Equal-Time Intervening Work Raises HiCache Resume Latency in a Single-GPU Experiment

Ada Mao | GPT6-luna

Human-led research with AI assistance; AI contribution disclosed in manuscript.

Manuscript type: bounded systems measurement / mechanism study

Status: evidence-backed draft; not a validated predictive-scheduling algorithm paper

Experiment: SGLang HiCache, Qwen3-8B, NVIDIA A100-SXM4-80GB; 20 paired blocks

Abstract

Hierarchical KV caches can preserve an LLM request's prefix beyond GPU memory, but a successful prefix restore does not guarantee low continuation latency. We test whether unrelated serving work during an agent-like pause changes the latency of a cached continuation, separately from the effect of simply waiting. On one NVIDIA A100-SXM4-80GB running SGLang 0.5.3 and Qwen/Qwen3-8B, we ran 20 randomized paired blocks. Each arm seeded a unique 12,000-token prefix, applied six unrelated 14,000-token pressure requests, and submitted the continuation two seconds after the pressure phase. The idle arm waited during the interval; the treatment arm launched an unrelated cold prefill during it. All 40 continuation requests had per-request cache-restore verification in the raw artifact and embedded server-log excerpts. The treatment-minus-idle median paired TTFT difference was +1,318.912 ms (20/20 positive; paired-block percentile-bootstrap 95% interval [+604.540, +1,458.767] ms). The treatment and idle arm TTFT medians were 1,440.939 ms and 92.724 ms, respectively. Effects were positive in both randomized order strata, but their magnitudes differed. This is evidence that, in this constructed single-worker workload, launching unrelated cold work during a fixed pause can substantially delay a cached continuation even when load-back is observed. It does not isolate GPU queueing from prefill overlap or other server-state pathways, and it does not evaluate predictive prefetch, an agent scheduler, production SLOs, or generality across systems.

1. Introduction

Multi-turn agents alternate model inference with external tool calls. During a tool pause, a serving engine may retain request state in GPU memory, move it to host memory, or evict it. When the agent resumes with an exactly matching token prefix, a hierarchical cache can restore reusable KV state rather than recompute the prefix. Yet a cache-hit signal alone may not predict the resumed request's observed time to first token (TTFT): the server may also be processing other requests, and the restore path may be sensitive to recent serving activity.

This study asks a narrow causal question: when the continuation is submitted at the same elapsed time after cache pressure, does launching an unrelated cold prefill during the pause change continuation TTFT relative to leaving the server idle?

This is a measurement study, not a new scheduling algorithm. Prior systems already study agent-execution prediction and proactive prefetch, tool-aware cache TTL, and queue-aware cache placement. We make no novelty claim for those mechanisms. The contribution here is a paired equal-duration control that distinguishes waiting from submitting unrelated work in one reproducible engine configuration.

Contributions

1. A randomized, 20-block comparison of idle waiting versus launching one unrelated cold prefill during an otherwise fixed two-second interval.

2. Per-arm cache evidence combining reported cached-token counts with server load-back log excerpts embedded in the raw result artifact.

3. A bounded finding: the intervening-work arm has materially higher continuation TTFT in this test, while the result does not identify the internal latency component or validate a predictive cache policy.

2. Relation to prior work and scope

CacheScout studies learning agent-execution transitions to support cache management and proactive prefetch; Continuum studies tool-aware KV-cache TTL and reload/queueing tradeoffs; Vesta studies queue-aware scheduling with cache locality and transfer costs. These works bound the novelty claim: agent prediction, prefetching, TTL retention, and joint cache/queue-aware scheduling are not contributions of this manuscript.

SGLang's HiCache provides hierarchical KV-cache functionality. In this experiment, engine-reported `cached_tokens` is treated as evidence of prefix reuse, not as a measurement of transferred bytes or a standalone proof of which memory tier supplied the KV. The raw experiment additionally records per-request `loading back ... tokens` log excerpts and a restore-verification flag. We therefore describe the tested event as an observed HiCache load-back, while acknowledging that the complete original server log is no longer available for independent line-by-line replay.

This study is distinct from the earlier “prefetch” timing runs in this project. Those runs used an ordinary generation request as a prefill surrogate. They did not invoke a verified proactive HiCache host-to-GPU transfer operation and are not evidence for predictive prefetch.

3. Hypothesis and experimental design

3.1 Hypothesis

Let Y_{b,I} be continuation TTFT for the idle arm in paired block b, and Y_{b,W} be continuation TTFT when unrelated cold work is launched during the pause. The primary paired outcome is

\[

D_b = Y_{b,W} - Y_{b,I}.

\]

Positive values indicate that the intervening-work assignment is slower. Before this run, the protocol specified a practical-effect gate of 100 ms: support for a practically material penalty requires the paired-median 95% interval's lower endpoint to exceed 100 ms and restore verification to succeed in at least 95% of both-arm observations.

3.2 Serving setup

The raw artifact records:

Item	Configuration
Model	Qwen/Qwen3-8B
Serving engine	SGLang 0.5.3
GPU	NVIDIA A100-SXM4-80GB
PyTorch / CUDA build	2.8.0+cu128 / 12.8
Independent unit	Paired session block
Blocks	20
Target prefix	12,000 tokens
Pressure	6 requests × 14,000 tokens per arm
Intervening request	One unrelated cold 12,000-token prefill
Pause window	2.0 seconds from the post-pressure interval start to continuation submission
Random seed	20261023

The server was configured for hierarchical cache with a 48-GB host pool, `write_through`, cache reporting, and debug logging, as recorded in the run configuration and earlier server setup. Prompts and tool-like contents were constructed by the experiment script; they are not a public agent trace, framework trace, or production workload.

3.3 Procedure

Each arm used its own unique namespace. Within that arm, the seed prompt and continuation shared an exact prefix as verified with the local tokenizer. After seeding, the script issued six unrelated pressure prompts. It then began the fixed two-second interval:

- Idle: issue no request during the interval.
- Intervening cold: launch one unrelated cold prefill immediately during the interval.

At the end of the interval, submit the continuation in both arms. If the unrelated prefill was still running, the continuation was submitted on schedule and could overlap it. Thus the treatment is the operational action “launch unrelated cold work during the pause,” including any remaining overlap at resume time; it is not a comparison that forces the background request to finish before continuation.

Arm order was randomized within each pair. The realized allocation was seven `idle` → `intervening_cold` blocks and thirteen `intervening_cold` → `idle` blocks. Different target prefixes prevent direct target-prefix reuse across arms, but global server/cache/load state can still carry over; order-stratified results are therefore reported.

3.4 Outcomes and cache verification

The primary outcome is client-observed streaming TTFT, from request submission to the first nonempty generated text event. The raw artifact records per-arm continuation TTFT, cached-token counts, server log evidence, restore-verification state, request metadata, and intervention timing/completion information.

All 20 idle and 20 intervening-work continuations were marked `restore_verified`. The raw artifact contains 20 load-back log excerpts for each arm; the separate independent audit reports 40 extracted `loading back ... tokens` lines in total. Observed cached-token counts were approximately 12,001–12,003 per continuation.

3.5 Statistical analysis

The independent unit is the paired block, not an individual request or generated token. The estimator is the median of the 20 paired differences D_b. The 95% interval is a percentile bootstrap over paired-block differences, resampling blocks with replacement; the independent audit used 20,000 replicates and seed 20261024. No blocks were excluded from the complete paired analysis. We also report the effect by randomized condition order because the sample is not evenly split and the magnitudes differ.

4. Results

4.1 Primary paired result

Quantity	Result
Complete paired blocks	20
Idle-arm median continuation TTFT	92.724 ms
Intervening-cold-arm median continuation TTFT	1,440.939 ms
Median paired difference, intervening − idle	+1,318.912 ms
Paired-block bootstrap 95% interval	[+604.540, +1,458.767] ms
Positive / zero / negative paired differences	20 / 0 / 0
Restore verified, idle	20/20 (100%)
Restore verified, intervening cold	20/20 (100%)

The interval is entirely above the predeclared +100-ms practical-effect threshold, and both-arm restore verification exceeds the 95% gate. The result supports the directional hypothesis for this tested configuration and workload.

4.2 Randomized-order strata

Condition order	Blocks	Median paired effect	Positive pairs
Idle → intervening cold	7	+1,454.447 ms	7/7
Intervening cold → idle	13	+849.975 ms	13/13

The sign is consistent across both order strata, while the magnitude differs by roughly 604 ms. This is compatible with order/carryover or changing system state modifying the size of the effect. With only seven blocks in one stratum and no direct queue/transfer telemetry, it is not possible to explain this heterogeneity or claim an order-invariant effect size.

4.3 Integrity and provenance checks

The downloaded raw JSON reports `status: complete`, 20 blocks, and no invalid or unverified rows in the independent audit. The SHA-256 recomputed from the downloaded raw JSON matches the hash recorded in the audit: `1b4641e0befc7ff92b918e8d1fa9d26c3e0e4edb1138c3c98d8de33a6f5f4500`. The run records two tokenizer-construction failures from the interrupted attempt at block 3; the resumed complete block is present in the final 20-pair data, and those failures did not contribute an outcome to the paired analysis.

The complete original SGLang server log could not be recovered. The independent audit JSON preserves 40 extracted load-back lines and the original log's reported SHA-256, but the full source log is absent. Consequently, a reader can inspect the preserved excerpts and per-arm evidence fields, but cannot recompute the source-log hash or audit surrounding log context from the archived files.

5. Discussion

The result shows that a verified load-back and low continuation TTFT are different properties. In the tested setup, the continuation still reported roughly 12k cached tokens and a load-back event in both arms, yet continuation TTFT was much higher when an unrelated cold prefill had been launched during the pause. The manipulated action may affect queueing, GPU scheduling, prefill overlap, memory/allocator state, or multiple pathways. This experiment measures the end-to-end effect of that action, not its decomposition.

For serving-system design, the bounded implication is that predicting prefix reuse alone is not enough to predict resumption latency: concurrent work and recent system activity can matter. This motivates queue- and load-aware cache-management research. It does not show that a particular scheduler can avoid the penalty, nor that a cache-aware router, speculative prefetch, or demotion policy improves user-visible outcomes.

The manuscript's result is a mechanism/interference characterization in one system configuration. It is not a broad empirical claim about all agent workloads. A workshop or systems-measurement venue may find the controlled result useful if the contribution is framed narrowly and the raw artifact/code are made available; novelty and venue fit still require an independent literature and reviewer assessment.

6. Limitations and threats to validity

1. One configuration: one A100, one Qwen model, one SGLang version, and one serving worker. No cross-hardware, cross-model, or multi-worker validation was run.

2. Constructed workload: prefixes and unrelated requests were generated by a script, not collected from a real agent framework or public trace. The two-second pause is a fixed experimental condition, not an empirical tool-return distribution.

3. No component decomposition: TTFT includes client-observed serving delay. Queue wait, prefill duration, GPU execution, host-to-device transfer time/bytes, and network time were not separately instrumented.

4. Treatment includes overlap: the unrelated prefill may remain active when the continuation is submitted. The estimand is the effect of launching it during the pause, not the isolated cost of a completed prefill followed by idle time.

5. Order sensitivity: effect magnitudes differ by randomized order, and no between-arm global-state reset was performed. All signs agree, but the pooled effect magnitude should not be generalized as invariant.

6. Server-log archival gap: row-level extracted evidence is present in JSON, but the complete original server log is missing. The log's recorded hash is not independently reproducible without that file.

7. Exploratory research history: several prior surrogate and mechanism pilots informed this experiment. The final comparison is predeclared in its run artifact, but the overall research program is not a single untouched confirmatory trial.

8. No algorithm baseline: no CacheScout, Continuum, Vesta, or comparable scheduler implementation was run. This study does not establish algorithmic novelty or superiority.

7. Conclusion

In a 20-block paired A100/SGLang experiment, launching an unrelated cold prefill during a fixed two-second pause increased continuation TTFT relative to idle waiting. The median paired increase was 1.319 seconds, the paired-bootstrap interval was 0.605–1.459 seconds, every paired difference was positive, and all 40 continuations had recorded cache-load-back verification. The result is meaningful but deliberately narrow: it demonstrates a serving-state sensitivity of cached continuation latency for a constructed workload on one configuration. It does not validate predictive agentic KV-cache scheduling, a new algorithm, production SLO benefits, or generality. No further GPU run is needed to report this bounded result; additional work to validate a predictive scheduling policy would be a separate research project.

Reproducibility appendix

Archived artifacts

- Raw 20-block result: `hicache_equal_time_final_20261018T012901Z.json`
- Independent audit and extracted server-log evidence: `hicache_equal_time_final_independent_audit.json`
- Reanalysis code: `audit_hicache_equal_time_final.py`
- Experiment runner: `run_hicache_equal_time_final.py`

The raw result's SHA-256 is `1b4641e0befc7ff92b918e8d1fa9d26c3e0e4edb1138c3c98d8de33a6f5f4500`. The audit JSON's SHA-256 is `3be2734adf6cb937ebe2bff418cda84cdf706a97609b024b5e2f88e88e92473e`. These are local archival copies of the user's downloaded files.

To independently recompute the paired estimate from the raw file without contacting the inference server:

```
python scripts/audit_hicache_equal_time_final.py \
  research/agentic_kv_cache_scheduling/results/hicache_equal_time_final_20261018T01290
  1Z.json \
  --output /tmp/hicache_equal_time_final_reaudit.json
```

The complete original server log is not archived. The audit artifact stores the 40 extracted load-back lines and the source-log hash supplied by the Pod-side audit. The data are constructed prompts and should not be described as a production or public agent trace.

References

1. Rui Zhang et al. “Learning Agent Execution for KV-Cache Management in Agentic Serving.” arXiv:2608.14624, 2026.

2. Hanchen Li et al. “Continuum: Efficient and Robust Multi-Turn LLM Agent Scheduling with KV Cache Time-to-Live.” arXiv:2511.02230, 2025/2026 version.

3. Lancheng Zou et al. “Vesta: Queueing-Aware LLM Scheduling for Real-World Workloads.” arXiv:2610.14686, 2026.

4. SGLang. Hierarchical KV Caching (HiCache) documentation.