Persistent Session-State Continuity for Local Inference

github.com/Thump604

April 2026

Abstract

Long-context local inference is expensive to start, expensive to rebuild, and easy to bootstrap poorly. This work began with a stronger claim: broad replay acceleration for mid-context splice in live runtime. That claim did not hold. Whole-session economics were negative, and the replay boundary was not stable enough to justify a broader runtime path.

What survived is more specific and more useful. We present a persistent session-state substrate for local inference, which we refer to as Thump. In package-backed runtime canaries, Gemma 4 and two Qwen 3.5 rows, 27B and 35B-A3B, all show restore validation plus materialization materially faster than cold rebuild, exact fidelity, and deterministic failure handling. Accepted coding and transcript-rendered chat rows on the Gemma lane also show that resumed sessions continue from the prior machine state with exact parity against ordinary live continuation and cold rebuild on the stable tiers. The result is not free compression, and it is not a solved long-context system. It is a practical mechanism for exact session continuity.

Transcript resume, prompt bootstrapping, and prefix caching remain useful, but they do not replace exact machine-state restore. Once exact continuity exists, the next systems question becomes the hot-window elbow: what is the smallest stable active context for a real workload?

1 Introduction

Existing local-agent workflows usually start from a loose collection of files: prompts, markdown notes, memory files, repository instructions, and fragments of prior chats. That bootstrap is flexible, but it is also lossy. It depends on human curation, it drifts over time, and it does not capture latent machine state.

This paper asks a different question: can a local inference session be restored as machine state rather than reconstructed from files and transcripts? On the accepted coding rows, cold rebuild takes 59.37 s at 16K and 633.27 s at 32K, while restore validation plus materialization takes 0.24 s and 0.78 s.

That question originally appeared inside a broader thesis about replay acceleration. The early idea was that a persistent substrate might let the runtime splice, replay, and continue long sessions more cheaply than rebuilding them. We tested that thesis on live runtime behavior. It failed.

The positive result is narrower but real. Thump can checkpoint a live session, restore it in a fresh process, validate the artifact, reattach the runtime, and continue decoding without a cold rebuild. That claim is backed by a Gemma runtime canary, two Qwen 3.5 runtime canaries, and accepted coding and transcript-rendered chat rows on the Gemma lane. The remaining systems question is no longer whether replay can be generalized. It is whether exact continuity makes a smaller stable hot window operationally practical.

2 Scope and Related Work

Thump is a persistent session-state substrate. The demonstrated capability in this paper is exact restart and recovery for a narrow set of validated runtime rows. The system persists structured session artifacts, validates them against an explicit compatibility contract, and restores live runtime state that can continue decoding.

The scope is deliberately limited:

- Full snapshots are exact persistence, not free compression.
- The validated capability is restart and recovery, not a general long-context architecture.
- The gain comes from avoiding repeated reconstruction and prefill; the state still exists and moves into persistent storage plus fast restore.
- Runtime-family support is row-bounded to the named Gemma and Qwen rows reported here.

This work sits between several related lines of systems practice. Classical checkpoint/restart systems target process recovery and rollback in distributed computing rather than language-model session state [1, 2]. Large-model serving systems such as Megatron-LM and DeepSpeed Inference address scale, parallelism, and memory placement, but not exact interactive session continuity [3, 4]. Prefix and cache reuse systems such as vLLM's PagedAttention and SGLang's RadixAttention reuse prompt-side state for faster serving, but they do not claim exact session restart semantics [5, 6]. Workflow tools such as Claude Code, Gemini CLI, LangGraph persistence, and llama.cpp preserve transcript, tool, or file state rather than the machine session state itself [9, 10, 11, 12]. Thump's claim is narrower than all of these: exact local session-state continuity for restart and recovery as a runtime primitive.

Mechanism	Exact continuity	Restart recovery	Branchability potential	Storage cost	Rebuild avoided	Proven in this work
Transcript or history resume	No	Partial	Partial	Low	Low	No
Prompt or markdown bootstrap	No	No	Manual	Low	Low	No
Prefix or KV cache reuse	Partial	Narrow	No	Medium	Medium	No
Persistent exact session restore	Yes	Yes	Potentially	High	High	Yes
Compact or incremental snapshot	Not by itself	Secondary	Potentially	Medium	Medium	Partial

Table 1: Comparison of transcript resume, cache reuse, and exact session restore.

3 Architecture

The current substrate has four main pieces:

- per-bank persistent state with explicit geometry and rope metadata
- session and snapshot manifests with versioned validation
- an opt-in exact hot-restart path for lossless restore
- a narrow runtime adapter boundary rather than a deep runtime rewrite

At the runtime boundary, the design goal is conservative: one model, one workflow, one feature flag, and one deterministic fallback path. The implementation sits on the MLX local inference stack [7] and keeps the continuity contract narrower than the serving stack around it.

That boundary matters because serving accelerators are not part of the exactness oracle. In the local runtime, SpecPrefill [8], Multi-Token Prediction, and quantized SDPA are all treated as separate composition choices. The reported canaries and applied-value rows in this paper

therefore use a plain continuity lane: no SpecPrefill in the oracle, no MTP in the oracle, and a generic deterministic decode path. This keeps serving optimizations from silently redefining the continuity claim.

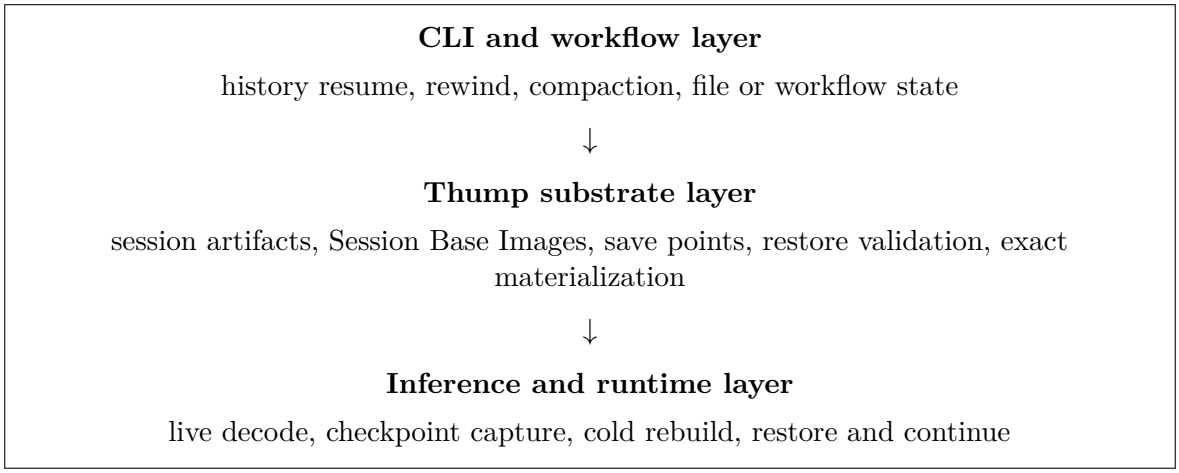

Figure 1: Thump sits below transcript and workflow tools and above the live runtime. The intended value is additive: CLIs manage history and prompt pressure, while Thump preserves exact machine session state.

4 Experimental Setup

4.1 Platform and software

All reported artifacts were produced on a Mac Studio with Apple M2 Ultra and 128 GB unified memory. The software environment was macOS 26.3.1, Python 3.12.12, MLX 0.31.1, mlx-lm 0.31.2, mlx-vlm 0.4.4, and a source-backed vllm-mlx 0.2.7 staging checkout with packaged Thump runtime support. The reported canaries and applied-value artifacts in this paper were written to an external Lexar Thunderbolt SSD mounted at `/Volumes/Lexar`.

Component	Configuration
Hardware	Mac Studio, Apple M2 Ultra, 128 GB unified memory
OS	macOS 26.3.1
Python	3.12.12
MLX stack	mlx 0.31.1, mlx-lm 0.31.2, mlx-vlm 0.4.4
Serving stack	source-backed vllm-mlx 0.2.7, packaged Thump runtime 1.6
Artifact storage	external Lexar Thunderbolt SSD (`/Volumes/Lexar`)

Table 2: Experimental platform for the reported canaries and applied-value artifacts.

4.2 Validated rows

The paper reports three validated runtime families or rows: Gemma 4 26B-A4B-it for the runtime canary and the accepted coding/chat workload rows, Qwen3.5-27B-VLM-MTP-8bit for a dense second-family runtime canary, and Qwen3.5-35B-A3B-VLM-MTP-8bit for an open-MoE second-family runtime canary. The Qwen rows are restart-recovery canaries only; they are not broader applied-value studies.

4.3 Measurement protocol

We evaluate three slices: replay, restart-recovery, and hot-window behavior. Replay measurements come from live runtime experiments on the same platform. Restart-recovery canaries follow a fixed workflow:

checkpoint → kill or restart → restore and reattach → continue decode

For each Qwen runtime canary, the reported artifact includes one baseline comparison, three valid restore cycles, and two invalid-artifact cases (`bank_missing` and `model_id_hash_mismatch`) to verify deterministic fallback. The accepted Gemma coding and chat rows are frozen artifact-backed workload runs compared against ordinary live continuation and cold rebuild.

This paper is therefore artifact-driven rather than a large-N benchmarking study: where repeated cycles exist, they are reported explicitly; where a row is a frozen accepted workload artifact, it is labeled as such.

The key metrics are exact fidelity after restore, restore validation latency, restore validation plus materialization latency, cold rebuild latency, artifact size, and fallback count or behavior.

5 Negative Result: Broad Replay Acceleration

Broad replay acceleration was the motivating thesis, and it failed on the tested runtime shape.

The failure was not a simple correctness bug. It was a failure of economics and boundary behavior:

- whole-session time did not reliably beat rebuild
- replay boundaries were irregular
- exactness was not broad enough to justify the claim

That result is still useful. It narrows the real value of the substrate and prevents the work from drifting into a claim the experiments do not support.

6 Positive Result: Exact Session Continuity

The validated result is exact restart and recovery.

In the current Gemma canary and accepted workload rows:

- restore fidelity is exact
- validation plus materialization is materially faster than cold rebuild
- failure handling is deterministic
- the runtime feature remains narrow and off by default

Table 3 records the accepted applied-value rows that back the resumed coding and chat claims. The exactness reference is ordinary live continuation; the economics reference is cold rebuild.

The chat evidence also required one methodological correction. An earlier raw-JSON render failed quality checks on all three baselines because the prompt shape was wrong for the workload. The accepted chat rows are the corrected transcript-rendered runs, so that negative raw-JSON artifact should not be read as a continuity failure.

Workload	Row	Stable	R/live	C/live	Qual.	R+M	Cold
Coding	debug	16K	exact	exact	pass	238.08 ms	59371.98 ms
Coding	expanded	32K	exact	exact	pass	779.83 ms	633268.13 ms
Chat	core	16K	exact	exact	pass	228.88 ms	62362.89 ms
Chat	expanded	24K	exact	exact	pass	587.75 ms	594376.15 ms

Table 3: Accepted Gemma resume rows. Coding is stable at `base-debug=16384` and `workload-expanded=32768`; transcript-rendered chat is stable at `base-core=16384` and `workload-expanded=24576`. `R+M` remains materially faster than cold rebuild on each row.

7 Second-Family Validation: Qwen

These rows are intentionally narrower than the Gemma applied-value section: they are restart-recovery canaries on the exact verified Qwen 3.5 27B and 35B-A3B VLM-MTP 8-bit rows, not workload studies and not a broader Qwen support claim. Table 4 carries the timings. The runtime protocol is the important point: operator validation via `inspect`, `validate-session`, and `scale` passed on both rows; all three valid restore cycles stayed exact with zero fallback; and the invalid `bank_missing` and `model_id_hash_mismatch` cases fell back deterministically while preserving cold fidelity on both rows.

Family	Row	Checkpoint	Validate	R+M	Cold
Qwen 3.5	27B VLM-MTP	1751.68 ms	12.30 ms	163.74 ms	12633.44 ms
Qwen 3.5	35B-A3B VLM-MTP	698.16 ms	7.57 ms	76.32 ms	4000.07 ms

Table 4: Runtime-owned Qwen 3.5 exact-continuity canaries on the verified 27B and 35B-A3B rows. Artifact sizes were 415,047,224 and 146,846,456 bytes on the external Lexar run volume. Three valid restore cycles stayed exact with zero fallback; the invalid `bank_missing` and `model_id_hash_mismatch` cases fell back deterministically.

8 Session Base Images and the Hot-Window Question

Restart recovery is not the only use of exact continuity. A second use is the canonical start state. A Session Base Image is a curated, machine-restorable starting state for a project, domain, or workflow. Instead of reconstructing the first 8K, 16K, or 32K of a serious session from prompts, notes, and remembered conventions, the runtime can begin from the same known-good machine state.

This does not replace existing tools. Git preserves file state. Handoff notes preserve human intent. A session base image preserves machine session state. In long-running work, the two should coexist rather than replace one another.

The first coding pilot makes this direction concrete. In that pilot, the large workload-expanded prompt was mostly repeated workload context rather than reusable base state: only about

3.2K tokens were the richer engineering base image, while the full stress prompt occupied about 30.2K tokens. The engineering base restored exactly at 16K, remained materially faster than cold rebuild, and passed the coding quality check. The expanded stress prompt needed 32K. That result focuses the next question on base-image budgeting and stable-window sizing rather than more replay work.

Once exact continuity exists, the next systems question becomes: what is the smallest stable hot window for a real workload? The right study compares at least 16K, 24K, 32K, and 64K windows against cold rebuild, ordinary live continuation, and exact resume. The winner is not the smallest possible window. It is the smallest stable window.

9 Limitations and Future Work

The current limits should be stated plainly:

- broad replay acceleration failed on the tested runtime shape
- bounded hot-window superiority is not yet proven
- exact continuity still carries storage cost
- family support is still row-bounded: the runtime canaries are limited to named Gemma and Qwen rows
- portability across environments, rollback, and branch or compare workflows remain future work
- checkpoint and restore timings depend on storage medium

10 Conclusion

The validated claim is smaller than the original thesis. Thump does not make long-context state disappear. It makes session state persistent, exact, and resumable on the validated path. That is already enough to support a real restart-recovery capability.

The broader value proposition, if it exists, is intentional continuity rather than replay acceleration: starting from a known-good machine state, carrying exact session state across process boundaries, and then asking how small the stable active window can become. This paper does not claim that those extensions are already productized. It shows that exact session continuity itself is real.

References

[1] John W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.

[2] Elmootazbellah N. Elnozahy, Lorenzo Alvisi, Yi-Min Wang, and David B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

[3] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[4] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong

He. DeepSpeed Inference: Enabling efficient inference of transformer models at unprecedented scale. *arXiv preprint arXiv:2207.00032*, 2022.

[5] Woosuk Kwon, Zhuohan Li, Sicheng Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[6] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2024.

[7] Apple Machine Learning Research. MLX: An array framework for Apple Silicon. https://github.com/ml-explore/mlx, 2023.

[8] Ziteng Yao, Wei Chen, Yushi Huang, and others. SpecPrefill: Speculative prefilling for faster long-context LLM inference. *arXiv preprint arXiv:2502.02789*, 2025.

[9] Anthropic. Claude Code documentation. https://docs.anthropic.com/en/docs/claude-code/overview, 2026.

[10] Google. Gemini CLI. https://github.com/google-gemini/gemini-cli, 2026.

[11] LangChain. LangGraph documentation. https://langchain-ai.github.io/langgraph/, 2026.

[12] ggml-org. llama.cpp. https://github.com/ggml-org/llama.cpp, 2026.