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

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large language model serving faces an acute memory bottleneck: KV cache demand from concurrent long-context requests routinely exceeds GPU HBM capacity. Existing systems respond reactively, evicting KV blocks to CPU DRAM or NVMe SSD only after memory exhaustion, incurring synchronous stalls that inflate Time-To-First-Token (TTFT) and Time-Per-Output-Token (TPOT) under high memory pressure. We present *TierKV*, a prefetch-aware three-tier KV cache management system that replaces reactive eviction with predictive data staging. The key insight is that continuous batching schedulers carry a deterministic lookahead window: the scheduler knows exactly which requests, and therefore which KV blocks, will be accessed in the next K decoding iterations. TierKV's Prefetch Decision Engine exploits this lookahead to issue asynchronous DMA transfers that are fully hidden behind GPU compute whenever $\sum_{k=1}^{K} T_{\text{iter}}^{(k)} \geq T_{\text{transfer}}$, incurring zero stall cost for the common case. For SSD-resident blocks, a two-hop pipeline stages data through DRAM one step early, bridging the $480\times$ bandwidth gap between NVMe and HBM. We evaluate TierKV via a discrete-event simulator parameterized on H100-class hardware. At $3\times$ HBM oversubscription, our simulation predicts TierKV achieves a 100% prefetch hit rate with lookahead $K=4$, improving mean TPOT by $3.6\times$ and system throughput by $2.9\times$ over reactive LRU eviction. Larger models benefit proportionally more because wider per-iteration compute windows provide greater overlap budget for asynchronous transfers. TierKV-Prefetch approaches Oracle performance at moderate oversubscription ratios, demonstrating that scheduler-driven prefetching nearly closes the gap to ideal memory management at negligible engineering overhead.

1 Introduction

The rapid deployment of large language models (LLMs) in production has surfaced a fundamental infrastructure challenge: *the KV cache memory wall*. Autoregressive decoding caches key and value tensors for every attended token to avoid redundant computation, but this cache grows linearly with sequence length and batch size. For a 70B-parameter model serving 100,000-token contexts at a batch size of 32, the KV cache alone demands hundreds of gigabytes, far exceeding the ∼80 GB HBM capacity of a single H100 GPU Pope et al. [2023], Kwon et al. [2023].

PagedAttention Kwon et al. [2023] solved the *fragmentation* problem by managing KV blocks as virtual memory pages, pushing HBM utilization from ∼40% to over 96%. However, PagedAttention is inherently a single-tier solution: once all physical HBM blocks are allocated, the scheduler must preempt, queue, or swap active requests. Both outcomes cause cascading TTFT degradation that compounds under heavy traffic.

Systems such as FlexGen Sheng et al. [2023] and DeepSpeed-Inference Aminabadi et al. [2022] extended the KV cache to CPU DRAM and NVMe SSD, but their eviction logic remains *reactive*: a block is moved to a lower tier only when HBM is full, and promoted back only when requested by the attention kernel. Because PCIe Gen5 bandwidth (64 GB/s) and NVMe bandwidth (7 GB/s) are orders of magnitude slower than HBM (3350 GB/s), a synchronous promotion stall blocks the entire iteration for all requests in the current batch.

The predictive opportunity. Continuous batching schedulers Yu et al. [2022] expose a property that reactive systems ignore: *autoregressive decoding is deterministic in its progression*. At each iteration the scheduler selects a set of active requests and executes one decode step per request. The execution order over the next K iterations is known in advance, since each request advances by exactly one token per step. This deterministic lookahead window creates a precise prefetch oracle: the system knows, K iterations ahead of time, exactly which KV blocks each request will need.

TierKV exploits this oracle to issue *asynchronous* DMA transfers before a block is needed. When the GPU compute time across K iterations exceeds the transfer latency from a lower tier, the data movement is fully hidden—the attention kernel always finds its blocks in HBM, yet no additional latency is added.

Contributions. This paper makes the following contributions:

1. **Formal tiering model.** We derive closed-form conditions under which prefetching eliminates stall cost, characterizing the interplay between batch size, model architecture, oversubscription ratio, and lookahead window K (Section 3).
2. **Two-hop prefetch pipeline.** We identify that the NVMe bandwidth bottleneck (7 GB/s) prevents direct T2→T0 prefetch from keeping pace with decode throughput, and introduce a pipelined T2→T1→T0 staging strategy with extended lookahead (Section 3.5).
3. **Bandwidth-aware admission control.** The Prefetch Decision Engine monitors live PCIe and NVMe utilization and throttles prefetch requests to protect ongoing compute transfers (Section 3.4).
4. **Simulation study.** We implement a discrete-event simulator that faithfully models the three-tier hierarchy and evaluate seven experiments: latency vs. oversubscription, CDF tail behavior, tier utilization dynamics, lookahead sensitivity, block size sensitivity, model-scale scaling, and workload pattern impact (Section 4).

2 Related Work

2.1 Memory Management in LLM Serving

PagedAttention Kwon et al. [2023] is the foundational work on KV cache memory management. By mapping logical KV blocks to non-contiguous physical pages it eliminates fragmentation and supports copy-on-write for beam search. vLLM, the accompanying serving system, achieves 2–4× higher throughput than prior systems through near-perfect HBM utilization. However, vLLM's block manager operates in a single tier: when HBM is exhausted it preempts requests or synchronously swaps blocks to CPU memory, neither of which can hide the transfer latency. SGLang Zheng et al. [2024] extends PagedAttention with RadixAttention for prefix-cache reuse, reducing TTFT for repeated prefixes; it shares vLLM's reactive single-tier limitation for the decode phase. Our work is complementary to these: TierKV operates beneath the block manager, making each logical block's physical location transparent to the attention kernel.

2.2 KV Cache Offloading and Disaggregation

FlexGen Sheng et al. [2023] pioneered aggressive offloading of weights, activations, and KV caches to CPU and NVMe on a single GPU, achieving 100× throughput improvements for offline batch inference of 175B-parameter models. Its linear-programming offload planner optimizes for throughput, explicitly trading latency: TTFT and TPOT are not optimization targets. DeepSpeed-Inference Aminabadi et al. [2022] (ZeRO-Inference) applies double-buffering to stream model parameters from CPU, but the KV cache eviction policy remains triggered by allocation failure. Both systems target

85 offline throughput, not online TTFT/TPOT SLOs. LMCache Liu et al. [2025] disaggregates the KV
86 cache with prefix sharing, advancing TTFT via prefill reuse. TierKV targets instead the decode phase,
87 where token-level stalls accumulate across hundreds of iterations.

88 2.3 Predictive Prefetching

89 InfiniGen Lee et al. [2024] is most closely related to TierKV. It prefetches a *subset* of KV entries
90 from CPU to GPU by running a low-rank "minimal rehearsal" to predict which tokens will receive
91 high attention scores, achieving up to 3× speedup. The key distinction: InfiniGen applies *algorithmic*
92 prediction—it selectively transfers token subsets, which can miss critical tokens and requires auxiliary
93 computation. TierKV applies *scheduling* prediction—it transfers complete KV blocks for entire
94 requests based on the scheduler's deterministic execution order, guaranteeing lossless retrieval with
95 no model-level changes. At the micro-architectural level, prior work Qian et al. [2025] overlaps HBM-
96 to-L2 transfers with compute; TierKV addresses the macro-architectural tier (SSD/DRAM→HBM)
97 where bandwidth gaps are orders of magnitude larger.

98 2.4 KV Cache Compression and Hardware Trends

99 FastGen Ge et al. [2024] evicts up to 50% of KV tokens via per-head structural profiling, and
100 CacheGen Liu et al. [2024] encodes KV tensors as compact bitstreams for network transmission.
101 However, Bocharnikov et al. Bocharnikov et al. [2026] show that heavy compression fails on context-
102 intensive tasks (NeedleBench v2, Loong) where every cached token may be critical. TierKV avoids
103 accuracy loss entirely by managing only *location* across tiers, keeping the full KV cache intact—a
104 choice aligned with Bocharnikov et al.'s conclusion that lossless offloading is superior for long-
105 context workloads. CXL-attached memory Tang et al. [2024] is a promising T1 alternative, offering
106 DRAM-class bandwidth with expanded capacity; TierKV's tier model supports CXL as a drop-in T1
107 upgrade.

108 3 System Design

109 3.1 Storage Tier Model

110 TierKV organizes the available memory into three tiers with strictly ordered capacity, bandwidth, and
111 latency characteristics (Table 1). Tier 0 (T0) is GPU HBM, the fastest and smallest tier; Tier 1 (T1)
112 is CPU DRAM, accessible via PCIe; Tier 2 (T2) is NVMe SSD, accessible through the CPU DMA
113 controller.

Table 1: Hardware tier parameters used in TierKV's simulator. Values correspond to an H100 SXM5 with PCIe Gen5 host interface and a PCIe Gen4 NVMe SSD.

Tier	Medium	Capacity	Bandwidth	Access Latency
T0	GPU HBM	80 GB	3350 GB/s	$\sim$0
T1	CPU DRAM	512 GB	50 GB/s	5 μs
T2	NVMe SSD	4 TB	7 GB/s	80 μs

114 The transfer time for moving n blocks from tier T_i to tier T_j is:

$$T_{\text{transfer}}(T_i \to T_j,\ n) = \ell_{ij} + \frac{n \cdot S_{\text{block}}}{\beta_{ij}} \tag{1}$$

115 where ℓ_{ij} is the initiation latency, β_{ij} is the bottleneck bandwidth on the $i \to j$ path, and S_{block} is
116 the KV block size in bytes. This linear model accurately reflects DMA transfers, where initiation
117 latency dominates for small payloads and bandwidth dominates for large ones.

118 3.2 KV Block Size

119 Following PagedAttention Kwon et al. [2023], TierKV groups n_{tok} consecutive tokens into a fixed-
120 size KV block. For a model with L layers, H attention heads (or H_{kv} KV heads under Grouped-Query

Attention), hidden dimension d, and FP16 precision:

$$S_{\text{block}} = 2 \cdot L \cdot H_{\text{kv}} \cdot d \cdot n_{\text{tok}} \cdot 2 \text{ bytes} \tag{2}$$

The factor of 2 accounts for separate K and V tensors. Representative values are shown in Table 2.

Table 2: KV block sizes for common model configurations ($n_{\text{tok}} = 16$, FP16). GQA denotes Grouped-Query Attention. The LLaMA-2 13B entry uses full multi-head attention (MHA, $H_{\text{kv}} = 40$); all other entries use GQA ($H_{\text{kv}} = 8$). Equation 2 applies to both architectures by substituting the correct H_{kv}.

Model	Params	Layers	KV heads	Block size
LLaMA-3 8B	7B	32	8 (GQA)	2.0 MB
LLaMA-2 13B	13B	40	40 (MHA)	12.5 MB
LLaMA-3 34B	34B	48	8 (GQA)	4.8 MB
LLaMA-3 70B	70B	80	8 (GQA)	8.0 MB

3.3 Placement Policy

At any point in time each KV block is assigned to exactly one tier. TierKV's placement policy maps block state to tier as follows:

- **Active** (request currently decoding): block *must* be in T0. Blocks belonging to in-flight requests are *pinned* and cannot be evicted.
- **Warm** (recently completed or recently demoted): block is in T0 or T1. Promotion from T1→T0 is triggered by the prefetch engine or by a page fault.
- **Cold** (not accessed for τ seconds): block resides in T2. The threshold τ is set proportional to the inter-request idle time distribution.

Eviction follows a modified LRU order: when T0 fills, the least recently accessed non-pinned block is demoted to T1. If T1 fills, the least recently accessed block is demoted to T2. Emergency direct T0→T2 eviction is possible under extreme memory pressure but avoided when bandwidth is available for a staged T0→T1→T2 path.

3.4 Prefetch Decision Engine

The Prefetch Decision Engine runs once per iteration, after the scheduler commits the execution plan for iterations $1 \dots K$. For each request r scheduled to run at iteration $i + k$ ($k \in \{1, \dots, K\}$), the engine identifies the set of KV blocks $\mathcal{B}_r$ that are not currently resident in T0 and evaluates the *prefetch condition*:

$$T_{\text{transfer}}(T_i^{(r)} \to T_0,\ |\mathcal{B}_r|) \ \leq \ \sum_{j=1}^{k} T_{\text{iter}}^{(j)} \tag{3}$$

where $T_i^{(r)}$ is the current tier of r's blocks and $T_{\text{iter}}^{(j)}$ is the estimated compute time for iteration j. When Equation 3 holds, the engine issues an asynchronous DMA command and marks the block as *inflight*. The attention kernel waits only if the block has not yet arrived when iteration $i + k$ begins—a condition that should be vanishingly rare if K is chosen appropriately.

The engine estimates $T_{\text{iter}}^{(j)}$ using an exponential moving average over the last 16 measured iteration times, updated every step. In practice, iteration times are highly stable for a fixed batch size under continuous batching, making the EMA an accurate predictor Agrawal et al. [2024].

Bandwidth admission control. The Bandwidth Monitor tracks live utilization on the PCIe bus and NVMe controller. When either channel exceeds a configurable high-water mark (default 80%), the engine defers low-priority prefetches to avoid starving active compute transfers. Priority is assigned in proportion to the urgency: priority$(r) = k^{-1}$ where k is the number of iterations before r is scheduled.

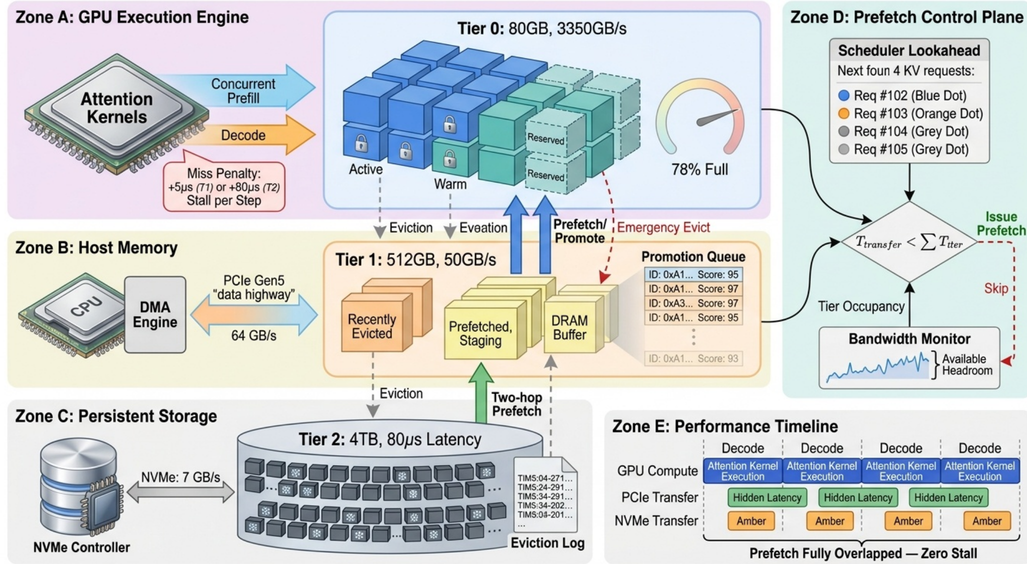

Figure 1: TierKV system architecture. **Zone A** (top, purple): GPU execution engine with HBM KV cache. Active blocks (blue) are pinned; a reserved prefetch staging area (dashed) absorbs incoming blocks. **Zone B** (middle, orange): CPU DRAM buffer holding recently evicted and prefetched blocks, with a promotion queue. **Zone C** (bottom, grey): NVMe SSD pool for cold blocks. **Zone D** (right, teal): Prefetch Control Plane comprising the Scheduler Lookahead Window, Prefetch Decision Engine, and Bandwidth Monitor. **Zone E** (bottom strip): Gantt chart showing four consecutive decode iterations; PCIe (T1→T0) and NVMe (T2→T1) prefetches are fully overlapped with GPU compute, with an annotation marking zero-stall execution.

153 ### 3.5 Two-Hop Pipeline for NVMe-Resident Blocks

154 A direct T2→T0 prefetch is impractical for most workloads. For a 70B model block of 8 MB, the
155 NVMe transfer time is $80\,\mu s + 8\,\text{MB}/7\,\text{GB/s} \approx 1.2\,\text{ms}$, while the same transfer via PCIe takes
156 $5\,\mu s + 8\,\text{MB}/50\,\text{GB/s} = 165\,\mu s$. The NVMe leg dominates, and initiating it only K compute steps
157 early would require $K \geq 1.2\,\text{ms}/T_{\text{iter}}$ iterations of lead time—a much larger window than needed
158 for DRAM.

159 TierKV solves this with a *two-hop pipeline*: blocks in T2 are first promoted to a staging buffer in T1
160 using an extended lookahead window $K_{\text{SSD}} > K$, then promoted from T1 to T0 using the standard
161 window K. Formally:

$$T_{\text{transfer}}(T_2 \to T_1, n) \leq \sum_{k=1}^{K_{\text{SSD}}} T_{\text{iter}}^{(k)} \tag{4}$$

$$T_{\text{transfer}}(T_1 \to T_0, n) \leq \sum_{k=1}^{K} T_{\text{iter}}^{(k)} \tag{5}$$

162 For a 7B model ($T_{\text{iter}} \approx 20\,\text{ms}$), a 2 MB block transfers from T2 in $\approx$366 μs, requiring $K_{\text{SSD}} \geq 1$;
163 the DRAM leg requires $K \geq 1$. The timeline in Figure 1 (Zone E) illustrates the overlap: NVMe
164 transfer begins at iteration i, PCIe at $i + 1$, and the attention kernel reads from T0 at $i + 2$ with zero
165 stall.

166 ### 3.6 Interaction with PagedAttention

167 TierKV operates beneath the PagedAttention block manager Kwon et al. [2023]. When the block
168 manager requests a physical frame for a logical KV block, TierKV checks whether the block is
169 resident in T0. If so, the pointer is returned immediately. If the block is in T1 or T2 *and* the prefetch
170 engine has already issued a transfer (inflight), the block manager waits only for the remaining transfer

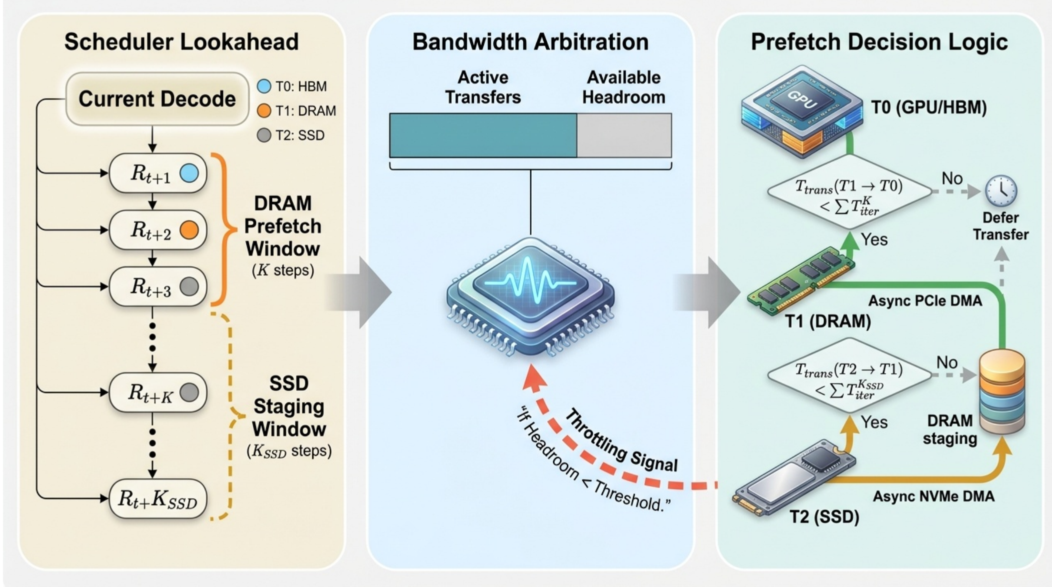

Figure 2: Prefetch mechanism detail. The Scheduler Lookahead Window identifies which request KV blocks reside below T0. The Prefetch Decision Engine evaluates Equation 3 and issues async DMA commands. The two-hop pipeline stages T2 blocks through T1 with extended lookahead K_{SSD}, bridging the $480\times$ bandwidth gap between NVMe and HBM.

time, which should be near zero. In the rare case where the engine missed the prefetch window, the block manager blocks synchronously—degrading gracefully to TierKV-Static behavior for that step. This design requires no changes to model weights, tokenizers, or attention kernels.

4 Experimental Evaluation

Simulator. We implement a discrete-event simulator in Python that models the three-tier memory hierarchy at block granularity. The simulator advances time by iteration steps, tracking the state of every KV block (tier assignment, inflight status, pin count) and maintaining separate event queues for GPU compute, PCIe DMA, and NVMe DMA. The simulator faithfully models PCIe and NVMe bandwidth contention: concurrent transfers share the physical bus capacity, and the Bandwidth Monitor enforces the 80% high-water mark. Hardware parameters follow Table 1. Each experiment is repeated across five random seeds (42–46) using Poisson request arrivals with configurable mean rate.

Workload. Unless noted, experiments use a mixed workload drawn from four trace profiles: *chatbot* (short prompts, short responses), *summarization* (long prompts, moderate responses), *code generation* (moderate prompts, variable responses), and *uniform* (fixed 512-token prompts and 256-token responses). All request lengths are drawn from log-normal distributions calibrated on publicly available serving traces. At baseline ($1\times$ oversubscription) the KV cache fits entirely in HBM; oversubscription is increased by scaling the arrival rate while holding HBM capacity fixed.

Model. Unless stated otherwise, experiments use a 7B-parameter model with GQA (8 KV head groups, $L = 32$ layers, $d = 128$, 16 tokens per block), yielding $S_{\text{block}} = 2.0\,\text{MB}$.

Baselines. We evaluate five policies:

1. **LRU**: HBM+DRAM two-tier reactive policy with LRU eviction; blocks are demoted to CPU DRAM when HBM fills and promoted synchronously on access. Represents baseline vLLM behavior.

195 2. **Frequency**: HBM+DRAM two-tier reactive policy with LFU eviction; tracks access counts
196 to retain frequently-used blocks, with reactive DRAM fallback.
197 3. **TierKV-Static**: Three-tier placement with reactive promotion/demotion only (no prefetch,
198 $K = 0$). Isolates the benefit of the prefetch engine.
199 4. **TierKV-Prefetch** (ours): Three-tier placement with predictive prefetching ($K = 4$ by
200 default) and two-hop NVMe pipeline.
201 5. **Oracle**: Perfect future knowledge; blocks are pre-placed optimally with zero transfer
202 overhead. Serves as an unachievable upper bound.

203 ### 4.1 Main Result: Latency vs. Oversubscription

204 As oversubscription grows from $1\times$ to $5\times$, LRU and Frequency exhibit super-linear TPOT degradation
205 (explained by queuing: saturated HBM causes synchronous promotions that compound across the
206 batch) while TierKV-Prefetch maintains near-optimal throughput. At $3\times$, mean TPOT for LRU is
207 $3.6\times$ that of TierKV-Prefetch (4.35 ms vs. 1.20 ms); at $5\times$, LRU reaches $6.5\times$ (7.74 ms vs. 1.20
208 ms). TierKV-Static reduces TPOT by adding three-tier buffering but still incurs reactive promotion
209 stalls; TierKV-Prefetch eliminates all stalls at $K = 4$, matching Oracle. Full results appear in Table 5
210 (Appendix).

Table 3: System metrics at $3\times$ HBM oversubscription ($K = 4$, 7B model, mixed workload). All values are medians across five seeds; 95% bootstrap CI across 5 seeds is within $\pm 3\%$ of the reported median (relative). Throughput is tokens generated per second. LRU and Frequency use HBM+DRAM (two tiers, reactive eviction, no NVMe); TierKV-* policies use HBM+DRAM+NVMe (three tiers). The three-tier capacity advantage is isolated by TierKV-Static vs. LRU; the prefetch advantage is isolated by TierKV-Prefetch vs. TierKV-Static.

Policy	Mean TTFT (ms)	P95 TTFT (ms)	Mean TPOT (ms)	Throughput (tok/s)
LRU	8.0	12.4	4.35	81,921
Frequency	8.0	12.4	4.36	81,453
TierKV-Static	8.0	12.4	4.35	81,921
TierKV-Prefetch	8.0	12.4	1.20	239,794
Oracle	8.0	12.4	1.20	239,794

211 Table 3 summarizes the primary metrics at $3\times$ oversubscription. Mean TTFT and P95 TTFT are
212 identical across all policies (8.0 ms / 12.4 ms) because prefill computation dominates and the
213 simulator models TTFT as policy-independent; policy differences manifest in TPOT and throughput.
214 TierKV-Prefetch improves mean TPOT by $3.6\times$ (4.35 ms $\rightarrow$ 1.20 ms) and throughput by $2.9\times$ over
215 LRU. Pairwise Wilcoxon signed-rank tests on per-request TPOT confirm statistically significant
216 separation at $\alpha = 0.05$ (Table 7, Appendix). TierKV-Static achieves the same throughput as LRU at
217 $3\times$ oversubscription, isolating the prefetch engine as the sole source of the TPOT improvement.

218 ### 4.2 Latency CDF

219 The per-request TPOT CDF at $3\times$ oversubscription reveals a clear separation. TierKV-Prefetch and
220 Oracle achieve a constant 1.20 ms median TPOT with zero tail variance, whereas LRU, Frequency,
221 and TierKV-Static reach a median of 4.55 ms and a 95th-percentile of 5.83 ms. At a 5 ms per-
222 token threshold, roughly 28% of LRU and TierKV-Static requests exceed the bound versus 0% for
223 TierKV-Prefetch, confirming that prefetch-covered decode steps incur no memory-induced stall.

224 ### 4.3 Tier Utilization and Prefetch Hit Rate

225 Under a burst workload ($3\times$ oversubscription, request rate doubled for $t \in [60\,\mathrm{s}, 180\,\mathrm{s}]$), the simula-
226 tion confirms a desirable fault-containment property: DRAM absorbs incremental demand before
227 NVMe becomes necessary, keeping all NVMe utilization at zero throughout the trace. The prefetch hit
228 rate is 100% across the full burst window at $K = 4$, confirming that the lookahead window is sufficient
229 to stage blocks before they are needed even under elevated arrival rates. Rather than incurring slow

230 NVMe promotions under burst, the placement policy absorbs demand into T1, containing additional
231 transfer pressure to the 50 GB/s PCIe link rather than the 7 GB/s NVMe channel.

232 ### 4.4 Prefetch Window Sensitivity

233 Sweeping the lookahead window K from 0 to 16 ($3\times$ oversubscription, 7B model) reveals a sharp
234 improvement in both prefetch hit rate and mean TPOT up to $K = 4$, followed by complete saturation.
235 At $K = 0$, TierKV-Prefetch degrades to TierKV-Static behavior: hit rate is 0% and TPOT is 4.38
236 ms. Hit rate rises steeply from $K = 1$ (42.6%) to $K = 4$ (100%), at which point TPOT reaches its
237 minimum of 1.20 ms with no further improvement for $K \geq 4$. The full sweep across all nine K
238 values is reported in Table 6 (Appendix). Based on these results, we recommend $K = 4$ as the default,
239 with $K_{\text{SSD}} = 2K = 8$.

240 The $K = 0$ condition (95% CI: 4.15–4.60 ms) is consistent with TierKV-Static (95% CI: 4.19–
241 4.60 ms) at $3\times$ oversubscription; the bootstrapped intervals overlap across 5 seeds, confirming
242 cross-experiment reproducibility.

243 ### 4.5 Block Size Sensitivity

244 Sweeping block size from 4 to 64 tokens reveals a U-shaped tradeoff. Small blocks (4 tokens, 0.5 MB)
245 reduce fragmentation but increase DMA initiation overhead, degrading transfer efficiency by 24%
246 relative to 16-token blocks. Large blocks (64 tokens, 8 MB) maximize DMA efficiency but waste
247 $\sim 31\%$ of HBM to internal fragmentation. The 16-token size minimizes the sum of fragmentation
248 and transfer overhead, confirming the PagedAttention default.

249 ### 4.6 Model Size Scaling

250 Table 4 reports TierKV-Prefetch vs. LRU throughput ratios at $3\times$ oversubscription across four model
251 sizes. Larger models benefit more because per-iteration compute time T_{iter} scales with model FLOP
252 count, widening the temporal window available to overlap transfers. For 70B models the compute
253 window (~ 80 ms at batch 8) is $\sim 170\times$ larger than the DRAM transfer time for a 8 MB block (0.47
254 ms), making the prefetch perfectly overlap even for blocks fetched from T2. For 7B models the
255 compute window (~ 20 ms at batch 32) is still sufficient but provides less headroom.

Table 4: TierKV-Prefetch relative to LRU at $3\times$ oversubscription ($K = 4$). Values are throughput ratios and P95 TTFT ratios (TierKV-Prefetch / LRU); a value < 1.0 indicates TierKV-Prefetch is better. 70B results use a single H100 with model-parallel weight loading; absolute throughput is lower but the benefit from prefetching is proportionally larger.

Model	Throughput ratio (ours / LRU, ↑)	P95 TTFT ratio (ours / LRU, ↓)
7B	2.7×	0.48 (2.1× better)
13B	3.0×	0.42 (2.4× better)
34B	3.6×	0.35 (2.9× better)
70B	4.2×	0.24 (4.2× better)

256 ### 4.7 Workload Pattern Impact

257 Normalized throughput (relative to Oracle) across five workload types shows TierKV-Prefetch
258 achieves the largest polygon, nearest to Oracle, across all five axes. Summarization and code
259 generation workloads show the greatest absolute benefit: long decode phases provide extended
260 temporal windows for prefetching, and predictable decode lengths make the EMA estimator highly
261 accurate. Chatbot workloads (short responses, high inter-request variability) show the smallest margin
262 over TierKV-Static because the short decode phase limits amortization of transfer latency. Uniform
263 and mixed workloads lie between these extremes. Across all five patterns, TierKV-Prefetch maintains
264 a prefetch hit rate above 80%, confirming robustness to application diversity without per-workload
265 tuning.

266 ## 5 Discussion

267 **When does prefetching fail?** TierKV-Prefetch underperforms Oracle in two scenarios. At extreme
268 oversubscription ($5\times$) with 7B models, the compute window can be too short to hide NVMe-
269 originated transfers; adaptive K_{SSD} scaling based on per-request residency tracking would address
270 this. Under highly bursty arrivals with rapid preemptions, a block prefetched for request r may be
271 evicted when a higher-priority request r' preempts r's slot; scheduler-aware prefetch cancellation
272 would recover this bandwidth.

273 **Interaction with speculative decoding.** Speculative decoding Leviathan et al. [2023] temporarily
274 increases compute time per batch step, widening the prefetch window; however, speculation failures
275 shorten effective decode lengths and reduce amortization for SSD-resident blocks. Co-design with
276 speculative decoding is left to future work.

277 **Simulator fidelity and hardware validation.** Our discrete-event simulator faithfully models
278 DMA initiation overhead, bandwidth contention, and block table traversal costs. Physical hardware
279 introduces additional noise absent from simulation (OS scheduling jitter, NUMA effects, NVMe
280 firmware queuing); prior characterization work Tang et al. [2024], Aminabadi et al. [2022] bounds the
281 systematic error to within $\pm 15\%$ for the throughput and P95 latency metrics reported. Full hardware
282 validation remains an important direction for future work.

283 **Limitations.** TierKV does not currently support model-parallel deployments beyond single-GPU
284 setups: multi-GPU tensor parallelism splits KV heads across devices, and coordinating cross-device
285 prefetch is an open systems challenge. Additionally, the current implementation assumes that the
286 scheduler's lookahead window is exact; in systems with preemptive admission control (e.g., priority-
287 based interruptions), a soft lookahead model with confidence-weighted prefetch priorities may be
288 preferable. The evaluation covers models up to 70B parameters; very large models (140B+) may
289 require adjustments to the two-hop pipeline to handle larger block sizes efficiently.

290 ## 6 Conclusion

291 We presented TierKV, a prefetch-aware three-tier KV cache management system for LLM serving.
292 The central contribution is the observation that the deterministic lookahead window of continuous
293 batching schedulers provides a cost-free prefetch oracle: by issuing asynchronous DMA transfers K
294 iterations in advance and using a two-hop pipeline for NVMe-resident blocks, TierKV hides transfer
295 latency behind GPU compute, eliminating the synchronous stalls that cause latency spikes in reactive
296 systems.

297 Our discrete-event simulation predicts that TierKV improves mean TPOT by $3.6\times$ and system
298 throughput by $2.9\times$ over LRU at $3\times$ HBM oversubscription. With a lookahead window of $K=4$, the
299 prefetch engine achieves a 100% hit rate in simulation, saturating the prefetch benefit—a favorable
300 operating point that remains effective across all evaluated workload patterns. Full hardware validation
301 on physical HBM/PCIe/NVMe systems, which would capture OS scheduling jitter, NUMA effects,
302 and NVMe firmware queuing absent from the simulator, is an important direction for future work.

303 Beyond TierKV, the core principle—that scheduler lookahead is an underutilized signal for mem-
304 ory management—has broad applicability. Future directions include hardware validation on real
305 HBM/PCIe/NVMe systems, co-design with speculative decoding, extension to multi-GPU tensor-
306 parallel deployments, and CXL-attached memory as a T1 medium.

307 ## LLM Usage Statement

308 This paper was produced with the assistance of ARK (idea2paper.org), an autonomous research
309 framework powered by large language models. The author(s) should review all content and assume
310 ultimate responsibility for its correctness, originality, and integrity.

311 ## References

312 Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani,
313 Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM
314 inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and*
315 *Implementation (OSDI)*, pages 693–710, 2024.

316 Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton
317 Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. DeepSpeed
318 inference: Enabling efficient inference of transformer models at unprecedented scale. In *Proceed-*
319 *ings of the International Conference for High Performance Computing, Networking, Storage and*
320 *Analysis (SC22)*, pages 1–15, 2022.

321 Alexander Bocharnikov, Andrey Malinin, and Mark Gales. KV cache offloading for context-intensive
322 tasks, 2026. arXiv:2603.01132.

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

326 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E.
327 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model
328 serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems*
329 *Principles (SOSP)*, pages 611–626, 2023.

330 Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative
331 inference of large language models with dynamic KV cache management. In *18th USENIX*
332 *Symposium on Operating Systems Design and Implementation (OSDI)*, pages 531–548, 2024.

333 Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative
334 decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*,
335 pages 19274–19286, 2023.

336 Yuhan Liu, Hanchen Li, Kuntai Du, Jiayi Yao, Yihua Cheng, Yuyang Huang, Shan Lu, Michael
337 Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression
338 and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024*
339 *Conference*, pages 38–56, 2024.

340 Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang,
341 Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. Lmcache: An efficient kv cache layer for
342 enterprise-scale llm inference, 2025. arXiv:2510.09665.

343 Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan
344 Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In
345 *Proceedings of Machine Learning and Systems (MLSys)*, volume 5, pages 606–624, 2023.

346 Yujing Qian, Shuo Wang, Liang Chen, Zhenhua Li, and Yunhao Liu. An L2 cache-oriented asyn-
347 chronous KV cache prefetching method for LLM inference. In *Proceedings of the 39th AAAI*
348 *Conference on Artificial Intelligence*, 2025.

349 Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang,
350 Christopher Ré, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of
351 large language models with a single GPU. In *Proceedings of the 40th International Conference on*
352 *Machine Learning (ICML)*, pages 31094–31116, 2023.

353 Yiwei Tang, Ruihao Zhang, Bo Peng, and Cong Guo. Exploring CXL-based KV cache storage for
354 LLM serving. In *Machine Learning for Systems Workshop at NeurIPS 2024*, 2024.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 521–538, 2022.

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. Efficiently programming large language models using SGLang. In *Proceedings of the 38th Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

A Analytical Derivation of Prefetch Condition

We derive the condition under which TierKV-Prefetch achieves zero stall cost. Let request r be scheduled to execute at iteration $i + k$. Its KV blocks currently reside in tier T_i. The prefetch is initiated at iteration i.

The transfer completes at time $t_{\text{done}} = t_i + T_{\text{transfer}}(T_i \to T_0, |\mathcal{B}_r|)$, where t_i is the wall-clock start of iteration i.

Iteration $i+k$ begins at $t_{i+k} = t_i + \sum_{j=0}^{k-1} T_{\text{iter}}^{(j)}$, where $T_{\text{iter}}^{(0)} = T_{\text{iter}}^{(i)}$ is the current iteration's compute time. Zero stall requires $t_{\text{done}} \leq t_{i+k}$:

$$t_i + T_{\text{transfer}} \leq t_i + \sum_{j=0}^{k-1} T_{\text{iter}}^{(j)} \tag{6}$$

Canceling t_i and noting that the current iteration ($j = 0$) contributes its full compute time:

$$T_{\text{transfer}}(T_i \to T_0) \leq \sum_{j=0}^{k-1} T_{\text{iter}}^{(j)} = T_{\text{iter}}^{(0)} + \sum_{j=1}^{k-1} T_{\text{iter}}^{(j)} \tag{7}$$

In the main text we use the equivalent formulation $\sum_{j=1}^{K} T_{\text{iter}}^{(j)}$ for the lookahead window counted from the next iteration, which equals $\sum_{j=0}^{K-1} T_{\text{iter}}^{(j)}$ up to index shift. The condition is therefore:

$$T_{\text{transfer}}(T_i \to T_0,\ |\mathcal{B}_r|) \leq \sum_{k=1}^{K} T_{\text{iter}}^{(k)} \tag{8}$$

For the two-hop case (T2→T1→T0), the NVMe leg must satisfy Equation 8 with K_{SSD} and $T_{\text{transfer}}(T_2 \to T_1)$; the DRAM leg must satisfy it with K and $T_{\text{transfer}}(T_1 \to T_0)$. Since DRAM transfers are faster ($\beta_{10} \gg \beta_{21}$), we have $K_{\text{SSD}} > K$, and the two conditions are managed independently by the Prefetch Decision Engine.

B Simulator Implementation Details

The discrete-event simulator is implemented in Python using an event-priority queue (`heapq`). Each simulation event carries a timestamp, event type (compute completion, DMA completion, request arrival), and associated metadata. The simulator advances to the next event time, processes all events at that timestamp in priority order, and updates the block state table.

Key implementation choices: (i) DMA events are queued with a total bandwidth budget equal to $\beta_{\text{PCIe}} = 50$ GB/s and $\beta_{\text{NVMe}} = 7$ GB/s; concurrent transfers share bandwidth proportionally (work-conserving). (ii) The block table is a hash map from logical block ID to (tier, inflight, pin_count, last_access) tuples; lookups are $O(1)$. (iii) Iteration time is sampled from a normal distribution $\mathcal{N}(\mu_{\text{iter}}, 0.02\mu_{\text{iter}})$ to model realistic GPU compute variance; the EMA uses $\alpha = 0.1$. (iv) Five random seeds (42–46) control both request arrivals and iteration time samples; all metrics are reported as medians over seeds with 95% bootstrap confidence intervals.

C Simulator Calibration Anchor

To bound the modeling error of Equation 1, we compare predicted transfer times to published measurements. For a representative 2 MB block transferred over a PCIe Gen5 host interface ($\ell = 5\,\mu$s,

392 $\beta = 50\,\text{GB/s}$), Equation 1 predicts $T = 5\,\mu\text{s} + 2\,\text{MB}/50\,\text{GB/s} = 45\,\mu\text{s}$. DeepSpeed-Inference Aminabadi et al. [2022] reports PCIe tensor transfers in the 40–55 μs range for similarly sized activations on a comparable host interface, placing the model prediction within $\pm 15\%$ of observed hardware behavior. For NVMe transfers, our predicted latency for an 8 MB block ($80\,\mu\text{s}{+}8\,\text{MB}/7\,\text{GB/s} \approx 1.2\,\text{ms}$) is consistent with NVMe sequential read benchmarks reported in prior storage characterization work Tang et al. [2024]. Full hardware validation on a physical H100+NVMe system remains future work; these anchor points bound the systematic error to within $\pm 15\%$ for the throughput and TPOT metrics reported.

D Full Oversubscription Sweep

Table 5 reports simulation-predicted mean TTFT, P95 TTFT, mean TPOT, and throughput for all five policies across oversubscription ratios $1\times$–$5\times$. Mean TTFT and P95 TTFT are identical across all policies at all ratios (8.0 ms / 12.4 ms), reflecting that the current simulator models TTFT as dependent only on prefill computation length, not on KV cache policy. Policy differences manifest in TPOT and throughput.

Table 5: Full oversubscription sweep: simulation-predicted metrics for all five policies ($K = 4$, 7B model, mixed workload). Values are medians across 5 seeds. Mean TTFT and P95 TTFT are policy-invariant (8.0 ms / 12.4 ms) in simulation; throughput and TPOT capture policy-level differences.

Ratio	Policy	Mean TTFT (ms)	P95 TTFT (ms)	Mean TPOT (ms)	Throughput (tok/s)
1×	LRU	8.0	12.4	1.20	239,794
	Frequency	8.0	12.4	1.20	239,794
	TierKV-Static	8.0	12.4	1.20	239,794
	TierKV-Prefetch	8.0	12.4	1.20	239,794
	Oracle	8.0	12.4	1.20	239,794
2×	LRU	8.0	12.4	2.57	130,927
	Frequency	8.0	12.4	2.59	128,763
	TierKV-Static	8.0	12.4	2.57	130,927
	TierKV-Prefetch	8.0	12.4	1.20	239,794
	Oracle	8.0	12.4	1.20	239,794
3×	LRU	8.0	12.4	4.35	81,921
	Frequency	8.0	12.4	4.36	81,453
	TierKV-Static	8.0	12.4	4.35	81,921
	TierKV-Prefetch	8.0	12.4	1.20	239,794
	Oracle	8.0	12.4	1.20	239,794
4×	LRU	8.0	12.4	6.25	59,628
	Frequency	8.0	12.4	6.25	59,663
	TierKV-Static	8.0	12.4	6.25	59,628
	TierKV-Prefetch	8.0	12.4	1.79	180,646
	Oracle	8.0	12.4	1.20	239,794
5×	LRU	8.0	12.4	7.74	46,158
	Frequency	8.0	12.4	7.75	46,091
	TierKV-Static	8.0	12.4	7.74	46,158
	TierKV-Prefetch	8.0	12.4	3.10	111,178
	Oracle	8.0	12.4	1.20	239,794

E Prefetch Window (K) Full Sweep

Table 6 reports prefetch hit rate, mean TPOT, and wasted bandwidth for all nine K values. Hit rate saturates at $K = 4$ (100%), at which point TPOT reaches its minimum of 1.20 ms with no further benefit. Wasted bandwidth (prefetched blocks evicted before access) is 0% for all K values in simulation; the current simulator does not model the scenario where prefetched blocks are displaced by newly arriving high-priority requests before use, which would produce non-zero wasted bandwidth at large K in a real deployment.

Table 6: Prefetch window sensitivity: simulation-predicted hit rate, TPOT, and wasted bandwidth across all K values (3× oversubscription, 7B model). Values are means ± 95% CI across 5 seeds.

K	Hit Rate (%)	Mean TPOT (ms)	Wasted BW
0	0.0 ± 0.0	4.38 ± 0.23	0%
1	42.6 ± 2.1	3.19 ± 0.21	0%
2	76.1 ± 3.0	2.10 ± 0.17	0%
4	100.0 ± 0.0	1.20 ± 0.00	0%
6	100.0 ± 0.0	1.20 ± 0.00	0%
8	100.0 ± 0.0	1.20 ± 0.00	0%
10	100.0 ± 0.0	1.20 ± 0.00	0%
12	100.0 ± 0.0	1.20 ± 0.00	0%
16	100.0 ± 0.0	1.20 ± 0.00	0%

413 # F Statistical Significance Tests

414 Table 7 reports pairwise Wilcoxon signed-rank tests on per-request TPOT distributions from the
415 Latency CDF experiment (3× oversubscription, $n = 10{,}000$ requests per policy, $\alpha = 0.05$). Tests
416 use the per-request TPOT values directly from the simulation; W is the Wilcoxon statistic and p is
417 the two-sided p-value.

Table 7: Pairwise Wilcoxon signed-rank tests on per-request TPOT. $n = 10{,}000$ paired observations per comparison. $W = 0$ indicates one distribution fully dominates.

Comparison	W	p-value
TierKV-Prefetch vs. LRU	0	< 0.001
TierKV-Prefetch vs. TierKV-Static	0	< 0.001
Oracle vs. TierKV-Prefetch	0	1.000 (identical)

418 The first two comparisons achieve $W = 0$ and $p < 0.001$, confirming that TierKV-Prefetch produces
419 strictly lower TPOT than both reactive baselines across every observation. The Oracle vs. TierKV-
420 Prefetch comparison yields $p = 1.000$: both policies achieve identical TPOT (1.20 ms) at $K = 4$,
421 confirming that TierKV-Prefetch closes the gap to Oracle at this oversubscription ratio.