
A Host-RAM Tier for the KV Cache: Up to 33× Sooner After Eviction, 16% of Cache Misses Served in Production

Yury Akinin (author) · Dmitrii Lozhkin (reviewer) · AVELIN AI · October 14, 2026

Abstract

An inference server that keeps the key-value (KV) cache of recent prompts in GPU memory must recompute a long conversation once its blocks are evicted. Copying each block to a pool in host RAM as it is produced, and back on demand, removes the recomputation of whatever the pool still holds. We report an eviction test on two setups and 26 hours of production on one H200 server with a 1M-token context. In the tests an evicted conversation of about 200K tokens yields its first token after 1.37 s and 0.78 s instead of 44.62 s and 12.37 s, 33 and 16 times sooner, and the three restored requests returned what they were asked for. In production (13,965 requests, 1.78 billion prompt tokens) 85.5% of prompt tokens came from the GPU cache, 2.3% from the RAM pool and 12.1% were computed: the pool supplied 41 million tokens in 517 restores, 16% of what the GPU cache had missed. The restores took 24 s of transfer in total and all writes to the pool 122 s, 0.13% of the period. There was no run without the pool, so the time it saved is modelled rather than observed.

1 Introduction

Agentic sessions resend a growing prompt at every turn. Servers avoid recomputing it by reusing attention state: the KV blocks of earlier prompts stay in GPU memory and are reused when a new prompt begins with the same tokens [1, 2]. The GPU memory left after the weights bounds this cache. On our production server it holds 2.1 million tokens, two full context windows. When a conversation is evicted, its next turn is a cold read, which takes tens of seconds for a few hundred thousand tokens.

Host RAM is the natural second tier: a GPU server usually carries more RAM than GPU memory, and serving leaves much of it idle. Research systems that keep conversation state in host memory or on disk between turns report large gains on replayed traces [3, 4], and KV state has also been compressed and shared across machines [5, 6]. The server we use ships a built-in variant: a pool in host RAM that receives blocks as they are produced and returns them on a miss. We add what such a stock tier supplies under live traffic, and what it costs.

2 Setup

Mechanism. The server stores KV state in blocks (1,600 tokens each on the production server). With offload on, blocks are also copied into a shared-memory pool in host RAM as they are produced: in production the bytes stored equal, within 0.4%, the KV of every computed prompt token and every generated token. The pool evicts when it fills. On a prefix miss in GPU memory the server looks the blocks up in the pool and copies them back; only what neither tier holds is computed. The server counts prompt tokens by where they came from and reports transfer volume and time every ten seconds.

Eviction test. A conversation of about 200K tokens with three codes hidden in it is read cold, repeated at once, pushed out of GPU memory by 10 (A) or 13 (C) other conversations of the same length, whose total exceeds the GPU pool, and requested again. We record the time to first token (TTFT), the bytes copied and whether the three codes come back. Setup A is a 96 GB workstation GPU with a 27B dense model, a GPU pool of 1.77 million tokens and a 100 GiB RAM pool; setup C is the production server. Both servers were otherwise idle, and each test ran once.

Production. Setup C served internal agentic sessions: an H200 with a 1M-token context, a GPU pool of 2,113,003 tokens (55.9 GiB) and a RAM pool of 110 GiB, which is 2,688 blocks or 4.30 million tokens, 2.0 times the GPU pool, on a host with 16 vCPUs and 197 GiB of RAM. The GPU holds 76.6 GiB of weights, 4-bit integers for the expert matrices, beside the KV pool; an embedding table of the model, about 47 GiB, is kept in pinned host memory. The observation period is the 26.0 hours from the start of the server’s counters to our snapshot; traffic arrived in 17 of those clock hours. It covers 13,965 finished requests with a mean prompt of 127K tokens, 15% of them above 200K and 4% above 500K, and up to 13 running at once. Totals come from the cumulative counters. Restore sizes come from the ten-second reports, which show 509 of the 517 restores. A restore is sized as the blocks it moved

minus four, the number the server moves on top of the tokens it serves (131 blocks for 127 in the eviction test; no restore was smaller than five blocks); so sized, the restores sum to 99.7% of the counter. To express a restore in seconds we use four cold reads timed on the same machine and model two days earlier (200K: 11.8 s, 500K: 31.3 s, 889K: 58.9 s, 981K: 66.0 s), interpolated linearly and, below 200K, extrapolated through the origin.

3 Results

Table 1: Eviction test. A conversation is read cold, hit again while still in GPU memory, evicted, and requested again. One run per setup; model sizes are parameter counts.

GPU	Model	Weights	KV dtype	Prompt, tokens	Time to first token, s			Sooner than cold	Copied, GB	Codes found
					cold	in GPU	from RAM			
A Workstation, 96 GB	27B dense	NVFP4	FP8	184,205	44.62	1.24	1.37	32.6×	6.51	3/3
C H200, 141 GB	180B MoE	4-bit integer	BF16	205,055	12.37	0.72	0.78	15.9×	5.75	3/3

An evicted conversation is back in about a second. After eviction the first token arrives in 1.37 s (A) and 0.78 s (C), against 44.62 s and 12.37 s for the cold read: 33 and 16 times sooner (Table 1, Figure 1a), and 0.13 s and 0.06 s later than a hit in GPU memory. The restores moved 6.5 and 5.8 GB. All three restored requests (two in A, one in C) returned the three codes. These ratios belong to prompts of about 200K tokens on an idle server: a hit itself takes about a second, so a shorter prompt gains less.

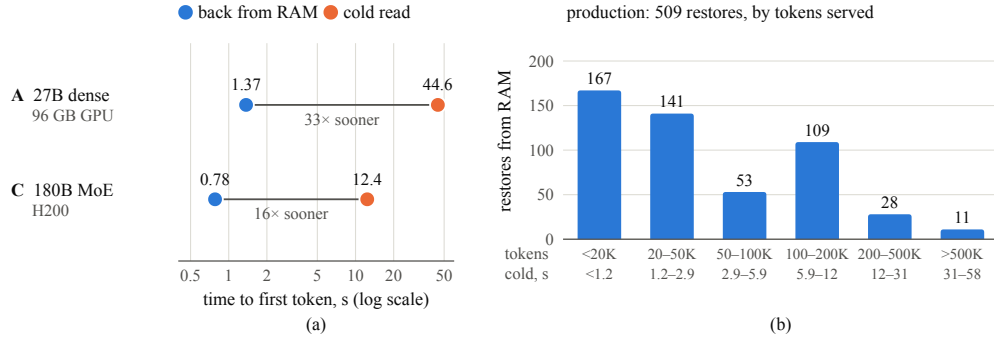


Figure 1: (a) Time to first token for a conversation of about 200K tokens: read cold, and returned from host RAM after eviction from GPU memory. (b) Production restores by tokens served (509 of 517 are visible in the server’s reports). The second label row is the modelled cold-read time for that size, extrapolated below 200K.

Table 2: Production counters of setup C over 26.0 hours: the source of prompt tokens, and the traffic between GPU memory and the RAM pool.

Prompt tokens by source	Tokens, M	Share, %	Pool traffic	Count	Volume, TB	Time, s
GPU cache	1,523.6	85.5	Restores, RAM to GPU	517	1.23	24.1
Host-RAM pool	41.2	2.3	Stores, GPU to RAM	70,977	6.28	122.4
Computed	216.4	12.1	Lookups	25,743	–	0.55
All prompt tokens	1,781.2	100.0	Deferred lookups	21	–	87.0

In production the pool supplies 2.3% of all prompt tokens, 16% of the misses. Of 1.78 billion prompt tokens, 85.5% were already in GPU memory, 2.3% came from the RAM pool and 12.1% were computed (Table 2). The pool therefore supplied 41.2 of the 257.6 million tokens that missed the GPU cache, 16.0%; without it they would have been computed, if nothing else changed. There were 517 restore operations, at most 3.7% of requests and about 30 per clock hour with traffic, averaging 80K tokens.

A few large restores carry the volume. The median restore serves 29K tokens, but 148 restores of at least 100K tokens carry 77% of the restored tokens and 11 exceed 500K (Figure 1b). The largest served 882K tokens: 24.4 GB moved in 0.48 s, for a prompt that takes about 58 s to read cold. Modelled cold reads for the visible restores sum to 41 minutes of prefill, 63% of it extrapolated below 200K; the transfers took 24 s, at 51 GB/s. The median restore, about 1.7 s of modelled cold read, gains little once the time of a hit is counted.

Transfer time is small. Blocks are written as they are produced: 70,977 store operations moved 6.28 TB in 122 s, 0.13% of the period. The 25,743 pool lookups took 0.55 s in total, 21 μ s each. The server deferred 21 lookups; the time from deferral to resolution, or to the end of the request, sums to 87 s, and 17 of the 21 took under 5 s. We measured transfer time, not its interference with computation.

What the pool does not do. It does not shorten a first read: 12.1% of prompt tokens were still computed, and 3.8% of first tokens took more than 20 s, while 76% took under one second. Nor is it unbounded: 6.3 TB passed through a 118 GB pool, 53 times its size, so blocks leave RAM too, and a restart empties it.

4 Limitations

The production data are one server, one workload and 26 hours, with the pool always on and empty at the start. The 16% is the share of misses the pool supplied; we did not run the same traffic without it, and the time it saved is modelled from cold reads on an idle machine, extrapolated for most restores. Restore volume is bursty: two clock hours carry 53% of it, and a few long sessions dominate. Restore sizes rest on the four-block correction and on an even split of the 24% of restores that share a ten-second report with another. The eviction test is one run per setup, on different models and GPUs, with three restored requests in all; a third run, on a model that fails the code check on cold reads, is excluded. The answer check is three codes, not a quality evaluation. We did not vary the pool size, so we cannot say how the share grows with it.

5 Recommendations

1. If host RAM is idle, give it to the KV cache, and leave headroom: on the production host a 120 GiB pool left 0.5 GiB of RAM free, while 108 and 110 GiB left 12 and 10 GiB.
2. Check shared memory first. The pool is a file in the shared-memory filesystem; on one host a 120 GiB pool failed to start because that filesystem held 107 GiB. Size the pool after the previous server process has exited: measured too early, the free space once gave us a 1 GiB pool.
3. Judge the tier by where prompt tokens come from. The pool’s usage gauge reports blocks held by transfers in flight; it read 0.3% here after 6.3 TB had passed through the pool.
4. Test with eviction, on a model that passes the check cold: fill more than the GPU pool, then ask for the first conversation again. Expect an empty pool after every restart.

Disclosure. Tests and analysis were produced with AI coding agents under the author’s direction. Production counters were read from the live server without changing it. All numbers were recomputed from the raw records.

References

- [1] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023. arXiv:2309.06180.
- [2] I. Gim, G. Chen, S. Lee, N. Sarda, A. Khandelwal, and L. Zhong. Prompt Cache: Modular attention reuse for low-latency inference. In *MLSys*, 2024. arXiv:2311.04934.
- [3] B. Gao, Z. He, P. Sharma, Q. Kang, D. Jevdjic, J. Deng, X. Yang, Z. Yu, and P. Zuo. Cost-efficient large language model serving for multi-turn conversations with CachedAttention. In *USENIX ATC*, 2024. arXiv:2403.19708.
- [4] L. Yu, J. Lin, and J. Li. Stateful large language model serving with Pensieve. In *EuroSys*, 2025. arXiv:2312.05516.
- [5] Y. Liu et al. CacheGen: KV cache compression and streaming for fast large language model serving. In *SIGCOMM*, 2024. arXiv:2310.07240.
- [6] Y. Liu, Y. Cheng, et al. LMCACHE: An efficient KV cache layer for enterprise-scale LLM inference. arXiv:2510.09665, 2025.