CASE STUDY · RESEARCH REPORT · 22 NOVEMBER 2026

Why we parked our async checkpoint prototype

Exact restoration worked. Under concurrent training, our whole-copy GPU checkpoint prototype did not earn an advantage over cadence-tuned PyTorch DCP. Here is the experiment, the comparison that changed our decision, and what we kept.

Independent diagnostic, not a customer deployment. A 3.225M-parameter character language model on one RTX 3090, using PyTorch 2.9.1+cu126. The GPU encoder, CBP1, is a research prototype, distinct from the production CRX/CXR1 format. This experiment does not establish large-model or distributed performance. Crukx is not affiliated with or endorsed by PyTorch.

THE BET

Let training continue

Capture an immutable snapshot, release training, then compress and persist in the background. Fewer bytes could reduce I/O and keep a recent recovery point without holding up useful computation.

An earlier isolated staging fence of about 0.64 ms made the idea worth testing. It did not establish throughput under concurrent training. DCP already saves asynchronously; our test had to measure an advantage beyond that.

THE DECISION

Stop scaling this implementation

After adjusting DCP's requested cadence, it delivered higher training throughput, more durable checkpoints and a more recent recovery point. CBP1 also required substantial additional GPU memory.

We froze the whole-copy implementation rather than move it to larger models or multiple GPUs. Exact snapshot semantics, durability-age accounting and state-specific profiles remain useful research artifacts.

CORRECTNESS SURVIVED

43 restore checks. Seven fresh-process audits.

All 43 checkpointed runs passed full-state restoration and exact five-step continuation. Seven intermediate snapshots captured during concurrent training also passed fresh-process loading, replay comparison and five-step continuation.

Saved state included model, AdamW optimizer, scheduler, RNG and data cursor. Gradients were excluded at optimizer-step boundaries because the next step zeros them. These checks are not random rank-kill or power-loss tests.

MEMORY CHANGED THE ECONOMICS

3.08× additional GPU allocation

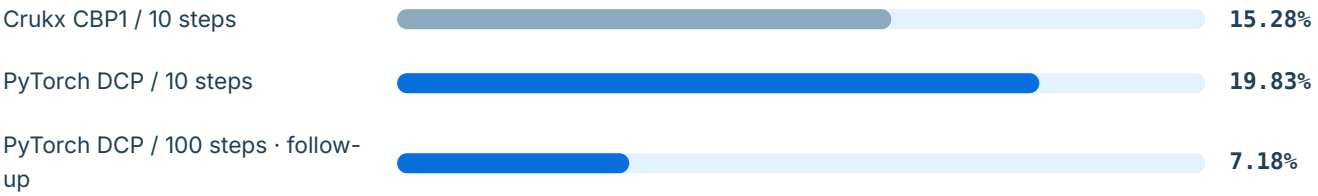
CBP1 added about 119 MB above the no-checkpoint peak, relative to a 38.70 MB checkpoint tensor payload. The no-checkpoint peak was 164.62 MB.

The 3.08× is additional peak PyTorch allocation divided by checkpoint tensor bytes, not total GPU memory. Whole snapshots, parent leases and scratch space all count. This implementation did not use copy-on-write or chunked snapshots.

THE COMPARISON THAT CHANGED THE DECISION

Same useful training work. Different recovery exposure.

8,000 measured steps per run, one seed, at an imposed 2,000 MB/s application write ceiling. Lower throughput loss and lower recovery age are better.



Measured long-run and adaptive follow-up results · 2,000 MB/s ceiling

SYSTEM / REQUESTED INTERVAL	STEPS/S	THROUGHPUT LOSS	DURABLE CHECKPOINTS	P95 RECOVERY AGE
Crukx CBP1 / 10 steps	122.67	15.28%	65	2.420 s
DCP / 10 steps	116.08	19.83%	275	0.707 s
Native sync / 10 steps	44.39	69.34%	800	0.256 s
DCP / 100 steps (follow-up)	134.24	7.18%	80	0.844 s

Why show both DCP cadences? At the original shared 10-step cadence, CBP1 had less throughput loss than DCP but produced 65 durable checkpoints versus 275, with older recoverable state. Faster training alone hid a recovery tradeoff.

After observing 65 CBP1 completions, we chose a 100-step DCP interval to approximate its actual checkpoint frequency. DCP then produced 80 checkpoints with better recency. This was an adaptive, single-seed follow-up with fresh bracketing controls, not an exhaustive sweep or an independent confirmatory trial. Recovery ages are better, not exactly matched.

Throughput loss uses the mean of each cohort's bracketing no-checkpoint controls: 144.797 steps/s for the long cohort, 144.628 for the follow-up. Counts include the final drain; throughput excludes it. Recovery age is elapsed time since the training state represented by the latest durable checkpoint.

METHODS AND BASELINE FAIRNESS

What we actually ran

Workload and controls

Four-layer character transformer, 3,225,153 parameters, real Shakespeare text, batch 8 × sequence 128. BF16 compute with FP32 saved weights and AdamW moments. One RTX 3090 with 24 GB VRAM.

53 timed runs: 42 primary runs across three seeds (1,000 measured steps), seven long runs, and four cadence-follow-up runs (8,000 steps). Each cohort/seed had bracketing no-checkpoint controls; primary arm order was randomized. These cohorts are not 53 independent replications.

Four configurations

No checkpointing; synchronous CPU snapshot + `torch.save` + fsync; actual PyTorch DCP async saving; and CBP1 GPU encoding with private snapshots and background persistence.

DCP used DefaultStager with staging overlapping forward/backward, then waited for staging before optimizer mutation. Both async arms allowed one in-flight checkpoint and skipped busy triggers. Rejected triggers were counted, not treated as durable saves.

Where the write ceilings came from

We imposed 2,000 / 1,000 / 500 / 100 decimal MB/s limits in the application writer, backed by real NVMe. The limiter waits according to bytes written in 4 MiB batches. These are maximum rates, not measured physical storage bandwidth or customer traces. DCP used a single writer thread with file synchronization; CBP1 used checksummed, fsynced publication.

What “cadence-tuned” means

We implemented a training-safe modern DCP path, then added the 100-step follow-up after inspecting the original results. We did not log engineering hours and cannot claim equal effort: more implementation work went into our prototype. This was a practical baseline adjustment, not exhaustive DCP optimization.

▼ Primary three-seed results and slower-storage follow-up

Median throughput loss at ceilings 2,000 / 1,000 / 500 / 100 MB/s: DCP 17.85% / 23.77% / 18.68% / 7.81%; CBP1 10.79% / 10.11% / 11.01% / 11.43%. Short CBP1 runs completed very few checkpoints, motivating longer runs rather than a throughput-only win claim.

At 100 MB/s in the 8,000-step cohort, DCP /10 lost 7.03% throughput, produced 66 checkpoints and had 1.732 s p95 recovery age. CBP1 /10 lost 10.97%, produced 14 and had 8.788 s age. DCP /100 in the adaptive follow-up lost 5.06%, produced 42 and had 2.167 s age. The downloadable CSV contains every timed run.

SCENARIO, NOT A MEASURED BILL

About 199 versus 80 GPU-hours wasted

Per 1,000 useful GPU-hours, an illustrative model put CBP1 /10 at about 199 and DCP /100 at about 80 under the 2,000 MB/s ceiling.

The model combines measured wall overhead, final drain, durable-state age and warm restore, assuming a whole-job mean time between failures of three hours. Wall overhead already includes stalls and contention. No customer failure rate, actual failure savings or billed GPU cost was measured. This tiny-workload scenario is not a scale forecast.

SCOPE AND LIMITS

What this does not establish

No multi-GPU or billion-parameter run, nvCOMP comparison, distributed commit, resharding, cross-hardware exactness or cold recovery. Follow-ups used one seed; the three-seed primary is not a population confidence interval.

The observed costs belong to this whole-copy implementation. They are not a fundamental bound on GPU compression or evidence that checkpointing is an unimportant problem.

RESEARCH DECISION · FROZEN 22 NOVEMBER 2026

Keep the evidence. Change the question.

We parked the current CBP1 checkpoint architecture, including larger-model scaling, multi-GPU expansion, nvCOMP integration and compression-ratio polishing. We preserved the harness, measurements, immutable-state contract, durability-age model and optimizer-state profiles.

Reopening requires a concrete customer workload and a materially different staging or memory design with a reason to beat a competent async baseline at comparable recovery exposure. The next test must count useful training, memory, restore costs and exactness together. Compression ratio alone is not a pass/fail gate.

Time and credit. The full-state and async research records span 21–22 November 2026; E2.1 is dated 22 November. Shafwan Safi directed the AI-assisted work with Codex implementation and orchestration. No reliable person-hour log exists, so we make no engineer-day claim.

The CSV and JSON publish aggregate metrics and hashes identifying the frozen local source records. They do not include raw model payloads or private codec implementation. Local experiment tag: `experiment/e21-2026-11-22-parked`.