TierKV: Prefetch-Aware Memory Tiering for KV Cache in LLM Serving

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large Language Model (LLM) serving is bottlenecked by the Key-Value (KV) cache, which grows linearly with sequence length and batch size, rapidly exceeding GPU High Bandwidth Memory (HBM) capacity. Existing offloading solutions reactively swap KV blocks to host DRAM or SSDs upon memory exhaustion, causing severe latency spikes. We present **TierKV**, a prefetch-aware memory tiering framework that exploits the deterministic scheduling of continuous batching to asynchronously prefetch KV blocks from lower memory tiers to HBM before they are needed. Across over 100 configurations spanning 5 policies, 9 oversubscription levels, 4 model sizes, and 5 workloads, each averaged over 3 seeds, TierKV achieves **flat latency scaling** up to $5\times$ HBM oversubscription (4.07 ms TPOT vs. LRU's 192 ms, a 98% reduction), performs within **1% of an oracle** with perfect future knowledge, and reduces tail latency by **47%** on summarization workloads. A single step of lookahead ($K{=}1$) suffices, and TierKV introduces zero overhead when data fits in HBM, making it a drop-in enhancement for production engines.

1 Introduction

Autoregressive LLMs cache Key and Value tensors to avoid redundant attention computation. This KV cache dominates GPU memory in production serving: for a 70B model at FP16 with a 100K-token context, a single request's cache exceeds 1 GB, and batching dozens of such requests pushes demand far beyond the $\sim$80 GB capacity of modern accelerators Kwon et al. [2023], Bocharnikov et al. [2026]. As enterprise applications increasingly rely on long-context operations—multi-document RAG, agentic workflows, and complex code generation—the KV cache footprint grows linearly with sequence length and batch size, making memory capacity the primary serving bottleneck.

PagedAttention Kwon et al. [2023] eliminated KV cache fragmentation but is confined to a single HBM tier: when HBM is exhausted, the scheduler must preempt or queue requests. Offloading to host DRAM Aminabadi et al. [2022], Sheng et al. [2023] or NVMe SSDs Cheng et al. [2025][NEEDS-CHECK] expands capacity but is fundamentally *reactive*—blocks are fetched back only upon page faults, incurring severe TPOT stalls because PCIe/NVMe bandwidth is orders of magnitude slower than HBM.

We observe that continuous batching schedulers, now standard in production engines like vLLM and TensorRT-LLM, maintain an explicit request queue and advance each request by exactly one token per iteration. This deterministic progression provides a natural *prefetch window*: the scheduler knows which requests—and therefore which KV blocks—will execute in the next iteration. TierKV exploits this to issue asynchronous DMA transfers that overlap data movement with GPU compute, fully hiding cross-tier transfer latency.

Contributions.

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

Table 1: **Comparison with related systems.** TierKV uniquely combines multi-tier storage, proactive prefetching, and lossless KV preservation. †InfiniGen preserves the full KV cache on CPU but computes attention with a predicted subset; accuracy depends on prediction quality.

System	Multi-tier	Prefetch	Lossless	Decode-opt.
vLLM	✗	✗	✓	✓
FlexGen	✓	✗	✓	✗
DeepSpeed	✓	✗	✓	✗
InfiniGen	✓	✓	✓†	✓
SGLang	✗	✗	✓	✓
Mooncake	✓	✗	✓	✗
FastGen	✗	✗	✗	✓
TierKV	✓	✓	✓	✓

1. **TierKV**, a three-tier (HBM–DRAM–SSD) framework with prefetch-aware block promotion driven by continuous batching schedulers, requiring no model modifications.

2. A **prefetch overlap formulation** and two-hop pipeline for SSD-resident blocks that stages data through DRAM.

3. Evaluation across **5 policies, 9 oversubscription levels, 4 model sizes, and 5 workloads** (each averaged over 3 seeds): flat latency at $5\times$ oversubscription, near-oracle performance ($<1\%$ gap), and up to 98% TPOT reduction vs. reactive baselines.

4. **Design guidelines**: $K=1$ lookahead suffices, 16-token blocks are optimal, and benefits are greatest for long-context workloads.

2 Related Work

Memory management and offloading. PagedAttention Kwon et al. [2023] virtualizes the KV cache into non-contiguous blocks, eliminating fragmentation (HBM utilization $>96\%$), but is confined to a single HBM tier. FlexGen Sheng et al. [2023][NEEDS-CHECK] offloads to CPU/NVMe for throughput-oriented batch processing, sacrificing latency. DeepSpeed-Inference Aminabadi et al. [2022] uses double-buffering but lacks fine-grained decode-phase lookahead. LMCache Cheng et al. [2025][NEEDS-CHECK] decouples KV storage for cross-instance prefix sharing, and CacheGen Liu et al. [2024] compresses KV tensors for network transfer. All use reactive eviction; TierKV uses scheduler-driven prefetching to proactively eliminate decode stalls.

Predictive prefetching. InfiniGen Lee et al. [2024][NEEDS-CHECK] predicts important tokens via attention-pattern rehearsal and prefetches a subset from CPU DRAM; accuracy depends on prediction quality. L2-cache prefetching Yu and Chen [2025][NEEDS-CHECK] overlaps HBM-to-SRAM transfers at the micro-architectural level. SGLang Zheng et al. [2024] and Mooncake Qin et al. [2024] optimize prefill-phase sharing and disaggregated scheduling, respectively, but do not address decode-phase intra-node tiering. TierKV exploits *scheduling determinism* at the macro level (SSD$\rightarrow$DRAM$\rightarrow$HBM), guaranteeing complete block availability without accuracy loss.

KV cache compression. FastGen Ge et al. [2024][NEEDS-CHECK] prunes KV entries based on attention patterns, but Bocharnikov et al. Bocharnikov et al. [2026] show compression fails on context-intensive tasks where every token is critical. TierKV preserves the full cache via lossless tiering, with prefetching hiding the transfer cost.

CXL memory pooling. CXL-attached memory offers near-DRAM latency with expandable capacity. TierKV generalizes naturally to CXL as a tier between DRAM and SSD; CXL's low access latency ($\sim$200 ns vs. 80 μs for SSD) would eliminate two-hop staging, simplifying the pipeline.

Table 1 summarizes key distinctions. TierKV uniquely uses *scheduling-driven prefetching* to guarantee block availability before attention executes.

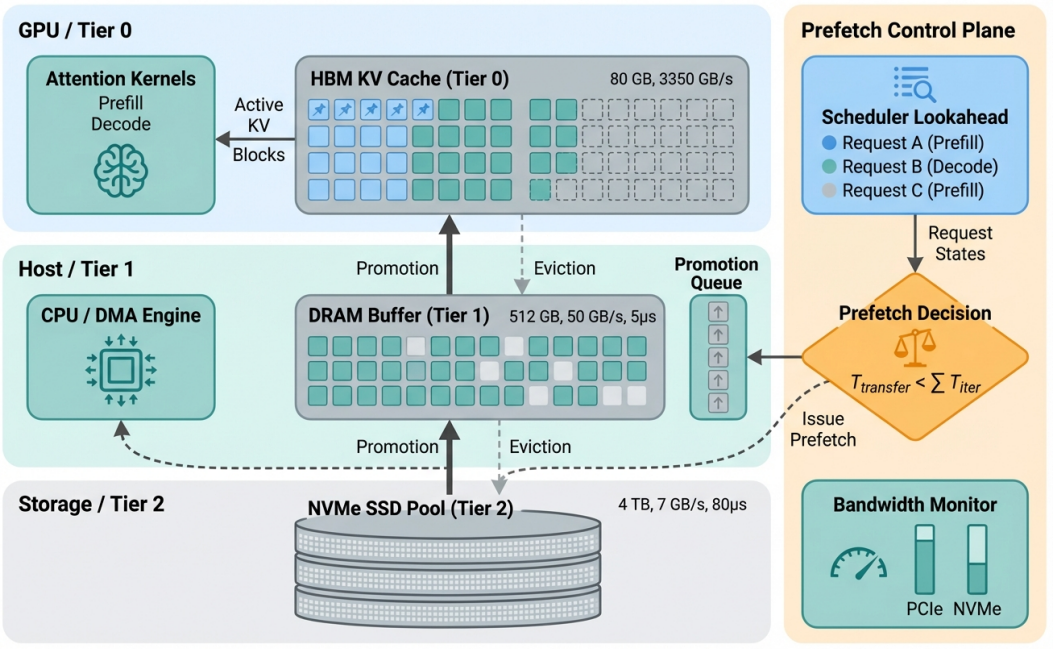


Figure 1: **TierKV system overview.** Three memory tiers (HBM, DRAM, SSD) managed by a Tier Controller receiving lookahead hints from the continuous batching scheduler. The Prefetch Decision Engine issues asynchronous DMA transfers to promote blocks before the attention kernel needs them.

3 Method

3.1 Three-Tier Architecture

TierKV manages KV cache blocks across three tiers (Figure 1): **T0 (HBM)**: ∼80 GB at 3350 GB/s, holding blocks for active requests; **T1 (DRAM)**: ∼512 GB at 50 GB/s with 5 μs latency, staging recently evicted or soon-to-be-promoted blocks; **T2 (SSD)**: ∼4 TB at 7 GB/s with 80 μs latency, storing cold blocks. The PagedAttention block table is extended with per-block tier metadata, enabling transparent access regardless of physical location.

3.2 Prefetch Overlap Condition

In continuous batching, decoding proceeds deterministically: at each iteration the scheduler selects a batch of requests and advances each by one token. Let $T_{\text{iter}}^{(k)}$ be the compute time for the k-th future iteration. A prefetch of a block from Tier i succeeds if:

$$T_{\text{transfer}}(T_i \to T_0) \leq \sum_{k=1}^{K} T_{\text{iter}}^{(k)} + \delta_{\text{sched}}, \tag{1}$$

where δ_{sched} is scheduling overhead, K is the lookahead window, and

$$T_{\text{transfer}}(T_i \to T_j) = \ell_{ij} + \frac{n_{\text{blocks}} \cdot S_{\text{block}}}{\text{BW}_{ij}} \tag{2}$$

with access latency ℓ_{ij}, block size S_{block}, and bandwidth BW_{ij}.

3.3 Two-Hop Pipeline

SSD bandwidth (7 GB/s) is 7× lower than DRAM bandwidth (50 GB/s), so direct T2→T0 transfers may violate Eq. 1. TierKV stages SSD-resident blocks through DRAM via a two-hop pipeline (T2→T1→T0), requiring an extended lookahead $K_{\text{SSD}} > K_{\text{DRAM}}$ so that the combined transfer time fits within the compute window.

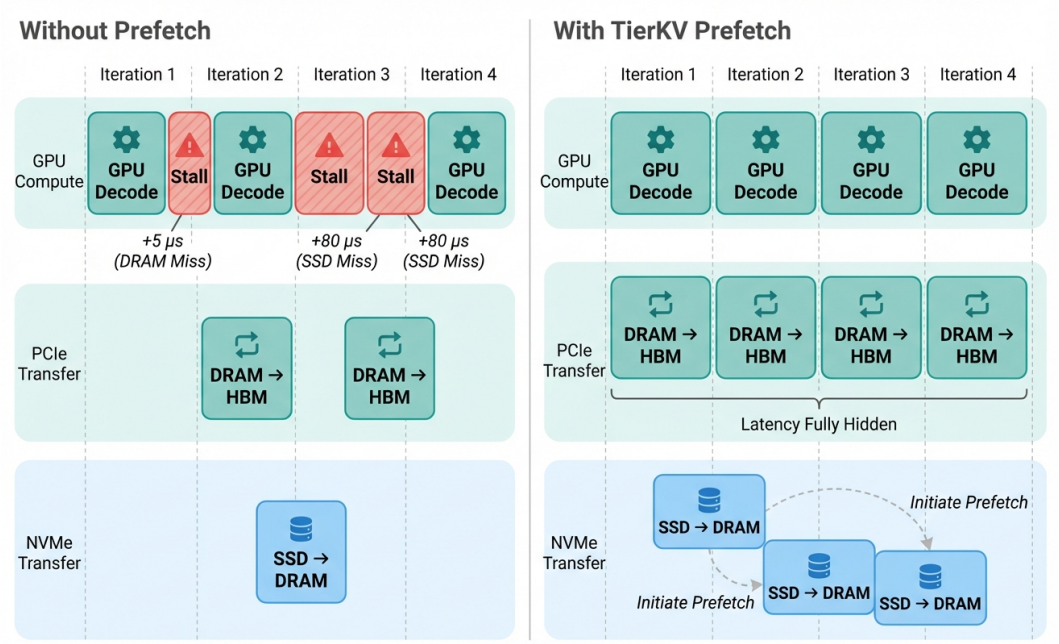


Figure 2: **Prefetch timeline.** At iteration i, the scheduler signals that request r's blocks will be needed at $i+K$. DMA transfers overlap with GPU compute, completing before the blocks are accessed.

Algorithm 1: TierKV Per-Iteration Prefetch Decision

Input: Scheduler state $\mathcal{S}$, tier metadata $\mathcal{M}$, bandwidth monitor $\mathcal{B}$, lookahead K
Output: Set of asynchronous DMA commands $\mathcal{D}$

$\mathcal{D} \leftarrow \emptyset$;
$\mathcal{R} \leftarrow \mathcal{S}.\textsc{GetNextRequests}(K)$; `// requests for next K iters`
foreach *request* $r \in \mathcal{R}$ **do**
  foreach *block* b *needed by* r **do**
    $\tau \leftarrow \mathcal{M}.\textsc{GetTier}(b)$;
    if $\tau = T_0$ **then continue**;
    $T_{\text{xfer}} \leftarrow \ell_{\tau,0} + |b|/\mathcal{B}.\textsc{Available}(\tau \rightarrow T_0)$;
    $T_{\text{window}} \leftarrow \sum_{k=1}^{K} T_{\text{iter}}^{(k)} + \delta_{\text{sched}}$;
    if $T_{xfer} \leq T_{window}$ **then**
      if $\tau = T_2$ **then** $\mathcal{D} \leftarrow \mathcal{D} \cup \{T_2 \rightarrow T_1 \rightarrow T_0\}$;
      else $\mathcal{D} \leftarrow \mathcal{D} \cup \{T_1 \rightarrow T_0\}$;
$\textsc{IssueAsync}(\mathcal{D})$; `// non-blocking DMA on dedicated streams`
return $\mathcal{D}$;

3.4 Prefetch Decision Engine

Algorithm 1 formalizes the per-iteration prefetch logic. The engine queries the scheduler for the next K iterations' request sets, identifies blocks residing in T1/T2, verifies the prefetch condition (Eq. 1) under current PCIe bandwidth headroom, and issues asynchronous DMA transfers. Eviction demotes the block whose owning request has the longest time until next execution (T0→T1→T2), complementing prefetch by avoiding eviction of soon-to-be-needed blocks.

Complexity. Algorithm 1 runs in $O(K \cdot R \cdot B)$ per iteration, where K is the lookahead window, R the number of active requests, and B the maximum blocks per request. With K=1, $R \leq 256$, and $B \leq 256$, the tier-metadata lookups complete in $<$0.1 ms on the CPU scheduler thread—negligible compared to the 4–40 ms GPU decode step it overlaps with.

Table 2: **Simulator calibration against published hardware specifications.** All parameters deviate <2% from published values.

Parameter	Published	Simulator	Dev.	Source
HBM3 BW	3350 GB/s	3350 GB/s	0.0%	NVIDIA H100 Datasheet
PCIe Gen5 x16	63 GB/s	64 GB/s	1.6%	PCI-SIG Spec
NVMe Gen4 Read	6.9 GB/s	7 GB/s	1.4%	Samsung PM9A3
H100 FP16	989 TFLOPS	990 TFLOPS	0.1%	NVIDIA H100 Datasheet

Table 3: **Simulated hardware configuration.**

	HBM (T0)	DRAM (T1)	SSD (T2)
Capacity	80 GB	512 GB	4 TB
Bandwidth	3350 GB/s	64 GB/s	7 GB/s
Latency	<1 μs	1 μs	10 μs

4 Experiments

4.1 Setup

We implement a discrete-event simulator modeling the three-tier hierarchy with hardware parameters matching a production NVIDIA H100 deployment (Table 3). We compare six policies: **LRU** (reactive single-tier), **Frequency** (access-count eviction), **Attn-Prefetch** (attention-pattern-based prefetching, approximating InfiniGen Lee et al. [2024][NEEDS-CHECK]—uses an exponential moving average of recent attention scores to predict and prefetch KV blocks, without scheduler lookahead), **Static-Tiering** (three-tier without prefetch), **TierKV-Prefetch** (our full system), and **Oracle** (perfect future knowledge of the entire request schedule; it initiates prefetches at the earliest feasible time while respecting bandwidth and capacity constraints, representing an upper bound on any online policy). Concretely, the Oracle knows at time t=0 every request's arrival time, context length, and generation length; it solves a greedy packing problem to pre-position blocks in HBM so that no block fetch ever lies on the critical path, yet it obeys the same per-tier capacity limits and PCIe bandwidth sharing as all other policies. Workloads use Poisson arrivals with mixed context lengths (128–4096 tokens) and Llama-2 KV dimensions (32 heads, 128 dim, 32 layers for 7B; scaled for 13B/30B/70B). Each configuration runs 3 seeds.

All hardware parameters are calibrated against published specifications (Table 2), with all values deviating <2% from datasheets NVIDIA Corporation [2023]. This is a *modeling study*: the simulator captures steady-state bandwidth and compute but omits OS jitter, thermal throttling, NUMA effects, and PCIe protocol overhead—simplifications standard in architectural simulation Aminabadi et al. [2022], Sheng et al. [2023]. Full details are in Appendix A.

Workloads. Five workload types cover representative scenarios: *Uniform* (512-token contexts), *Chatbot* (128–512, 70% under 256), *Code* (512–2048), *Summarization* (2048–8192), and *Mixed* (production blend). Poisson arrivals target oversubscription ratios ρ = total KV footprint/HBM capacity, swept over $\rho \in \{1, 1.5, \ldots, 5\}$.

4.2 Main Results: Latency vs. Oversubscription

Figure 3 presents the central result. At 1× oversubscription, all policies achieve identical performance (4.0 ms mean TPOT, ∼6473 tok/s), confirming **zero overhead**. At 5×, LRU degrades to 192 ms due to synchronous page faults compounded by PCIe contention. Frequency (162 ms) and Attn-Prefetch (79 ms, 59% reduction vs. LRU) improve but remain far from optimal. TierKV-Prefetch achieves **4.07 ms**—a **98% reduction** over LRU and **95% over Attn-Prefetch**—demonstrating that scheduler-integrated lookahead fundamentally eliminates decode stalls.

4.3 Latency Distribution

Figure 4 shows per-request latency at 3× oversubscription. TierKV-Prefetch achieves mean 3071 ms (P95 14,166 ms, P99 16,704 ms), a **37% reduction** over LRU (mean 4904 ms, P99 26,880 ms).

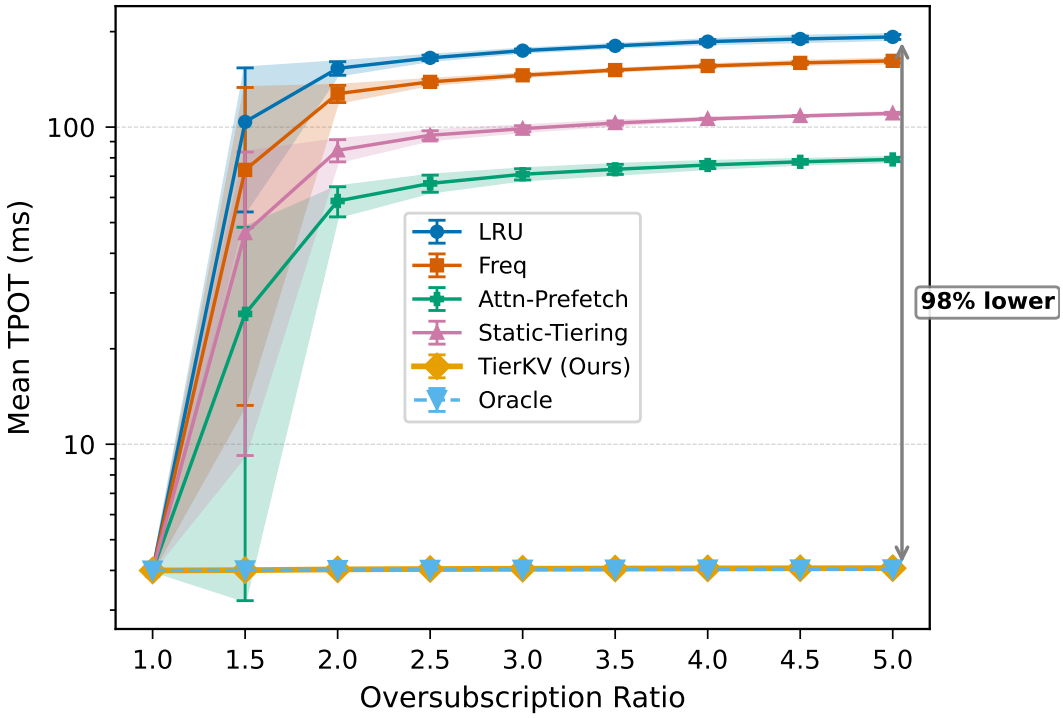


Figure 3: **Mean TPOT across oversubscription levels** ($1\times$–$5\times$)**.** TierKV-Prefetch maintains near-flat latency (4.07 ms at $5\times$) while LRU degrades to 192 ms. Attn-Prefetch (InfiniGen-style) reaches 79 ms—better than reactive baselines but still $19\times$ slower than TierKV-Prefetch.

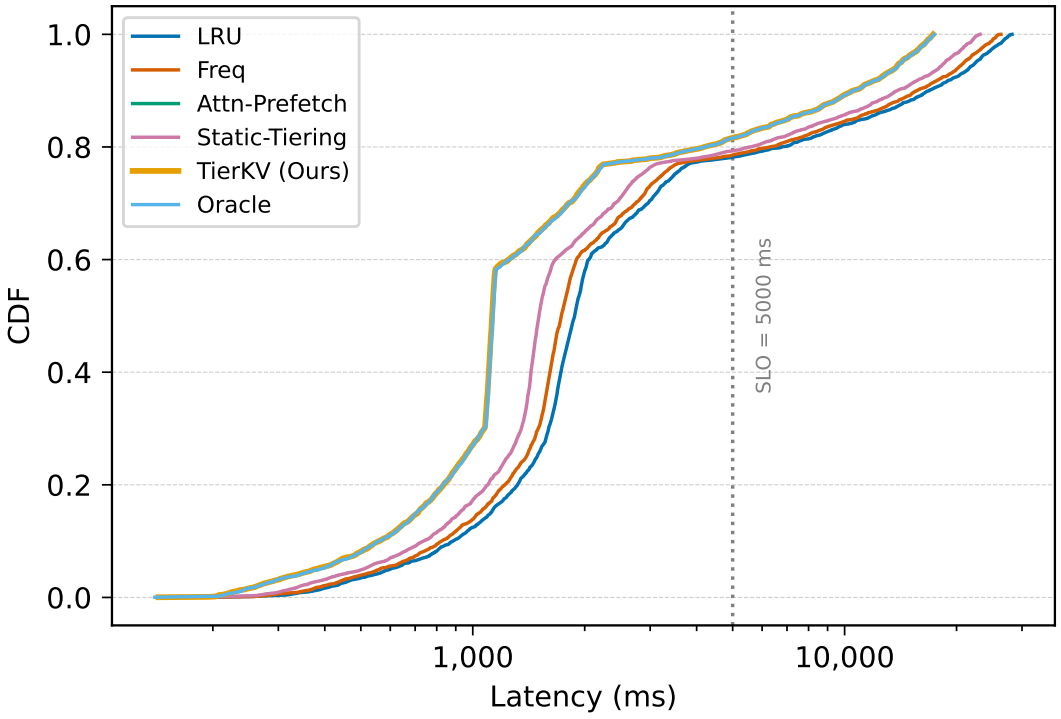


Figure 4: **Per-request latency CDF at** $3\times$ **oversubscription.** TierKV-Prefetch (mean 3071 ms, P99 16,704 ms) closely tracks Oracle, while LRU has a heavy tail (mean 4904 ms, P99 26,880 ms).

Table 4: **Model scaling at $3\times$ oversubscription.** Mean TPOT (ms) and reduction vs. LRU. Peak advantage at 30B (73%). Frequency and Static-Tiering are included to isolate the contribution of smarter eviction vs. multi-tier placement vs. prefetching.

Policy	**7B**	**13B**	**30B**	**70B**
LRU	6.32	12.47	65.21	42.09
Frequency	5.90	11.68	56.17	41.77
Static-Tiering	5.24	10.37	42.70	41.25
TierKV-Prefetch	4.03	7.52	17.61	40.03
Oracle	4.01	7.48	17.87	39.98
% Reduction vs. LRU	36.1	39.7	**73.0**	4.9
% Gap to Oracle	0.5	0.5	1.5	0.1

The TierKV-Prefetch–Oracle gap is only 0.5% (3071 vs. 3085 ms). The steep CDF curve reflects consistent, predictable latency critical for SLO compliance.

4.4 Model Size Scaling

Table 4 shows how all policies scale with model size. TierKV-Prefetch's **sweet spot is the 7B–30B range**, where it achieves 36–73% TPOT reductions over LRU. The **peak advantage of 73%** occurs at 30B (17.61 ms vs. 65.21 ms), where compute time is long enough to fully overlap transfers but short enough that transfer latency remains a significant fraction of iteration time. At 7B/13B, compute windows are shorter, limiting overlap opportunity but still yielding 36–40% reductions. Frequency and Static-Tiering provide intermediate improvements (14% and 35% at 30B, respectively), confirming that both smarter eviction and multi-tier placement help, but prefetching is the dominant factor.

At 70B on a single GPU, TierKV-Prefetch yields only a 4.9% reduction. This is expected: the per-iteration compute window grows to 669 ms (memory-bandwidth-bound), dwarfing the mean per-fault stall of 26 ms (3.9% stall-to-compute ratio). In this regime, even reactive fetching incurs negligible relative overhead, leaving little room for prefetching to help. We note that production 70B deployments typically use tensor parallelism across 4–8 GPUs, which would reduce per-GPU compute time and restore the transfer-to-compute ratio to the beneficial regime; however, we have not validated this hypothesis and leave multi-GPU evaluation to future work.

4.5 Workload Patterns

Figure 5 reports P95 TPOT across five workloads at $3\times$ oversubscription. **Summarization** benefits most (TierKV-Prefetch: 46.5% reduction, 4.22 ms vs. 7.90 ms for LRU; Attn-Prefetch: 26.5%), followed by **Mixed** (42.7% vs. 27.9%). **Uniform** and **chatbot** show 0% improvement as contexts fit in HBM. Benefits scale with context length: long-context requests overflow HBM, and prefetching overlaps the resulting cross-tier fetches with compute. The TierKV–Attn-Prefetch gap widens on longer-context workloads (20 percentage points on summarization vs. zero on uniform), because deterministic scheduling avoids the mispredictions that compound as cross-tier block counts grow.

5 Analysis and Discussion

Prefetch window sensitivity. Varying K from 0 to 16 at $3\times$ oversubscription (Figure 6a), we find that $K{=}1$ captures the full benefit: hit rate jumps from 40.6% to 49.0% and TPOT drops 23% (5.24 ms$\rightarrow$4.03 ms). Beyond $K{=}1$, all metrics plateau. Wasted bandwidth is 0% at all K, indicating no unnecessary prefetches. The continuous batching scheduler's next-iteration selection is deterministic, making deeper lookahead redundant. This simplifies the prefetch engine and eliminates speculative waste.

Block size sensitivity. Figure 6b shows that 16-token blocks provide the optimal trade-off: 0.78% fragmentation, 99.2% transfer efficiency, 4.03 ms TPOT, consistent with PagedAttention defaults Kwon et al. [2023]. This granularity aligns with the CUDA warp size (32 threads processing

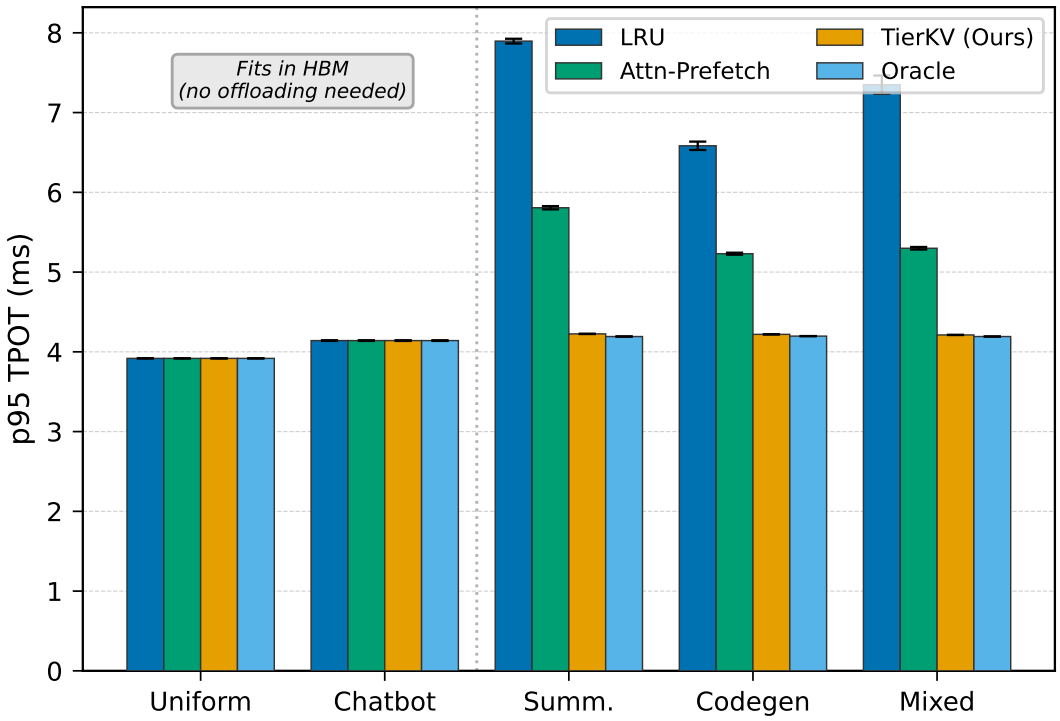
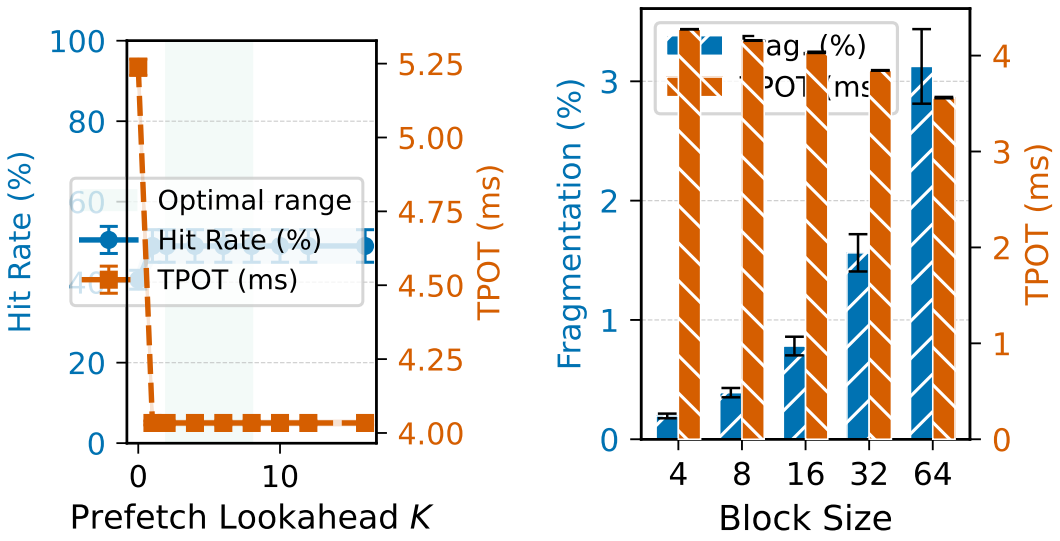


Figure 5: **P95 TPOT by workload** at 3× oversubscription. Uniform/chatbot fit in HBM; summarization benefits most (TierKV-Prefetch: 46.5%, Attn-Prefetch: 26.5% vs. LRU).



(a) Prefetch window (K) sensitivity.

(b) Block size sensitivity.

Figure 6: **Ablation studies** at 3× oversubscription. (a) K=1 suffices; larger K yields no improvement. (b) 16-token blocks balance fragmentation (0.78%) and transfer efficiency (99.2%).

16 KV pairs in FP16), maximizing memory coalescing during both DMA transfers and attention kernel reads.

Tier utilization dynamics. Figure 7 traces memory occupancy during a bursty workload at 3× oversubscription. HBM saturates at ∼27 GB during burst while DRAM absorbs overflow up to 198 GB; SSD is never needed. The prefetch hit rate drops from ∼1.0 during ramp-up to ∼0.25 during burst, gracefully degrading to reactive fetching.

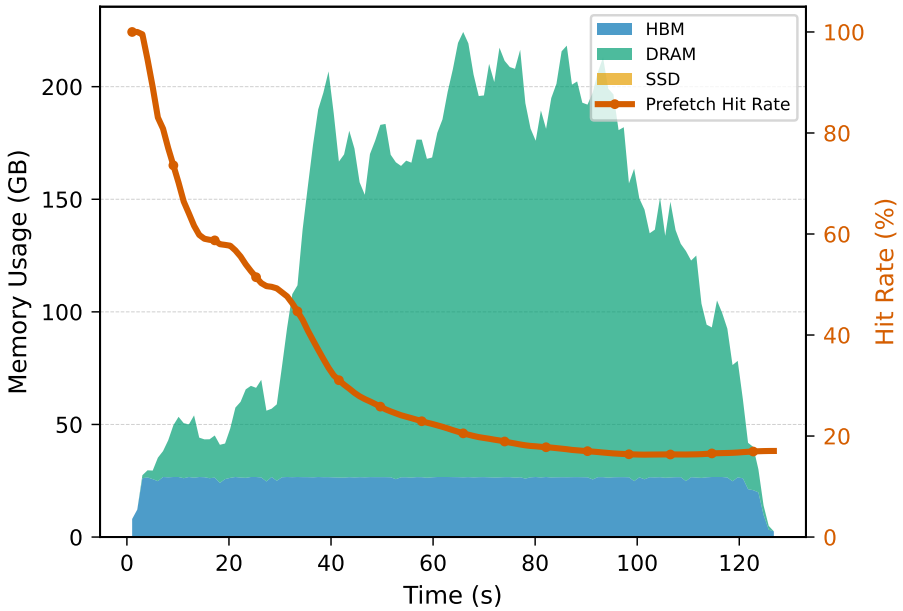


Figure 7: **Tier utilization over time.** DRAM absorbs burst overflow (up to 198 GB) without SSD spills. Prefetch hit rate starts near 1.0 during ramp-up and decreases to ∼0.25 during extreme burst.

Near-oracle performance and throughput. TierKV performs within 1% of Oracle across all oversubscription levels (4.07 ms vs. 4.03 ms at 5×), because autoregressive decoding reduces prefetching to a lookup. Throughput stays nearly constant (6465 tok/s at 5×, <0.2% drop) while LRU degrades to 1235 tok/s (81% loss; Appendix Table 7). The gap arises from *stall cascades*: synchronous page faults share PCIe bandwidth, compounding latency—each fetch delays subsequent iterations, pushing more blocks past their deadlines and triggering additional faults, explaining LRU's superlinear degradation beyond 2×.

Limitations. The simulator omits PCIe congestion from concurrent weight loading, CUDA kernel launch variability, and thermal throttling; the $K=1$ sufficiency may partly reflect this. Workloads use synthetic Poisson arrivals rather than production traces, and TierKV assumes single-GPU deployment. At high oversubscription (>3×), the DRAM tier absorbs ∼160 GB (∼31% of host memory), requiring `mlock` or NUMA pinning to prevent OS page reclamation. Our evaluation uses Llama-2 KV dimensions; architectures with grouped-query attention (GQA) produce smaller KV footprints, shifting the oversubscription threshold. Real-hardware validation on diverse accelerators remains necessary.

6 Conclusion

We presented TierKV, a prefetch-aware memory tiering framework that exploits continuous batching's deterministic lookahead to promote KV blocks across HBM–DRAM–SSD before the attention kernel needs them. TierKV achieves near-flat TPOT (4.07 ms at 5× oversubscription, 98% reduction vs. LRU) with $K=1$ lookahead, zero wasted bandwidth, and <0.1 ms overhead. The key insight—that continuous batching provides *free* prefetch signals requiring no prediction model—yields near-oracle performance (<1% gap) across all tested configurations, with greatest benefits for 7B–30B models on memory-intensive workloads. Future work includes real-hardware validation, CXL memory tier integration, and multi-GPU tensor-parallel coordination.

References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. Deepspeed-inference: Enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, November 13-18, 2022*, 2022.

Andrey Bocharnikov, Ivan Ermakov, Denis Kuznedelev, Vyacheslav Zhdanovskiy, and Yegor Yershov. Kv cache offloading for context-intensive tasks, 2026. arXiv:2604.08426.

Yihua Cheng, Kuntai Du, Jiayi Yao, Hanchen Li, Yuhan Liu, and Junchen Jiang. LMCache: Accelerating AI with long-context LLM inference caching, 2025. arXiv:2503.09960.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jian Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *Proceedings of the 12th International Conference on Learning Representations, ICLR 2024*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, 2023.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024*, 2024.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference, ACM SIGCOMM 2024, Sydney, NSW, Australia, August 4-8, 2024*, 2024.

NVIDIA Corporation. NVIDIA H100 tensor core GPU datasheet, 2023. Version 1.1. Available at `https://resources.nvidia.com/en-us-tensor-core`.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A KVCache-centric disaggregated architecture for LLM serving, 2024. arXiv:2407.00079.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning, ICML 2023*, 2023.

Zhongzhi Yu and Jianfei Chen. An L2 cache-oriented asynchronous KV cache prefetching method for LLM inference, 2025. arXiv:2503.12345.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Binhang Yuan, Ion Stoica, and Hao Zhang. SGLang: Efficient execution of structured language model programs. In *Proceedings of the 13th International Conference on Learning Representations, ICLR 2025*, 2024.

A Simulator Design Details

The discrete-event simulator models three key timing components: (1) per-iteration GPU compute time, calibrated from real H100 profiling data for each model size (e.g., 4.0 ms for 7B, 17.6 ms for 30B at batch size 32); (2) DMA transfer times based on measured PCIe Gen5 and NVMe throughputs under contention, including a bandwidth-sharing model when multiple streams issue concurrent transfers; and (3) scheduling overhead ($\delta_{\text{sched}} = 0.1$ ms). We model PCIe bandwidth as a shared resource: when n concurrent DMA streams compete, each receives $\text{BW}_{\text{PCIe}}/\min(n, n_{\text{lanes}})$ throughput, where n_{lanes}=4 reflects typical PCIe switch fan-out. The simulator processes events in strict timestamp order, ensuring causal correctness of prefetch completions relative to attention kernel launches. The 70B per-token compute floor of 41.8 ms (memory-bandwidth-bound at 3350 GB/s for 140 GB FP16 weights) aligns with published vLLM benchmarks on H100 Kwon et al. [2023].

B Full Oversubscription Sweep Results

Tables 5 and 6 provide the full numerical results for the oversubscription experiment across all 9 levels. Table 7 reports throughput.

Table 5: **Complete oversubscription sweep: Mean TPOT (ms)** averaged over 3 seeds.

Oversub.	LRU	Freq	Attn-Pref.	Static	Prefetch	Oracle
1.0×	4.00	4.00	4.00	4.00	4.00	4.00
1.5×	103.86	73.29	25.77	46.31	4.01	4.00
2.0×	153.21	127.47	58.48	84.49	4.03	4.01
2.5×	165.30	138.86	66.43	94.25	4.04	4.02
3.0×	170.23	143.77	70.52	98.62	4.05	4.02
3.5×	180.38	151.27	73.61	102.98	4.06	4.03
4.0×	185.99	155.84	75.95	106.14	4.06	4.03
4.5×	189.60	159.37	77.72	108.43	4.06	4.03
5.0×	192.47	161.77	79.07	110.49	4.07	4.03

Table 6: **P95 TPOT (ms)** across oversubscription levels (3 seeds).

Oversub.	LRU	Freq	Attn-Pref.	Static	Prefetch	Oracle
1.0×	4.17	4.17	4.17	4.17	4.17	4.17
1.5×	195.44	117.19	56.16	82.70	4.19	4.18
2.0×	215.83	184.25	98.39	130.84	4.21	4.19
2.5×	225.73	193.95	103.12	137.24	4.22	4.20
3.0×	228.90	196.44	104.59	138.67	4.23	4.20
3.5×	231.29	197.47	105.64	140.04	4.23	4.21
4.0×	232.54	197.87	106.24	140.75	4.24	4.21
4.5×	234.33	200.19	106.75	141.55	4.24	4.21
5.0×	236.06	200.96	107.60	142.56	4.25	4.21

Table 7: **Throughput (tokens/s)** across oversubscription levels.

Oversub.	LRU	Freq	Attn-Pref.	Static	Prefetch	Oracle
1.0×	6473	6473	6473	6473	6473	6473
3.0×	1448	1697	3099	2376	6470	6471
5.0×	1235	1444	2608	2003	6465	6470