How Far Can PIMs Replace GPUs in Energy-Efficient Multi-Turn Agent Serving?

Junwha Hong
Agency for Defense Development

Guseul Heo
KAIST

Hyunsu Ye
KAIST

Jongse Park
KAIST

Olivia Hsu†
Carnegie Mellon University

***Abstract*—Multi-turn agent workloads reuse an expanding context across turns, allowing prefix caching to reduce redundant prefill. The resulting decode-heavy execution is well suited to Processing-In-Memory (PIM), but replacing GPUs with PIM also reduces prefill compute and GPU memory for reusable KVs.**

We characterize GPU-to-PIM replacement across six agent workloads using a disaggregated serving model. Our results show that decode share alone does not determine how many GPUs can be replaced. The replacement limit instead depends on fresh-prefill volume and the live-session KV working set across tool gaps. We further evaluate Session-KV residency, which retains reusable KVs on PIM to avoid recomputing GPU-evicted prefixes. Session-KV residency reduces energy by up to 78% and latency by up to 67% relative to the same GPU-PIM configuration without residency. These results call for provisioning GPU-PIM pools via fresh-prefill volume and live-session KV working set rather than by decode share alone.

***Index Terms*—LLM serving, agent workloads, prefix caching, processing-in-memory, energy efficiency.**

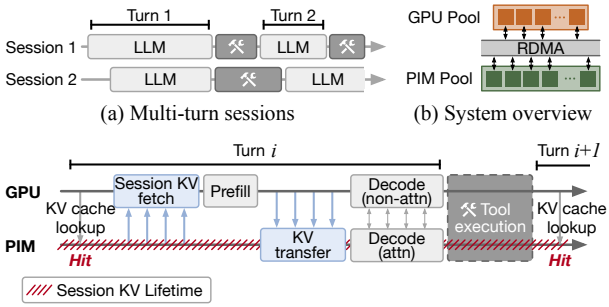
I. Introduction

Agentic LLM serving introduces a stateful, multi-turn execution pattern distinct from independently served LLM requests. As illustrated in Figure 1 (a), an *agent session* consists of multiple *turns*. In each turn, the LLM processes the context accumulated from preceding turns and may invoke an external tool. The interval during which the LLM waits for a tool result is a *tool gap*.

A later turn typically consists of a prefix shared with the preceding turn and a short suffix containing newly introduced context. Reusing the cached KVs of this shared prefix allows the system to prefill only the new suffix [1], whereas decode attention must still process the entire accumulated context. Multi-turn agent serving thus becomes decode-heavy.

Since decode attention reads the KV cache at every decoding step and is limited by memory bandwidth, Processing-In-Memory (PIM) [2], [3], [4] is a natural candidate for energy-efficient decode attention. We study the disaggregated GPU-PIM system (Figure 1 (b)), where GPUs execute prefill and non-attention operations while an independently provisioned PIM pool executes decode attention. Moving decode attention from the GPU suggests that PIM tiers could replace part of the GPU pool. However, shrinking the GPU pool also reduces both GPU compute for prefill and GPU HBM for storing reusable KV caches. We ask whether a high decode share is sufficient for GPU-to-PIM replacement once both resources shrink.

We observe that a high decode share alone does not determine how far PIMs can replace GPUs. Instead, GPU replaceability is governed by two sources of prefill on the reduced GPU pool. The remaining GPUs must perform *fresh prefill* for initial session inputs and context appended in later turns. Reduced GPU HBM may also evict reusable KVs, introducing *recomputation prefill* when a later turn reconstructs the evicted prefix.



(a) Multi-turn sessions (b) System overview

(c) Execution path for a session

Fig. 1. Overview of multi-turn agent serving on a disaggregated GPU-PIM pool.

We characterize this replacement limit across six agent workloads. Two workload properties govern it. The *fresh-prefill volume* sets the irreducible prefill demand on the remaining GPUs, and the *live-session KV working set* determines whether reusable KVs survive tool gaps under reduced GPU HBM. Even across decode-dominated workloads, the number of GPUs that can be replaced varies with these two properties.

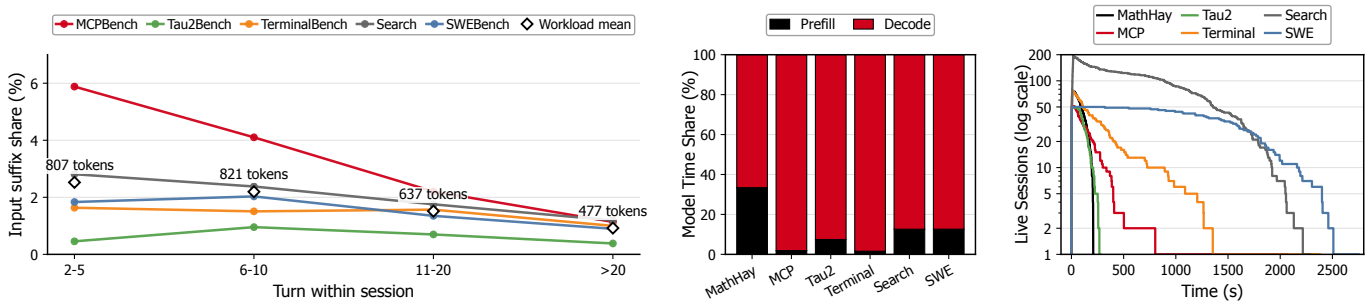
We evaluate whether retaining session KVs in PIM can prevent this recomputation. Under *Session-KV residency*, each session's KVs stay on its assigned PIM device after a turn completes, and a later turn retrieves a GPU-evicted prefix from PIM instead of recomputing it on the GPU (Figure 1 (c)). When the evicted state fits in PIM, this policy reduces energy by up to 78% and latency by up to 67% relative to the same GPU-PIM configuration without residency. The achievable benefit depends on the live-session KV working set.

This paper makes the following contributions:

- Characterization of the compute and memory demands of multi-turn agent workloads.
- Analysis of why decode share alone fails to predict latency, energy, and EDP.
- Evaluation of *Session-KV residency*, which retains session KVs on PIM across tool gaps to reduce prefix recomputation.

We hope this study motivates full-serving evaluation and dynamic provisioning of GPU-PIM pools under changing resource demand.

†Corresponding author at owh@cmu.edu. This work was supported by the Agency for Defense Development (912A45701, 50%) and IITP grant funded by the Korea government (MSIT) (RS-2026-25521385, 50%).



(a) Input suffix share across later turns. (b) Prefill and decode time. (c) Live sessions over time.

Fig. 2. Workload observations from agent traces and GPU-only baseline runs. Agent sessions exhibit high reuse of prior contexts **(a)**, shift model time towards decode **(b)**, and keep many sessions alive across tool gaps **(c)**. Panels (b) and (c) use tool gaps from the traces as-is (upper bound).

TABLE I
GENERALAGENTBENCH WORKLOAD CHARACTERISTICS

Benchmark	Sessions	Turns/session	Input	Output
MathHay	75	1.0	130.6K	440
MCPBench	51	10.7	30.1K	459
Tau2Bench	50	14.0	27.2K	96
TerminalBench	78	30.0	35.7K	296
Search	199	24.3	38.0K	138
SWEBench	50	88.8	45.7K	249

II. Characterization of Agent Workloads

A. Methodology

We study six GeneralAgentBench [5] trace families from Claude Sonnet 4.5 with different system prompts and agent behavior (Table I). *MathHay* inputs are scaled down to fit a 131K-token context window. We simulate serving these workloads with Qwen3-30B-A3B on four H100 TP=1 GPU-only instances using LLMServingSim [6], a real-profile-driven framework. Request placement follows the session-affinity policy used by Autellix [7], assigning each request to the GPU instance that previously served the same session. Following Autellix, the first request of each session arrives according to a Poisson process. We set the session arrival rate to 10.0 sessions/s. The simulator releases subsequent requests of each turn after the traced tool gap elapses. The recorded tool gaps of GeneralAgentBench include preceding LLM inference time, so we report results using the traces as-is (upper-bound) and a best-effort correction based on the mean reconstructed inference time (lower-bound).

B. Agent Workload Observations

We observe that multi-turn agent workloads exhibit high prefix reuse, shifting model time toward decode. Tool gaps keep many sessions live concurrently, increasing contention for limited KV-cache capacity.

Figure 2 (a) shows that later turns append only a short input suffix. The mean suffix across the five multi-turn benchmarks is 807 tokens at turns 2-5 and 477 tokens beyond turn 20, or 2.52% and 0.92% of the input context. *MCPBench* appends the largest share and stays below 6% at every turn range.

Figure 2 (b) then shows how this prefix reuse shifts model time toward bandwidth-bound decode. On the GPU-only baseline, decode consumes 88.1–89.2% of model time across the six benchmarks. *MathHay* has a relatively high prefill ratio (33.4%). Its single-turn sessions cannot exploit cross-turn reuse, although they reuse long cross-session prefixes spanning both the system prompt and the shared document sequence.

Finally, Figure 2 (c) shows that tool gaps keep many sessions live concurrently while their prior KVs remain reusable. Across the five multi-turn benchmarks, tool gaps account for 50–62% of total session lifetime, reaching 88–91% for *Tau2Bench*. These gaps require the memory system to preserve session KVs, rather than only serve the active batch. In *Search*, up to 197 of 199 sessions remain live concurrently. *MathHay* is single-turn and has no tool gaps.

III. System Model

We study a disaggregated GPU-PIM system that offloads decode attention to PIM and evaluate a configuration that uses PIM memory as a second-tier prefix cache for agent sessions.

GPU-PIM heterogeneous pool. Recent CXL-PNM systems scale PNM capacity within a CXL fabric, including rack-scale deployments [8]. We instead model PIM as a network-disaggregated memory-side compute tier connected to GPU servers over RDMA, allowing GPU and PIM resources to be provisioned independently. Each PIM node contains multiple host-visible PIM devices that share an RNIC port. To our knowledge, no existing PIM prototype has demonstrated such a disaggregated RDMA setup. However, prior PIM prototypes [9], [10] show that near-memory compute can run behind a standard host-visible interface. We further assume that the host-visible PIM memory is pinnable and registerable as an RNIC memory region.

GPU-PIM execution model. A GPU-side scheduler performs continuous batching and chunked prefill. Before prefill, the scheduler assigns each request to the least-loaded PIM device. The GPU transfers newly generated KVs to the assigned PIM device, overlapping communication with GPU computation whenever possible. The request joins the decode batch once all its KVs are available at the PIM device. At each decode iteration, the GPU performs the QKV projections, and the

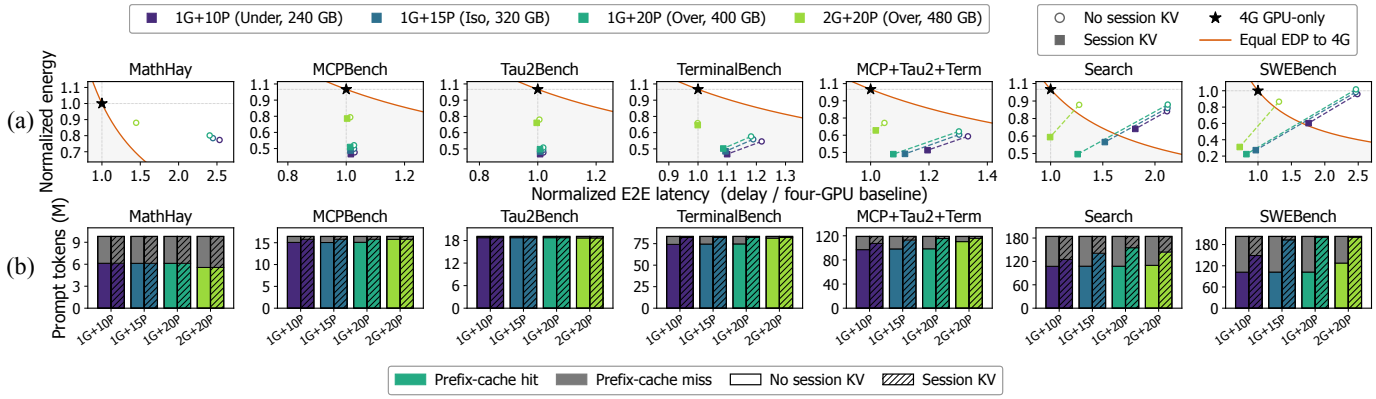


Fig. 3. GPU-to-PIM replacement across capacity-ordered configurations. **(a)** Energy and end-to-end latency normalized to each benchmark's four-GPU-only baseline (star). Dashed lines connect configurations without (open) and with *Session-KV residency* (filled). The amber curve shows equal EDP to the baseline; lower-left is better. **(b)** Prompt tokens split into prefix-cache hits (colored) and misses (gray), without residency (left, solid) and with residency (right, hatched).

assigned PIM device computes decode attention and returns the output vector. The GPU then executes the remaining non-attention operations.

PIM as a second-tier prefix cache. Under *Session-KV residency*, PIM retains each session's KVs after decode and serves as a second-tier prefix cache (Figure 1 (c)). The scheduler maps each new turn to the PIM device holding its session KVs, and otherwise to the least-loaded device. The scheduler then composes the reusable prefix from KVs cached on the GPU and PIM. Let G and P denote the GPU and PIM prefix-cache hit lengths. If $P > G$, the scheduler fetches $[G, P)$ from PIM to GPU and prefills only the tokens beyond $\max(G, P)$. If the selected PIM device lacks capacity, the scheduler evicts inactive prefix KVs in LRU order and, if the session KVs still do not fit, remaps the session to another device.

IV. Methodology

Hardware Model and Simulator. We implement *Session-KV residency* in LLMServingSim [6], building on its support for continuous batching with chunked prefill, prefix caching, and real-profile-driven timing and power models. We instantiate the PIM tier with the Duplex Logic-PIM backend [4], modeling each PIM device as an H100-spec HBM stack with $4\times$ the internal bandwidth at equal capacity (other backends are also supported); PIM attention latency and power come from the Duplex cycle-accurate hardware backend. The simulator models GPU-PIM transfers using the configured link bandwidth and latency and accounts for transfer contention. It also includes the constant power draw of each provisioned RNIC port. The link is InfiniBand NDR (50 GB/s aggregate per port, 240 ns latency, 24.9 W per port). We match each removed GPU's 80 GB GPU HBM capacity with five 16 GB PIM devices that share one RNIC port to minimize port energy. To model worst-case contention, the simulator limits each device to one-fifth of the port bandwidth.

Model and workload. We use the same model, workloads, arrival process, request routing, and tool gap methodology as in Section II. Since we observe the same trends for both tool-gap setups, this section contains results from the traces as-is.

Configurations. Starting from a four-GPU (320 GB) baseline, we lower the GPU count (G) and add PIM devices (P). The resulting configurations, 1G+10P, 1G+15P, 1G+20P, and 2G+20P, hold 80 GB of GPU-resident HBM except 2G+20P, which holds 160 GB, and place total HBM capacity below (240 GB), at (320 GB), or above (400 and 480 GB) the baseline capacity (320 GB). We evaluate configurations with and without *Session-KV residency*.

V. Evaluation

We first quantify GPU-to-PIM replacement without *Session-KV residency* (open markers in Figure 3 (a)) and then compare with Session-KV residency (filled markers).

A. How far can PIM bandwidth replace GPU resources?

The amount of prefill work on the reduced GPU pool sets whether a workload lands below or above the baseline EDP.

MathHay isolates the first source, fresh prefill. Prefix caching reuses the cross-session system prompt and document sequence, removing 57% of its 9.8M prompt tokens at 2G+20P. The remaining 4.2M tokens, 56K per session on average, are dominated by session-unique suffixes that added capacity cannot convert into hits. These tokens still prefill on the GPUs, raising latency $1.4\times$ even at the over-provisioned 2G+20P and pushing EDP to $1.27\times$ (Figure 3 (a)).

MCPBench and *Tau2Bench* leave little prefill work on the reduced GPU pool. Their sessions reuse most prior context across turns, and the reusable prefixes stay resident in the remaining GPU HBM because fewer than 50 sessions are live for most of the run (Figure 2 (c)). Removing GPUs shifts decode to PIM with minimal recomputation. Even the under-provisioned 1G+10P reaches $0.43\times$ and $0.42\times$ EDP for *MCPBench* and *Tau2Bench*, respectively. *TerminalBench* also reuses prior context but keeps more sessions live, placing it at the boundary of this regime. At 2G+20P the remaining GPU HBM holds most reusable prefixes, but at 1G+20P, the prefixes no longer fit in GPU HBM and are evicted across tool gaps, increasing prefix-cache misses to $4.4\times$ relative to 2G+20P (Figure 3 (b)).

Search and *SWEBench* exhibit the second source, recomputation. Their decode shares on the four-GPU baseline are close to *Tau2Bench*'s (Figure 2 (b)), yet once GPUs are removed, *Tau2Bench* lands below the baseline EDP and the two land above it. Both workloads accumulate context across many turns, reaching up to 99K tokens at the deepest *SWEBench* turn, but the source of memory pressure differs. *SWEBench* runs sessions far longer than *Tau2Bench* at the same session count (50 in Table I) while *Search* keeps up to 197 sessions live simultaneously (Figure 2 (c)). In both cases, the resulting memory pressure causes eviction of reusable prefix KVs across tool gaps, forcing the remaining GPUs to recompute them. Latency rises by $2.12\times$ and $2.46\times$ at 1G+15P for *Search* and *SWEBench*, respectively, and EDP remains above the baseline ($1.10\times$ and $1.14\times$) even at the over-provisioned 2G+20P. *Session-KV residency* (Section III) retains these evicted KVs on PIM, letting later turns fetch them instead of recomputing.

B. How far can session-KV residency extend the replacement?

Session-KV residency improves replacement only when reduced GPU HBM actually evicts reusable prefixes, and how much it recovers varies with the live-session KV working set.

TerminalBench shows the recovery at the eviction point identified in Section V-A. In all 1G configurations, *Session-KV residency* recovers evicted KVs from PIM instead of recomputing them. At 1G+20P, for example, it reaches $0.49\times$ the four-GPU EDP at 9% higher latency, versus $0.67\times$ EDP at 18% higher latency without residency.

MCPBench and *Tau2Bench* gain little from *Session-KV residency* in their standalone runs ($0.97\times$ energy of their no-residency runs) because eviction barely occurs (Figure 3 (b)). To separate workload effects from memory pressure, we combine *MCPBench*, *Tau2Bench*, and *TerminalBench* into one 179-session stream ($51{+}50{+}78$) that shares the reduced GPU cache pool. At 2G+20P, combining the three workloads raises prefix-cache misses without *Session-KV residency* by 370%, 177%, and 118% for *MCPBench*, *Tau2Bench*, and *TerminalBench*, respectively, relative to their standalone runs. Enabling *Session-KV residency* for the combined stream cuts the prefix-cache misses by $2.7\times$ and EDP by 11.8% relative to the no-residency configuration (Figure 3 (a)). The benefit of *Session-KV residency* therefore depends on the number of sessions sharing the reduced cache pool as well as on session structure (e.g., turn count and accumulated context length).

The benefit of *Session-KV residency* depends on the live-session KV working set, which separates *SWEBench* from *Search*. At 1G+20P, *SWEBench*'s accumulated state fits in the PIM tier, allowing residency to restore nearly every evicted prefix. The prefix-cache hit ratio increases from 50.3% to 98.8%. *Session-KV residency* also reduces energy by 78.0% and latency by 66.5% relative to no residency (Figure 3 (a)). *Search*'s much larger live-session set pressures PIM capacity, so PIM evicts some session KVs before later turns can reuse them, limiting the prefix-cache hit ratio to 84.3%. Even with these PIM evictions, residency cuts EDP by 69.6% relative to no residency at 1G+20P.

Storage Ablation. *Session-KV residency* uses PIM both to preserve reusable KVs and to execute decode attention. To isolate PIM's decode-attention execution, we compare session-aware PIM at 1G+20P with two storage-only baselines that use the same residency policy but run decode on the GPU. *Storage-RDMA* matches PIM's disaggregated capacity and transfer path. *Storage-PCIe5* uses a single capacity-matched same-node store with 64 GB/s bandwidth without RNIC port power. Averaged over *MCP+Tau2+Term*, *Search*, and *SWEBench*, session-aware PIM achieves $0.80\times$ the energy of *Storage-PCIe5* and $0.60\times$ that of *Storage-RDMA*. It also reduces latency by 30% and 32% over Storage-PCIe5 and Storage-RDMA, respectively. Storage-only residency removes the recomputation but runs decode attention on the remaining GPU, paying higher latency and energy. The gap shows that PIM's contribution beyond KV residency comes from executing decode attention on the stored KVs.

VI. Conclusion

In this paper, we evaluated how far PIM can replace GPUs in multi-turn agent serving. GPU-to-PIM replacement is beneficial only while the remaining GPUs can sustain fresh prefill and recomputation caused by KV eviction. To minimize recomputation, *Session-KV residency* fetches GPU-evicted prefixes from PIM rather than recomputing them, cutting energy by up to 78% and latency by 67% relative to the same GPU-PIM baseline configuration. Across workloads, the effective GPU-PIM ratio depends on the fresh-prefill volume and the live-session KV working set. These results motivate full-system evaluation under multi-turn workloads and dynamic provisioning of disaggregated GPU-PIM pools as compute, bandwidth, and KV-capacity demands shift.

References

[1] L. Zheng *et al.*, "SGLang: Efficient Execution of Structured Language Model Programs," *Advances in Neural Information Processing Systems*, vol. 37, 2024.

[2] J. Park *et al.*, "AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference," in *Proc. ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2024.

[3] G. Heo *et al.*, "NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing," in *Proc. ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems*, 2024.

[4] S. Yun *et al.*, "Duplex: A Device for Large Language Models with Mixture of Experts, Grouped Query Attention, and Continuous Batching," in *IEEE/ACM Int. Symp. Microarchitecture*. IEEE, 2024.

[5] X. Li *et al.*, "Benchmark Test-Time Scaling of General LLM Agents," *arXiv preprint arXiv:2602.18998*, 2026.

[6] J. Cho *et al.*, "LLMServingSim 2.0: A Unified Simulator for Heterogeneous and Disaggregated LLM Serving Infrastructure," *IEEE Int. Symp. on Performance Analysis of Systems and Software*, 2026.

[7] M. Luo *et al.*, "Autellix: An Efficient Serving Engine for LLM Agents as General Programs," *arXiv preprint arXiv:2502.13965*, 2025.

[8] D. Kim *et al.*, "Scalable processing-near-memory for 1M-token LLM inference: CXL-enabled KV-cache management beyond GPU limits," in *Int. Conf. Parallel Architectures and Compilation Techniques*. IEEE, 2025.

[9] L. Ke *et al.*, "Near-Memory Processing in Action: Accelerating Personalized Recommendation With AxDIMM," *IEEE Micro*, vol. 42, no. 1, 2021.

[10] F. Devaux, "The true Processing In Memory accelerator," in *IEEE Hot Chips Symp.* IEEE, 2019.