

TorchPass Snapshots for training and inference

October 8, 2026 · clockwork.io/blog/torchpass-snapshots-for-training-and-inference/

How saving a running job differs from cloning an initialized engine

A training job might need to drain its machines for maintenance. An inference service might need three new replicas for a traffic spike. Both can use a GPU snapshot. Each needs a different result from it.

For training recovery, the snapshot preserves one job's progress and execution so that the same job can continue after an interruption. For inference cold starts, the snapshot captures an engine just after initialization, so new replicas can start from it in seconds instead of repeating minutes of initialization. This article compares those two workflows.



What a snapshot preserves

A process snapshot saves supported execution state so the program can resume from its saved point. CRIU, Checkpoint/Restore In Userspace, handles Linux process state. NVIDIA's cuda-checkpoint handles supported CUDA state. It copies GPU memory to host memory, releases GPU resources and reconstructs them on restore [1].

As an aside, process snapshots are often confused with application checkpoints, so it's important to highlight the difference. A process snapshot operates at the platform layer, whereas an application checkpoint records logical training state that the application or framework declares. This includes weights, optimizer state and training progress. Recovery builds a runtime and loads those records. Formats such as PyTorch Distributed Checkpoint can redistribute saved state across supported changes in GPU count or parallelism [2].

The table below contrasts how snapshots are used for training recovery and inference cold-starts:

Dimension	Training recovery snapshot	Inference cold-start snapshot
Saved point	A consistent point in an active job	An initialized engine before live traffic
State that matters	Model weights, optimizer state, and each worker's CPU and GPU state at point of capture.	Initialized serving state, such as processed weights in HBM, compiled kernels, allocated KV cache buffers, and captured CUDA graphs
Refresh policy	New save points are needed as the training job advances.	Snapshots are taken once per model and runtime configuration. They are only rebuilt when either the model or environment changes.
Success	The job can continue successfully when restored from the snapshot.	Each replica can start from the snapshot and is ready to serve correctly.

Sources: [3,11,14].

Why each side wants snapshots

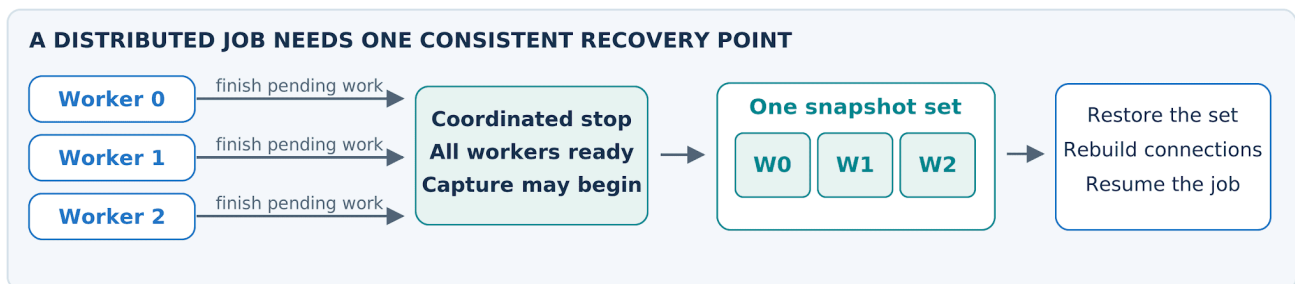
Training: Needs to preserve progress through interruptions. Preemption, maintenance and capacity consolidation can require a healthy job to stop. A protected snapshot can let it resume later on compatible capacity without any lost work. Periodic snapshots can also support recovery after a sudden failure. Noting that any work after the selected snapshot was taken must then be repeated [14]. Application checkpoints can automate these workflows too if they have a tested integration that saves complete state and resumes correctly [6]. However, the advantage of Platform snapshots is that cloud operators and platform teams can save training state without relying on or coordinating with application teams, particularly for event-driven interruptions such as preemption, maintenance and capacity consolidation [14] (especially important for Neoclouds).

The cost of interruption is very visible in large training runs. Meta recorded 466 planned and unplanned interruptions over 54 days during Llama 3 training on 16,384 GPUs [21]. In a separate AWS test, a Llama-3 70B training job on 256 H100 GPUs took 4 minutes 52 seconds to recover through conventional checkpoint-based recovery [22].

Inference: Needs to preserve startup work for fast replica bring-up. In order to account for fluctuations in inference traffic, inference services must be able to add or remove replicas on demand [4]. This means spinning new serving replicas up, where each requires costly initialization work such as starting its framework, loading weights, compiling kernels, allocating its KV cache and capturing CUDA graphs [4]. When startup is slow, inference services keep idle replicas running in reserve to absorb peak traffic [4]. Restoring a snapshot taken after the initialization steps lets a new replica skip much of that work, enabling new replicas to be added elastically [3]. Potoroom reports about 35 to 45 seconds with a snapshot restore versus about 220 seconds without one for one image-editing configuration (the container image was already on the node) [5].

The training challenge: Keep the workers consistent

A distributed job has cooperating worker processes, or ranks (often hundreds or thousands of them). They exchange data through collective operations, most commonly using the NVIDIA Collective Communications Library (NCCL). A collective needs the participating ranks to make progress. Freezing one rank while another waits for it can prevent capture from completing [1,12]. The ranks must each reach a consistent communication point. That point can be partway through a training step; it is not an arbitrary instant [13,14].



Network resources need separate handling. Remote direct memory access, or RDMA, relies on connections and registered memory associated with the running processes. Saving process memory does not make those old connections portable [3,14]. NVIDIA's NCCL checkpoint shim relies on the application to enter the checkpoint sequence explicitly. It destroys the communicators before capture and reconstructs them after restore. Its documented workflow does not provide the platform-controlled coordination needed to bring an unmodified, actively running job to that stopping point. The shim also currently excludes CUDA graph capture [7]. Clockwork's TorchPass supplies that coordination through its NCCL interceptor. It brings the required workers to a consistent communication point before releasing transport resources and capturing their state [14].

Restore also needs a compatible destination. GPU topology, driver, kernel, container environment and required files must meet the capture system's rules. A newer driver alone does not establish compatibility. Success means the complete saved state must survive removal of the source

machines [14,16].

The inference challenge: Restore replicas with the correct state on new machines

An inference snapshot preserves an initialized engine for reuse. Each restored replica must be prepared to run on its destination before it serves requests. This includes rebuilding connections and shared GPU resources where required, refreshing replica-specific identifiers, and loading any model weights omitted from the image. Captured CUDA graphs must also refer to valid memory addresses and communication resources [3,8].

- **Multi-GPU snapshots must restore shared memory and communication.** Tensor parallelism splits a model across GPUs. During initialization, its workers set up access to shared GPU buffers and establish NCCL communication. Restoring the engine on another machine requires the workers to regain that access and communication. NVIDIA's checkpoint utility supports some GPU-memory-sharing mechanisms, but it cannot capture all the mechanisms an inference engine may use [1]. Disabling those mechanisms may affect serving performance [19].
- **Saved CUDA graphs must work on the restored replica.** A CUDA graph records GPU operations for fast replay, including the memory addresses and resources they use. Rebuilding NCCL communication can leave the graph referring to resources that no longer exist. Restore must preserve those references or update them where supported. Otherwise, the affected graph must be captured again, adding startup work [7,8].
- **Each restored replica needs its own identity and service registration.** A snapshot can retain the original engine's identifiers and connection details. Reusing those identifiers can make several replicas appear to be the same instance [15]. Before accepting requests, each replica must have a unique identity and be registered with its current network address. Routers and other services can then find the correct instance [3].

A STARTUP TEMPLATE STILL NEEDS RESTORE WORK

REUSE

- Initialized runtime
- Compiled kernels
- Validated CUDA graphs

REBUILD

- Empty KV-cache storage
- Supported connections
- Weights, if stored separately

REFRESH

- Replica identity
- Service registration
- Credentials / random state

Then admit live requests on each restored replica.

Where the Industry stands today

Inference support depends on the engine and GPU layout.

As of October 2026, the table below describes the current status of several offerings

Implementation	Documented snapshot scope
vLLM 0.31.0	Experimental initialized-engine create/restore for tensor parallelism of one (TP=1). [10]
NVIDIA Dynamo v1.4.2	Single-GPU vLLM/SGLang support; multi-GPU vLLM is highly experimental; multi-node work remains in progress. [17]
Standalone Dynamo Snapshot	The separate ai-dynamo/snapshot project lists single-GPU workloads only. [18]
Modal GPU Memory Snapshots	Alpha; generally incompatible with multi-GPU code. [11]
GKE Pod snapshots	Documents multi-GPU Pods on L4 GPUs with GKE Sandbox. This is a constrained supported case, not a general multi-GPU guarantee. [9]

Training: checkpoints, peer recovery and platform snapshots

Application checkpoints remain the standard recovery path for large training runs [2,22]. AWS HyperPod also offers framework-integrated recovery from state held by surviving peers, reducing recovery to 47 seconds in the same 256-GPU test [22]. Microsoft reports transparent checkpointing in its internal Project Forge production system [24].

Clockwork's TorchPass is the first commercially available product for platform-controlled snapshots of running multi-node GPU training jobs without training-code changes [20]. TorchPass brings the required workers to a consistent stopping point, captures their process and GPU state, and rebuilds communication when they resume. Cloud operators can initiate this operation without application-specific checkpoint callbacks [14].

TorchPass has two capabilities, both of which use snapshots. **TorchPass live migration** moves departing workers onto available replacement capacity in real-time so training can continue after just a short pause. **TorchPass snapshots** can preserve the job until compatible capacity becomes available. Before releasing the original machines, the platform must protect the complete snapshot outside those machines [14], by either replicating it somewhere else or placing it on external persistent storage.

In a preliminary Clockwork benchmark on eight nodes with 64 H200 GPUs, capture-and-resume pauses averaged just 18.43 seconds for a Megatron-LM job.

Two problems, two sets of questions

For training, success means continuing the saved job correctly. Ask how long snapshot capture pauses the workers, when the complete image becomes protected, and how long restore takes to resume useful training. Check how much work must be repeated after a failure. Confirm which types of failure the saved image survives and whether the platform can initiate capture without application-specific checkpoint code [14].

For inference, success means starting a correctly serving replica. Measure the time from the launch request to readiness, including any remaining weight loading, communication setup, graph recapture and service registration. Compare against a tuned cold start on the same configuration. Verify serving throughput as well as startup time [3,8,19].

For anyone evaluating snapshots, a pilot on the intended model, GPU layout and runtime should test desired outcomes. A training snapshot should preserve the job through an interruption; An inference startup snapshot should make an initialized engine reusable across new replicas.

Sources

[1] NVIDIA. The cuda-checkpoint Utility. Repository README. See “CRIU,” “The Utility,” “610 Features,” and “Functionality” for capture and restore behavior, supported inter-process memory and remaining limitations. GitHub

[2] PyTorch. Distributed Checkpoint — torch.distributed.checkpoint. PyTorch 2.14 documentation. Describes model and optimizer checkpointing, in-place loading and load-time resharding. PyTorch Docs

[3] Saereesitthipitak, S., Feigin, D., Mailthody, V. S., and Khadkevich, M. NVIDIA Dynamo Snapshot: Fast Startup for Inference Workloads on Kubernetes. NVIDIA Technical Blog, 27 May 2026. See “Dynamo Snapshot: The workload” for the capture boundary and post-restore service registration, and the optimization sections for memory and weight handling. NVIDIA Developer

[4] Kabakibo, H. S., Trivedi, A., and Wang, L. Breaking the Ice: Analyzing Cold Start Latency in vLLM. arXiv:2606.07362, version 3, 29 June 2026. Analyzes the stages of inference-engine startup. arXiv

[5] Lacalle Castillo, D. How we cut inference cold start to seconds with NVIDIA Dynamo. Photoroom, 23 July 2026. See “Results,” including the approximately 220-second versus 35–45-second comparison when the container image is already on the node. Photoroom

- [6] Amazon Web Services. Using elastic training in Amazon SageMaker HyperPod. Amazon SageMaker AI Developer Guide. See “Training code modification” and “Implement Checkpoint Saving and Checkpoint Loading.” AWS Documentation
- [7] NVIDIA. NCCL Checkpoint Shim README. NCCL repository, contrib/nccl_checkpoint. Describes explicit application preparation and restore calls, communicator reconstruction and current limitations, including CUDA graph capture. GitHub
- [8] NVIDIA. Constraints. CUDA Graph Best Practice for PyTorch. See “Static Graph Parameters,” “Memory Addresses,” and “Memory Lifetime Requirements.” NVIDIA Docs
- [9] Google Cloud. About GKE Pod snapshots. Google Kubernetes Engine documentation. Updated 2 October 2026. See “Requirements” and “Limitations” for GKE Sandbox and the documented multi-GPU L4 configurations. Google Cloud Documentation
- [10] vLLM Project. Release v0.31.0. 5 October 2026. See “Highlights → Fast restart” for experimental initialized-engine snapshot creation and restoration with tensor parallelism of one. GitHub
- [11] Modal. Memory Snapshots. Modal documentation. See “GPU Memory Snapshots,” “Limitations of GPU Memory Snapshots,” and the FAQs on snapshot refresh and randomness. Modal
- [12] Shukla, D., Sivathanu, M., et al. Singularity: Planet-Scale, Preemptible and Elastic Scheduling of AI Workloads. arXiv:2202.07848, 2022. Sections 4.3, “Communication state,” and 4.3.1, “Distributed barrier.” Cited here for the collective-communication stopping problem, not for Clockwork’s implementation or commercial status. arXiv
- [13] Stoyanov, R., Spišáková, V., Ramos, J., et al. CRIUgpu: Transparent Checkpointing of GPU-Accelerated Workloads. arXiv:2502.16631, 2025. Section 4.5, “Multi-GPU Checkpoints,” explains the additional synchronization required for multi-node workloads. arXiv
- [14] Clockwork. TorchPass Snapshots: Platform-Controlled Snapshots for Distributed Training.

[15] Amazon Web Services. Handling uniqueness with Lambda SnapStart. AWS Lambda Developer Guide. General guidance on uniqueness when multiple execution environments are restored from the same snapshot. This is not a GPU-snapshot implementation reference. AWS Documentation

[16] CRIU Project. Checkpoint and Restore for CUDA applications with CRIU. CUDA plugin README, release v4.2.1. See “Checkpointing Procedure” and “Known Limitations,” including GPU-type and GPU-count requirements. GitHub

[17] NVIDIA. Snapshotting GPU Workers. NVIDIA Dynamo documentation, version 1.4.2. See “Backend and Topology Support.” NVIDIA Dynamo Documentation

[18] NVIDIA. Dynamo Snapshot — ai-dynamo/snapshot. Standalone project README, especially “Limitations.” The documented scope is distinct from the integrated Dynamo backend/topology matrix. GitHub

[19] NVIDIA. Dynamo snapshot lifecycle implementation. components/src/dynamo/common/snapshot/lifecycle.py, function `configure_snapshot_capture_env()`. Documents the snapshot-mode changes to NCCL allocation and communication features. GitHub

[20] Clockwork.io. Clockwork.io Raises \$31M as LinkedIn, Together AI and WhiteFiber Adopt Its Resilience Software to Stop Wasting GPU-Hours. Company announcement distributed by PR Newswire, 5 October 2026. Announces multi-node platform snapshots for training and states Clockwork’s industry-first claim.

[21] Llama Team, AI @ Meta. “The Llama 3 Herd of Models.” arXiv:2407.21783, 2024, §3.3.4, “Reliability and Operational Challenges.” Supports the 466 interruptions over 54 days.

[22] Amazon Web Services. “Checkpointless training on Amazon SageMaker HyperPod: Production-scale training with faster fault recovery.” AWS Machine Learning Blog, 15 December 2025. The “Performance results” table reports 4 minutes 52 seconds for traditional recovery and 47 seconds

for checkpointless recovery at 256 GPUs. The implementation sections explain the framework integration and surviving-peer state.

[24] Microsoft. "Sustainable by design: Innovating for energy efficiency in AI, part 1." 12 September 2024. Reports Project Forge's internal production use, including transparent checkpointing. It does not establish an equivalent customer-deployable multi-node snapshot product.