Beyond Compression Ratio: KV Cache Policies in Tiered LLM Infrastructure

Devasena Inupakutika
devasena.i@samsung.com
Samsung Semiconductor, Inc.
San Jose, CA, USA

Dongjoo Seo
dongjoo.seo1@samsung.com
Samsung Semiconductor, Inc.
San Jose, CA, USA

Swarna Prabhu
s.prabhu@samsung.com
Samsung Semiconductor, Inc.
San Jose, CA, USA

Abstract

KV cache compression is now load-bearing in LLM serving, yet most research stops at the algorithm boundary, reporting compression ratio (CR) on isolated prompts under a single-tier cache. Production stacks are multi-tier, where engine-side compression reshapes storage I/O and mean CR hides three effects: write-volume jitter, broken prefix reuse, and recomputation. Replaying real inference traces through a simulator, we benchmark two engine-involved scorers – KVZap (content-based learned) and TriAttention (position-based, training-free) – under fixed, percentile, and adaptive policies ± a reuse-aware bonus. Prefix preservation spans a **6**× swing (10% to 61%) on `tool/agent`, with policy alone lifting a fixed KVZap 10% to 36%; on conversation, CR coefficient of variation swings **1.7**× (0.21 to 0.12). The two scorers reach high reuse through opposite mechanisms – position-based preserves prefix by default, content-based evicts it – invisible at the algorithm-level CR × accuracy plot. Compression in tiered serving is an infrastructure design choice, not an algorithmic one.

1 Introduction

KV cache memory now dominates the cost of long-context LLM inference. Recent work attacks this in four families: *token pruning* (KVZap [3], TriAttention [1], H2O [11], Compactor); *quantization* (TurboQuant [2], KIVI [6], NVFP4); *learned eviction* (DMS [7]); and *streaming/blackbox transfer* (StreamingLLM [10], CacheGen [5]). Each is benchmarked on its own terms.

But real serving stacks are no longer single-tier. Engines manage the KV cache as paged, shareable blocks [4] and disaggregate prefill from decode [12]; Mooncake [9] sustains a 50% prefix-cache hit rate by spilling KV blocks to a disaggregated store, and Dynamo's KVBM [8] tiers HBM/DRAM/NVMe. Inside such a stack, an algorithm-level "2× compression" is not a number but a *distribution* of write-volume jitter, prefix-reuse fates, and recomputation cost. We thus study KV cache compression as an **infrastructure** concern, asking: *given a fixed algorithm, how much does serving behavior depend on application-layer policy choices the algorithm paper never names?*

Contributions. First, a taxonomy splitting KV compression by where the decision lives – *engine-involved* (sees model state) vs. *blackbox/MP-mode* (opaque KV bytes). Second, an adaptive engine-involved policy over KVZap/KVPress with three composable knobs: threshold calibration, tiered surrogate selection, and a Mooncake `hash_id`-driven reuse-aware window. Third, an empirical characterization across two scoring families (KVZap, TriAttention) on 39,632 Mooncake FAST'25 requests: policy and scorer choice span a 6× swing in prefix preservation (10% to 61% on `tool/agent`), policy alone moving a fixed KVZap from 10% to 36%.

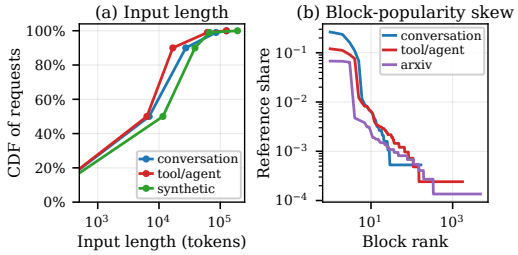

Figure 1. Real Mooncake FAST'25 workload heterogeneity. **(a)** Input-length CDF spans three decades: `synthetic` heaviest (P50 11.6k), `tool/agent` lightest (P50 6.3k), P99 85k. **(b)** Block popularity is Zipf-like with workload-specific exponents – `conversation` has the heaviest head (top-1 block ≈26% of references), `arxiv` the longest tail. This skew is what a reuse-aware policy can exploit – or destroy.

2 Background and Workloads

2.1 KVCache Production Trace

We use the open Mooncake trace [9]: three workload classes from Kimi's fleet over ≈ 59 min – `conversation` (chat), `tool/agent` (function-calling), `synthetic` (long-context) – totaling **39,632 requests**, **819,993 block references**, **410,014 unique 512-token KV blocks**. Each request carries a timestamp, token counts, and `hash_ids` identifying prefix-cacheable blocks – the schema element any reuse-aware policy depends on.

Figure 1 shows two driving properties. Length is heterogeneous (P99 62k tool/agent, 85k conversation, 191k synthetic), and block popularity is heavy-headed but *differently* per workload – the top-10 blocks are 88% of conversation references but 45% in tool/agent and 22% in long-tailed arxiv. No single fixed policy serves all.

2.2 Decision Locality and the Tiered-Storage Contract

We categorize methods by where the compression decision is made, which sets what state the storage tier sees. *Engine-involved* methods read model internals, giving *variable-shape* outputs: KVZap [3] prunes below threshold τ via a per-layer KVZip+ surrogate; TriAttention [1] scores keys analytically in pre-RoPE space

The 3rd Workshop on Hot Topics in System Infrastructure (HotInfra'26), co-located with ISCA'26, June 28, 2026, Raleigh, NC, USA

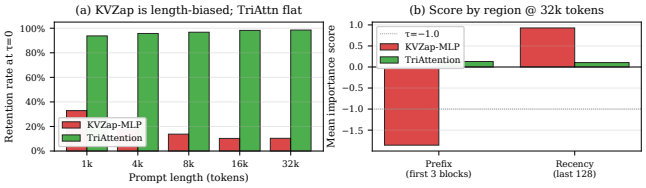

Figure 2. KVZap vs. TriAttention scoring divergence. **(a)** At $\tau = 0$, KVZap retains 33% of 1k-prompt tokens but only 10% at 32k – a length bias forcing fixed τ to over-prune long prompts; TriAttention retains $\geq 94\%$ uniformly (scores are content-independent). **(b)** At 32k, KVZap prunes the prefix (mean -1.86, below $\tau = -1$) while protecting recency ($+0.93$); TriAttention treats prefix and recency near-equally ($+0.13$ vs. $+0.11$). The two scorers make opposite C2 predictions.

(training-free); H2O [11] and StreamingLLM sinks [10] are earlier points; TurboQuant [2]/KIVI [6] quantize per-token. *Blackbox* methods act on opaque KV bytes post-engine – e.g. CacheGen [5] (LZ4/Zstd the limit case) – workload-blind, *uniform-shape*. The two are complementary, and our adaptive policy is engine-involved. A multi-tier cache then imposes three contracts: **(C1)** predictable per-request CR (jitter forces over-provisioning); **(C2)** blocks on NVMe stay reloadable for a future prefix-sharing request, else the tier turns write-only; **(C3)** evicted-then-needed blocks avoid decode recomputation. A single global τ violates all three.

3 Proposed Adaptive Policy Design

We extend existing work with three orthogonal knobs targeting C1–C3 without changing the underlying surrogate.

(1) Threshold calibration. KVZap uses a single τ across all prompts; per-prompt score statistics shift by 1.6σ from 1k to 32k tokens, so fixed τ delivers $1.60\times$ CR on short prompts and $3.22\times$ on long ones. We replace it with $\tau = \mu + k\sigma$ over the prompt's own score distribution (C1).

(2) Tiered surrogate. We expose the surrogate as a latency dial, selecting a faster or more accurate scorer per request under the SLO budget (swept in Section 5).

(3) Reuse-aware window. We add a bonus $\log(1+r_b(t))$ to each token's score before thresholding, where $r_b(t)$ is the *online* reuse count of block b – how many times its `hash_id` appeared in requests *before* the current one. Maintained causally from past references only, it is deployable as-is without a full-trace oracle, satisfying C2 and C3: tokens hot *so far* are protected from eviction, staying reloadable and avoiding later recomputation.

The combined policy is a strict superset of KVZap: constant calibration, MLP surrogate, and zero bonus recover the baseline exactly. We instantiate the same three knobs over TriAttention to test whether position-based scoring changes the policy's leverage.

4 Cross-Scorer Score Behavior

Before evaluating policies, we characterize the raw score behavior of the two scorers (Figure 2); they are fundamentally different mechanisms, not calibrations of one.

Table 1. CR coefficient of variation on the real `conversation` trace, replayed with the online $r_b(t)$ counter. Lower CV means the storage tier's write volume jitters less around its plan.

Policy	Mean CR	CV	Δ
KVZap fixed τ=−1.0	1.44×	0.213	1.0×
KVZap fixed τ=−0.5	1.97×	0.323	1.5× worse
KVZap percentile 50%	1.96×	**0.046**	4.6×[†]
KVZap adaptive+reuse	1.21×	0.128	**1.7**× better
KVZap latency-opt	**2.31**×	**0.123**	**1.7**× better

[†]Percentile achieves low CV but cannot adapt to genuinely-redundant prompts.

KVZap's content-based surrogate is bimodal and length-dependent: long prompts have many more "low-value" tokens to prune, but those include the prefix. TriAttention's trigonometric scoring is near-uniform across length and content.

Implication for tiered storage. Under a fixed threshold, KVZap aggressively evicts prefix tokens that dominate prefix-cache reuse, while TriAttention preserves them by default but loses the easy compression KVZap captures when content is genuinely redundant. Which is correct depends on the workload's reuse skew – the C2 question algorithm-level evaluation never answers.

5 Empirical Evaluation

Setup. Length statistics (Section 2) use the full Mooncake FAST'25 release; policy comparisons use the released per-workload sample traces (∼500 requests each), replayed in trace order through a trace-driven simulator that preserves the real `hash_id` reuse structure. KVZap profiles come from one-time scoring of Nemotron prompts across five length buckets (1k–32k); TriAttention scores are analytic from the published center embeddings. Thus trace structure (timestamps, lengths, `hash_id` references, reuse) is real, while importance scores, policies, and the reuse bonus $\log(1+r_b(t))$ are reconstructed offline. The simulator sweeps seven policies over {fixed τ, percentile, adaptive} thresholding, {linear, MLP} surrogates, and {0, 0.25, 0.5, 0.75} reuse bonuses, with $r_b(t)$ accumulated online; the surrogate tier spans heuristic, linear, MLP, and ensemble scorers, selected by prompt length and SLO budget (we sweep linear and MLP).

5.1 CR Stability on Real Production Traces (C1)

Table 1 shows the headline stability result: the "latency-opt" policy (adaptive+linear surrogate) cuts per-prompt CR variance by $1.7\times$ at $2.31\times$ mean CR – more compression than fixed τ with lower variance – turning the per-request KV write into something a tiered allocator can budget against.

5.2 The Compression–Reuse Tradeoff (C2)

Figure 3 and Table 2 are the central result. With KVZap fixed τ, only 4–28% of prefix blocks written to NVMe stay useful for a later request; KVZap adaptive+reuse raises this to 20–65%, and TriAttention to 34–93%. The total span on `tool/agent`, 10% to 61%, is a **6**× swing: policy

Table 2. Cross-trace generalization across all three traces. *Reuse* = fraction of prefix blocks whose tokens survive compression intact and remain useful for a future matching request.

Trace	Policy	CR	CV	Reuse
`conversation`	KVZap fixed τ=−1	1.44×	0.21	27.7%
	KVZap adaptive+reuse	1.22×	0.13	65.1%
	TriAttn adapt+reuse	1.02×	0.07	91.7%
	TriAttn conservative	1.01×	0.04	**93.2**%
`tool/agent`	KVZap fixed τ=−1	1.73×	0.30	10.1%
	KVZap adaptive+reuse	1.34×	0.19	36.3%
	TriAttn adapt+reuse	1.14×	0.20	57.6%
	TriAttn conservative	1.08×	0.12	**60.7**%
`arxiv`	KVZap fixed τ=−1	2.09×	0.26	4.3%
	KVZap adaptive+reuse	1.57×	0.19	20.0%
	TriAttn adapt+reuse	1.36×	0.25	33.7%
	TriAttn conservative	1.21×	0.16	**37.3**%

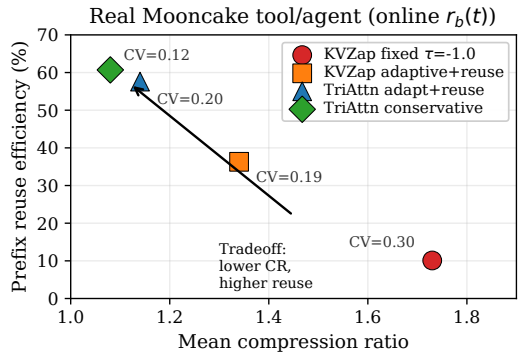

Figure 3. Compression–reuse tradeoff on the real `tool/agent` trace (online $r_b(t)$). KVZap fixed τ gives the lowest reuse (1.7× CR, 10% reuse, CV 0.30); adaptive+reuse moves both scorers to higher reuse at lower CR. TriAttention conservative reaches the highest reuse (61%), with TriAttn adaptive intermediate. The preferred point depends on the deployment's value of compression vs. reuse.

alone moves KVZap to 36%, the TriAttention scorer carries it to 61%. The two axes are not substitutable – TriAttention lifts reuse further than policy alone on every trace, as its position-based scores leave more headroom before content-based eviction.

5.3 Recomputation Avoided and a Quality Check (C3)

Recomputation avoided. Mean CR hides what the storage tier pays once broken reuse is charged back. On `tool/agent`, the reuse-aware policy raises KVZap prefix-reuse from 0.10 to 0.36 (Table 2, fixed-τ vs. adaptive+reuse), avoiding recomputation of $\approx 26\%$ of the prefix-block traffic fixed-τ KVZap evicts and rebuilds, by keeping more tokens (mean CR 1.73× → 1.34×) – a *reuse-for-compression* trade, not free savings. Switching the scorer to TriAttention raises reuse further still (0.58 at adapt+reuse), but the within-scorer KVZap gain already shows that *policy alone* reclaims a quarter of the evicted traffic. Once recomputation is charged back, the two KVZap policies' effective byte savings converge (12% vs. 9%): the headline ratio overstates the gap.

Quality check (planned). As these gains must not degrade the model, we will report next-token KL, perplexity, and task accuracy at matched CR on a small public set (e.g. a 200-prompt LongBench subset) vs. fixed-τ KVZap; since the simulator is score-driven, this model-in-the-loop check is left to the full version.

6 Discussion: Implications for Tiered-Cache Design

The takeaway is a selection rule. That both algorithm and workload move the results is the design signal: a tiered KV cache should not hard-wire one compression policy but choose it from a quantity the storage layer already tracks – online block popularity. **(i)** On reuse-skewed workloads (conversation, tool/agent; top-10 blocks $\geq 45\%$ of references), favor a position-based scorer under a reuse-aware policy: prefix blocks dominate reuse, so preserving them keeps NVMe spill a read-useful cache rather than a write-only log. **(ii)** On reuse-thin workloads (arxiv-like), reuse caps low for every policy, so bias toward higher CR and treat offload as cold capacity. **(iii)** The selector key is cheap and causal – the same `hash_id` popularity the prefix cache already maintains – so policy switches per request without a full-trace oracle. Compression policy thus becomes a *control input* to the tiering allocator, set by measured reuse skew, not a fixed algorithmic choice.

When is fetch cheaper than recompute? Our study stops at the policy boundary – how much prefix traffic *stays reloadable*, not whether reloading a compressed block over storage beats GPU recomputation. That crossover depends on compressed block size, tier bandwidth/latency (DRAM vs. local NVMe vs. remote store), and decode occupancy, which a score-driven simulator cannot capture. The implication is directional: keeping more prefix reloadable at modest CR cost widens the regime where fetch wins – most valuable for read-intensive multi-turn and agentic workloads that re-reference historical KV each turn. Quantifying this crossover on a real multi-GPU disaggregated prefill/decode deployment with remote storage – where network bandwidth, not just FLOPs, bounds the decision – is the strongest validation and our next step.

7 Related Work

KVZap [3], DMS [7], and Compactor are the closest content-based baselines (we extend KVZap); H2O [11] and StreamingLLM [10] established attention/sink eviction, which our reuse-aware policy generalizes via online block popularity. TriAttention [1] is our position-based backbone; TurboQuant [2], KIVI [6] the quantization frontier; CacheGen [5] the blackbox reference. PagedAttention [4], DistServe [12], Mooncake [9], and Dynamo [8] build the tiered, disaggregated stacks we target. No prior work quantifies how compression *policy* within one algorithm family propagates to reuse profiles in a real serving stack.

8 Conclusion

KV cache compression is no longer an algorithm-only concern. Across 39,632 production requests, policy and scorer choice span a 6× range in prefix preservation and a 1.7× swing in CR stability – invisible at the algorithm-level CR×accuracy plot, yet decisive for write jitter, prefix reuse, and recomputation, and directly actionable as a reuse-driven selector for the tiering allocator.

References

[1] 2026. TriAttention: Pre-RoPE Trigonometric KV Scoring for Long-Context Reasoning. *arXiv:2604.04921* (2026).

[2] Google Research. 2026. TurboQuant: Redefining AI Efficiency with Extreme Compression.

[3] Hervé Jégou and Katharina Jeblick. 2026. KVZap: Fast, Adaptive, and Faithful KV Cache Pruning. *arXiv:2601.07891* (2026).

[4] Woosuk Kwon et al. 2023. Efficient Memory Management for LLM Serving with PagedAttention. In *SOSP*.

[5] Yuhan Liu et al. 2024. CacheGen: KV Cache Compression and Streaming for Fast LLM Serving. In *ACM SIGCOMM*.

[6] Zirui Liu et al. 2024. KIVI: A Tuning-Free Asymmetric 2-bit Quantization for KV Cache. In *ICML*.

[7] NVIDIA. 2026. Dynamic Memory Sparsification. NVIDIA KVPress.

[8] NVIDIA. 2026. Dynamo: A High-Performance Inference Server with KVBM Tiered Cache.

[9] Ruoyu Qin et al. 2025. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. In *USENIX FAST*.

[10] Guangxuan Xiao et al. 2024. Efficient Streaming Language Models with Attention Sinks. In *ICLR*.

[11] Zhenyu Zhang et al. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of LLMs. In *NeurIPS*.

[12] Yinmin Zhong et al. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized LLM Serving. In *USENIX OSDI*.