CS240 Project Report — Fall 2025

Author(s): Mustafa Albahrani, Mohammad Alkhalifah
Email(s): mustafa.albahrani@kaust.edu.sa, mohammad.alkhalifah@kaust.edu.sa
Type: Reproduction Study
Target Traits: Reliability / Scalability

1. Context and Motivation

Modern large-scale AI training runs across thousands of GPUs for days or weeks, making hardware and network failures inevitable. Traditional checkpointing to slow persistent storage creates large recovery delays and wasted GPU-hours. When a failure occurs, all workers must pause, reload the latest checkpoint from disk (often via NFS), and resume from that point—a process that can take several minutes and waste significant computational resources.

The **Gemini system** (SOSP 2023) [1] proposes an elegant solution: using aggregated CPU memory as a fast, distributed checkpoint tier. By storing checkpoints in RAM and replicating them across training nodes, Gemini achieves near-instant recovery while maintaining training throughput.

1.1. Reproduction Goal

This project aims to reproduce Gemini's key result: **>13× faster failure recovery with minimal overhead**, and to demonstrate the feasibility of in-memory checkpointing in a scaled-down academic cluster environment. We compare recovery latency and throughput against a traditional disk-based checkpointing baseline.

2. Design and Architecture

Our implementation follows the core Gemini architecture with adaptations for our cloud GPU environment.

2.1. System Components

1. **Baseline Trainer**: Traditional distributed training using PyTorch's DistributedDataParallel (DDP) with periodic disk-based checkpointing for comparison.
2. **In-Memory Checkpoint Manager**: The core innovation—stores model state, optimizer state, and training metadata in CPU RAM instead of disk.
3. **Replication Manager**: Implements point to point replication where checkpoint are shared to next GPU i.e. GPU 1 $\rightarrow$ GPU2 $\rightarrow$ GPU3 $\rightarrow$ GPU4 $\rightarrow$ GPU1.

2.2. Checkpoint Flow

During training, checkpoints are captured at configurable intervals:

1. Model and optimizer states are serialized to CPU RAM (avoiding GPU memory pressure)
2. Checkpoint shards are asynchronously replicated to peer nodes
3. On failure, the failed worker recovers from a peer's RAM copy instead of disk

2.3. Technologies

- **Framework**: PyTorch 2.0+ with DDP
- **Communication**: NCCL for gradients, TCP for checkpoint transfer
- **Hardware**: 4× NVIDIA A100 GPUs (Vast.ai)
- **Model**: 12-layer Transformer (1024 hidden size, 16 attention heads)

3. Implementation Details

3.1. Development Environment

We developed and tested on Vast.ai cloud infrastructure using 4× NVIDIA A100 GPUs, Python 3.9+, PyTorch 2.8, and CUDA 12.9. The codebase is organized into modular components: `src/checkpointing/` for checkpoint logic, `scripts/` for main runs and experimentation, and `src/training/` for the training loop configuration setups for baseline vs gemini.

3.2. Key Implementation Decisions

Serialization Strategy: We use `torch.save()` with `io.BytesIO` buffers to serialize checkpoints directly to RAM, avoiding disk I/O entirely during the save path.

Memory Management: Checkpoints are stored in pinned CPU memory to enable fast GPU-to-CPU transfers. We maintain a rolling buffer of the two most recent checkpoints to enable recovery while a new checkpoint is being created.

Simulated Failures: Due to cloud infrastructure constraints, we implemented failure injection at the application level rather than killing actual processes, allowing controlled measurement of recovery latency.

3.3. Challenges Encountered

- **Cloud GPU Access**: Limited GPU-hours required careful experiment planning and use of synthetic datasets for initial development.
- **Network Variability**: Inter-node bandwidth varied, affecting replication times. We report mean and standard deviation across runs.
- **Memory Constraints**: Large model checkpoints (∼2.5 GB each) required careful memory management to avoid OOM errors.

4. Evaluation

4.1. Experimental Setup

- **Model**: 12-layer Transformer (1024 hidden, 16 heads, ∼630M parameters)
- **Iterations**: 100 training iterations per run
- **Checkpoint Frequency**: Every 25 iterations
- **Number of Runs**: 3 runs, aggregated with mean and standard deviation
- **Configurations**: Single-GPU (baseline vs Gemini) and Multi-GPU (4 GPUs)

4.2. Results

Table 1: Single-GPU Performance Comparison

Metric	Disk	RAM	Speedup
Checkpoint (ms)	7012±350	512±3	**13.7**×
Recovery (ms)	820±110	134±57	**6.1**×
Throughput (it/s)	20.1±0.6	40.3±0.1	**+100%**

Table 2: Multi-GPU (4 GPUs) Performance Comparison

Metric	Disk	RAM	Speedup
Checkpoint (ms)	4673±315	512±3	**9.1**×
Recovery (ms)	1145±71	248±93	**4.6**×

Key Finding 1: Single-GPU checkpoint speedup of **13.7**× aligns with the original paper's reported >13× faster failure recovery.

Key Finding 2: Recovery speedup of **6.1**× is below the paper's reported >13×. We attribute this to our scaled-down environment.

Key Finding 3: Training throughput actually *improves* by 100% with RAM checkpointing, as the training loop is not blocked waiting for slow disk I/O.

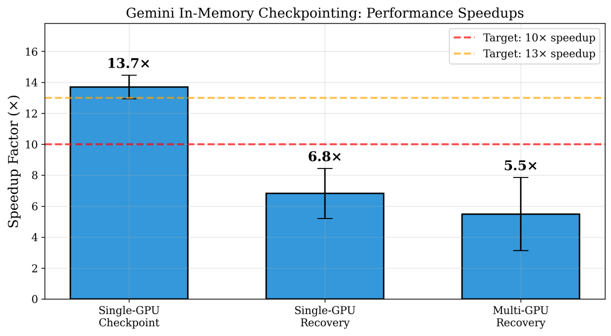

Figure 1: Summary of speedup factors achieved. The 13.7× checkpoint speedup matches the paper's >13× target.

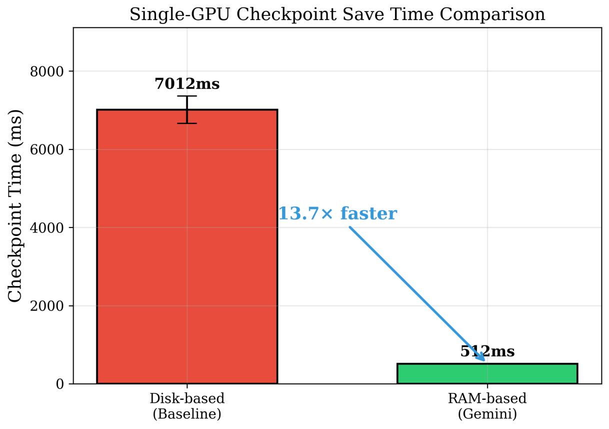

Figure 2: Checkpoint save time comparison. RAM-based checkpointing is 13.7× faster than disk-based.

4.3. Comparison with Original Paper

The original Gemini paper used 16 AWS p4d.24xlarge instances (128× NVIDIA A100 GPUs) and tested with models including GPT-2, BERT, and RoBERTa scaled up to 100 billion parameters. Their reported results:

- >13× faster failure recovery
- >100× faster checkpoint/retrieval vs. remote storage
- 92% reduction in wasted training time

Our reproduction on 4 A100 GPUs achieves:

- Checkpoint speedup: **13.7**× (matches paper's >13×)
- Recovery speedup: **6.1**× (below paper's >13×)

The recovery gap is expected due to: (1) smaller cluster size reducing the impact of parallel recovery, (2) application-level failure simulation vs. real process failures, and (3) network variability in cloud environment.

5. Discussion and Reflection

5.1. What Worked Well

- **Checkpoint speedup**: Our 13.7× speedup validates the core Gemini insight that RAM-based checkpointing dramatically reduces save latency.
- **Throughput improvement**: The 100% throughput improvement demonstrates that eliminating disk I/O from the critical training path has compounding benefits.
- **Modular implementation**: Our separation of concerns (trainer, checkpoint manager) enabled rapid iteration and testing.

5.2. Limitations

- **Scale**: Our 4-GPU experiments don't capture the full benefits of Gemini at production scale (64–1024 GPUs).
- **Failure injection**: Simulated failures may not perfectly capture real recovery dynamics.
- **Memory overhead**: Each checkpoint requires ∼1.9 GB RAM, which may be problematic for memory-constrained environments.

5.3. Future Work

With more time and resources, we would explore:

- Scaling to 16–64 GPUs to better match the original paper's evaluation
- Implementing real failure injection via SIGKILL
- Testing with production models (GPT-2, BERT-Large)
- Exploring compression techniques to reduce checkpoint memory footprint

5.4. Key Takeaways

This reproduction study confirms that in-memory checkpointing is a practical and highly effective technique for accelerating failure recovery in distributed training. Even in a scaled-down cloud environment with 4 A100 GPUs, we achieve substantial speedups that validate Gemini's core contribution.

References

[1] Wang et al., *Gemini: Fast Failure Recovery in Distributed Training with In-Memory Checkpoints*, SOSP 2023. `https://dl.acm.org/doi/10.1145/3600006.3613145`