTS EACH). TTFT = TIME TO FIRST TOKEN.

Metric	Static	Capsule
Serving requests / run	200	200
Post-adm. p99 TTFT (ms)	126 ± 0	125 ± 0
End-to-end p99 TTFT (ms)	126 ± 0	$3{,}564 \pm 119$
Training steps	119 ± 1	232 ± 3
Preemption events / run	0	20

TABLE IV
SIMULATION RESULTS: SERVING QUALITY AND TRAINING PROGRESS ACROSS SCHEDULING POLICIES (24-HOUR TRACE, 8 A100 GPUS, 2 S TTFT SLO).

Policy	p99 TTFT	SLO Miss	Tokens	Tok/GPU-hr
Static	3,400 ms	3.3%	19.2M	100.4K
Co-locate	>1,500 s	26.1%	19.0M	99.2K
Kill-restart	1,452 ms	0.3%	11.2M	58.6K
Full C/R	15,410 ms	7.8%	19.4M	101.0K
Capsule	1,346 ms	0.28%	21.9M	114.1K

B. Mixed-Workload Experiment

Table III reports results from a real 2-GPU experiment with vLLM serving and DeepSpeed LoRA-16 training on Llama-2-7B.

Capsule completes 232 ± 3 training steps vs. 119 ± 1 for static partitioning (95% more) with identical post-admission p99 TTFT (125 ± 0 vs. 126 ± 0 ms). End-to-end p99 TTFT rises to 3.6 ± 0.1 s, dominated by time-to-yield (waiting for the training worker to reach its next yield point, with measured p50 of 469 ms); once admitted, serving latency matches the static baseline. Across 60 preemption events (3 runs), zero training tokens are lost.

C. Cluster-Level Simulation

A trace-driven 24-hour simulation on 8 A100 GPUs compares five scheduling policies: static partitioning, co-location, kill-restart, full GPU checkpoint/restore, and Capsule. Co-location runs serving and training concurrently on shared GPUs via temporal (MPS-style) sharing with a measured interference penalty applied to both and no preemption; kill-restart checkpoints training every 5 min and discards work since the last checkpoint on each preemption; full C/R uses CRIUgpu-estimated 15 s capture and 15 s restore. The trace replays diurnal serving load (peak-to-trough ratio $10\times$) alongside continuous LoRA-16 training jobs. All capture, restore, and transfer latencies are set to measured values from Section III. Static partitioning allocates 4 GPUs each to serving and training; other policies share the full pool. Table IV summarizes the key tradeoffs.

Capsule achieves the highest retained training throughput (114.1K tokens/GPU-hr, $1.95\times$ kill-restart) while matching kill-restart on serving quality (0.28% SLO miss, 1.3 s p99 TTFT). Kill-restart wastes 89,462 GPU-seconds of rolled-back work across 1,719 preemptions; Capsule loses zero across 6,309 preemptions.

TABLE V
CROSS-NODE CAPSULE TRANSFER (2×A100-80GB, INFINIBAND, LLAMA-2-7B). ONLY THE MUTABLE DELTA CROSSES THE NETWORK.

Phase	Serving	LoRA-16
Capture	13.8 ms	111.9 ms
Network transfer	98.5 ms	1,149 ms
Remote load	69.9 ms	572.9 ms
Replay	64.6 ms	—
Total (net+load+replay)	233 ms	1,722 ms
Delta transferred	16 MB	235 MB

TABLE VI
4-GPU A100 REAL EXPERIMENT (50 REQUESTS, 5 RUNS).

Policy	Train steps	p99 TTFT (ms)	Preempts
Static	53 ± 1	282 ± 2	0
Kill-restart	90 ± 3	160 ± 2	4
Full C/R	58 ± 2	158 ± 0	8
Capsule	57 ± 2	159 ± 2	8

D. Cross-Node Transfer and Multi-GPU Results

A practical scheduler must also move workloads between nodes. We validate cross-node capsule transfer on two A100-80GB nodes connected by InfiniBand (Table V). For serving, the 16 MB mutable delta (active KV pages) transfers in 98.5 ms, with total end-to-end restore of 233 ms including network, load, and replay. For LoRA-16 training, the 235 MB delta transfers in 1,149 ms, yielding 1,722 ms total restore. In both cases, model weights (13.5 GB) are already resident on the destination via the weight anchor; only the mutable delta crosses the network.

We also validate on a 4-GPU A100 node. Each policy runs 50 serving requests and trains LoRA-16 on all 4 GPUs across 5 repeated runs. Capsule achieves p99 TTFT of 159 ± 2 ms, nearly half the 282 ± 2 ms of static partitioning, because all 4 GPUs can serve during bursts. Kill-restart achieves similar TTFT but discards progress at each preemption. Table VI summarizes the 4-GPU results.

E. Component Analysis

Table VII reports a component analysis for serving on Llama-2-7B (32 sequences, 512 tokens). Exact mode (fp16 KV) yields 320 ms analytical capture at PCIe bandwidth. Compact mode (INT4) reduces this to 80 ms but is lossy. Without weight anchors, capture grows to 620 ms (+13.5 GiB of model weights), confirming that the anchor/delta separation is the primary source of size reduction for serving and LoRA workloads.

IV. RELATED WORK

Mixed serve+tune engines. FineInfer [8], FlexLLM [9], LLMStation [10], and SIRIUS [11] co-execute serving and training on shared GPUs but target integrated scheduling rather than a reusable continuation abstraction with exact resume. None expose preemption cost as a scheduler-visible quantity or guarantee bitwise-identical continuation.

TABLE VII
COMPONENT ANALYSIS FOR SERVING CAPTURE (32 SEQ, 512 TOK, LLAMA-2-7B, 25 GB/S PCIE).

Config.	Delta	Capture	Impact
Exact (fp16 KV)	8 GiB	320 ms	Exact resume
Compact (INT4)	2 GiB	80 ms	Lossy, 4× smaller
− Wt. anchors	15.5 GiB	620 ms	7.8× larger
− Replay	8 GiB	320 ms	Recompute prefix

Serving migration. Llumnix [6] migrates KV cache across instances for load balancing. SpotServe [12] handles spot preemption for serving via reparallelization. ServerlessLLM [7] optimizes cold-start latency. All three are serving-specific and do not address training state.

Training checkpointing. DataStates-LLM [13] provides asynchronous I/O for LLM checkpoints. ByteCheckpoint [14] unifies checkpoint formats across frameworks. Checkmate [15] replicates gradients over the network to eliminate checkpoint I/O. Parcae [16] optimizes liveput on preemptible instances. CheckFreq [17] overlaps checkpointing with computation for frequent training checkpoints, but even a zero-cost checkpoint still rolls back to the last checkpoint on preemption, whereas Capsule resumes from the exact yield point with no rollback. None of these systems address serving or expose preemption cost to a cross-workload scheduler.

Generic GPU checkpointing. CRIUgpu [1] and PhoenixOS [18] capture full GPU address spaces without exploiting LLM structure. PipeSwitch [19] and GPreempt [20] pipeline model loading for faster switching but still transfer full model state.

Cluster schedulers. Gandiva [21], Tiresias [22], Pollux [23], and GFS [24] manage GPU clusters but treat preemption as kill-restart. Capsule provides the runtime substrate that tells the scheduler what to capture and what it costs.

V. DISCUSSION

Time-to-yield dominates end-to-end preemption latency. Capsule capture itself takes 22–29 ms for LoRA-16, but the scheduler must wait for the current workload to reach its next yield point. For training, this delay equals one full forward-backward pass (measured p50: 469 ms on Llama-2-7B). For serving, the delay is one decode iteration ($\sim$30 ms). Reducing time-to-yield through finer-grained yield points (e.g., per-microbatch or mid-layer checkpoints) is the highest-impact optimization target.

Scope of exactness validation. Token-exact continuation is validated for greedy decoding (5 seeds, 100 preemptions), temperature sampling (T=0.7), top-p (p=0.9), and beam search, all on same-node restore. Cross-node restore is spot-checked. INT4 compact-mode captures are lossy by design and not claimed to be exact.

Architectural coverage. The vLLM adapter targets vanilla continuous batching with paged KV. Tensor-parallel serving (TP=2) works but the capsule KV adapter does not yet support multi-GPU capture. Pipeline-parallel serving and grouped-query attention (GQA)/multi-query attention (MQA)/mixture-of-experts (MoE) architectures would require new yield-point callbacks but no changes to the core capsule abstraction. Extending to other frameworks (e.g., TensorRT-LLM, SGLang) follows the same pattern: identify yield points, register capture callbacks, and implement delta serialization.

Number format coverage. The current compact mode uses INT4 quantization for lossy KV compression. Lossless formats such as DFloat11 and block-scaled formats like MxFP4/MxFP6 could offer intermediate points on the size–fidelity tradeoff; exploring these is a natural next step.

Scheduler simplicity. The scheduler demonstrates that continuation costs are useful for preemption decisions; it is not a scheduling contribution. It lacks fairness guarantees and online cost adaptation. Because the benefit score scales inversely with SLO slack, low-priority jobs with small capture costs could in principle be thrashed back-to-back; a production deployment would need per-job cooldown timers or minimum-run-length guarantees to prevent starvation. A production scheduler like GFS [24] could use Capsule as a runtime substrate, replacing coarse checkpoint/restart costs with fine-grained semantic continuation costs.

VI. CONCLUSION

We introduced continuation capsules, a framework-level substrate for exact, low-latency pause/resume of mixed LLM workloads at semantic yield points. Capsule decomposes GPU state into immutable weight anchors, mutable deltas, and replay plans, yielding 7.5× to 116× smaller preemption snapshots for serving and LoRA finetuning. On real hardware, Capsule enables 95% more training steps than static partitioning while preserving exact continuation. Time-to-yield, the delay to the next legal preemption boundary, is the dominant residual cost and the most important target for future scheduling optimization.

REFERENCES

[1] R. Stoyanov, V. Spisakova, J. Ramos, S. Gurfinkel, A. Vagin, A. Reber, W. Armour, and R. Bruno, "CRIUgpu: Transparent checkpointing of GPU-accelerated workloads," arXiv preprint arXiv:2502.16631, 2025. [Online]. Available: https://arxiv.org/abs/2502.16631

[2] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *Tenth International Conference on Learning Representations (ICLR 2022)*, 2022. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[3] Q. Hu, P. Sun, and T. Zhang, "Understanding the workload characteristics of large language model development," USENIX ;login: Online, mar 2024. [Online]. Available: https://www.usenix.org/publications/loginonline/understanding-workload-characteristics-large-language-model-development

[4] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with PagedAttention," in *Proceedings of the 29th ACM Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 611–626. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[5] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. Virtual Event / Atlanta, Georgia, USA: IEEE/ACM, 2020, p. 20. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00024

[6] B. Sun, Z. Huang, H. Zhao, W. Xiao, X. Zhang, Y. Li, and W. Lin, "Llumnix: Dynamic scheduling for large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 173–191. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/sun-biao

[7] Y. Fu, L. Xue, Y. Huang, A.-O. Brabete, D. Ustiugov, Y. Patel, and L. Mai, "ServerlessLLM: Low-Latency serverless inference for large language models," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 135–153. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/fu

[8] Y. He, H. Yang, Y. Lu, A. Klimovic, and G. Alonso, "FineInfer: Serving large language models with concurrent inference and fine-tuning," in *Proceedings of the 4th Workshop on Machine Learning and Systems (EuroMLSys 24)*. ACM, 2024.

[9] G. Oliaro, X. Miao, X. Cheng, V. Kada, M. Wu, R. Gao, Y. Huang, R. Delacourt, A. Yang, Y. Wang, C. Unger, and Z. Jia, "FlexLLM: Token-level co-serving of LLM inference and finetuning with SLO guarantees," in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association, 2026.

[10] Y. He, H. Yang, Y. Lu, A. Klimovic, and G. Alonso, "Resource multiplexing in tuning and serving large language models," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. Boston, MA: USENIX Association, Jul. 2025, pp. 1639–1655. [Online]. Available: https://www.usenix.org/conference/atc25/presentation/he-yongjun

[11] J. Wang, Y. Wang, M. Han, and R. Chen, "Colocating ML inference and training with fast GPU memory handover," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. Boston, MA: USENIX Association, Jul. 2025.

[12] X. Miao, C. Shi, J. Duan, X. Xi, D. Lin, B. Cui, and Z. Jia, "SpotServe: Serving generative large language models on preemptible instances," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 1112–1127. [Online]. Available: https://doi.org/10.1145/3620665.3640411

[13] A. Maurya, R. Underwood, M. M. Rafique, F. Cappello, and B. Nicolae, "DataStates-LLM: Lazy asynchronous checkpointing for large language models," in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 227–239. [Online]. Available: https://doi.org/10.1145/3625549.3658685

[14] B. Wan, M. Han, Y. Sheng, Y. Peng, H. Lin, M. Zhang, Z. Lai, M. Yu, J. Zhang, Z. Song, X. Liu, and C. Wu, "ByteCheckpoint: A unified checkpointing system for large foundation model development," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. Philadelphia, PA: USENIX Association, Apr. 2025, pp. 559–578. [Online]. Available: https://www.usenix.org/conference/nsdi25/presentation/wan-borui

[15] A. Bhardwaj, W. Wang, J. Carin, A. Belay, and M. Ghobadi, "Checkmate: Zero performance overhead model checkpointing via network gradient replication," in *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*. USENIX Association, 2026.

[16] J. Duan, R. Lu, H. Duanmu, C. Shi, X. Zhang, D. Lin, I. Stoica, and H. Zhang, "Parcae: Proactive, liveput-optimized DNN training on preemptible instances," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. Santa Clara, CA: USENIX Association, Apr. 2024.

[17] J. Mohan, A. Phanishayee, and V. Chidambaram, "CheckFreq: Frequent, fine-grained DNN checkpointing," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 203–216.

[18] X. Wei, Z. Huang, T. Sun, Y. Hao, R. Chen, M. Han, J. Gu, and H. Chen, "PHOENIXOS: Concurrent OS-level GPU checkpoint and restore with validated speculation," in *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, ser. SOSP '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 996–1013. [Online]. Available: https://doi.org/10.1145/3731569.3764813

[19] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin, "PipeSwitch: Fast pipelined context switching for deep learning applications," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. Virtual Event: USENIX Association, Nov. 2020, pp. 499–514. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/bai

[20] R. Fan, T. Ren, M. Xie, S. Gao, J. Shu, and Y. Lu, "GPREEMPT: GPU preemptive scheduling made general and efficient," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*. Boston, MA: USENIX Association, Jul. 2025, pp. 263–272. [Online]. Available: https://www.usenix.org/conference/atc25/presentation/fan

[21] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang, F. Yang, and L. Zhou, "Gandiva: Introspective cluster scheduling for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 595–610. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/xiao

[22] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Zhuo, "Tiresias: A GPU cluster manager for distributed deep learning," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 485–500. [Online]. Available: https://www.usenix.org/conference/nsdi19/presentation/gu

[23] A. Qiao, S. K. Choe, S. J. Subramanya, W. Nishi, C. Ho, G. Pekhimenko, and E. P. Xing, "Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. Virtual Event: USENIX Association, Jul. 2021, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/qiao

[24] J. Duan, S. Xu, S. Qian, D. Yang, K. Wang, C. Liao, Y. Yu, Q. Hua, H. Hu, Q. Wang, W. Wu, D. Bao, T. Lu, J. Cao, G. Xue, G. Yang, L. Zhang, and G. Chen, "GFS: A preemption-aware scheduling framework for GPU clusters with predictive spot instance management," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ser. ASPLOS '26. New York, NY, USA: Association for Computing Machinery, 2026, pp. 117–131. [Online]. Available: https://doi.org/10.1145/3760250.3762231