

RoleKV - Role-Aware KV Cache Management for the Inverted Age-Importance of LLM Agent Context

Eashan Dash¹, Arun Ramachandran^{1,2}, Prakash Raghavendra¹

¹AMD India Private Limited, ²Indian Institute of Science

eashan.dash@amd.com, aruncoimbatore.ramachandran@amd.com, prakash.raghavendra@amd.com

Abstract

When serving LLM agents in production, the KV cache grows over turns, and mainstream serving engines (e.g., vLLM) rely on least recently used (LRU) eviction to manage memory pressure. While LRU is well-understood for OS pages and single-turn LLM serving, its behavior on *agent* workloads remains underexplored. We present an *inverted age-importance* phenomenon in agent workflows, where the oldest KV blocks (system prompts, tool outputs) are the most persistently attended, while the newest blocks (chain-of-thought reasoning) decay fastest. As a result, recency-based eviction systematically discards the wrong blocks in agent workloads, producing two distinct overheads: (1) wasted prefill work on preempt-discard engines, and (2) wasted PCIe bandwidth on CPU-offload engines. We propose RoleKV, a training-free, lossless eviction policy that evicts KV blocks based on their semantic role in agent workflows. A lightweight Role Tagger assigns each block to a semantic role, and RoleKV then evicts by role-aware decay rather than recency. Unlike model-level KV-compression methods that incur quality loss, RoleKV is a *system-level* policy that decides *where* blocks reside (GPU vs. CPU/disk), without affecting model outputs. RoleKV cuts preemptions by up to 64.3%, PCIe traffic by up to 63.8%, and improves end-to-end latency by up to $1.41\times$ on 8B-70B models, with gains that grow on longer agent traces and persist on multi-GPU setups. This recasts KV eviction in production LLM serving from a recency problem to a role-aware retention problem.

1 Introduction

AI inference is shifting from single-pass model calls to long-running agentic workflows that interleave reasoning, action, and tool execution (Yao et al., 2023b; Shinn et al., 2023; Jimenez et al., 2024; Zhou et al., 2024; Schick et al., 2023; Qin et al., 2024; Yao et al., 2023a). In production de-

ployments, the resulting context regularly exceeds GPU memory capacity, so serving engines manage pressure through an *eviction policy* that decides which KV cache blocks to discard or offload. The universal default is least recently used (LRU), which evicts the oldest-allocated blocks.

Three forces make this choice increasingly consequential. Inference has overtaken training as the dominant AI infrastructure cost, and the KV cache is the binding throughput constraint at production scale (Kwon et al., 2023). Both production responses to KV memory pressure, preempt-discard (e.g., vLLM, SGLang, Orca) and CPU-offload (e.g., LMCache, FlexGen), delegate to the eviction policy the question of what to retain. LRU was suitable for OS pages and single-turn inference. This paper examines whether LRU remains the right default for agent workloads where context grows monotonically and KV blocks carry heterogeneous semantic roles.

Two lines of work approach the KV-pressure problem, with complementary blind spots. *Model-level KV-cache compression* (Zhang et al., 2023; Li et al., 2024; Liu et al., 2024; Xiao et al., 2024) alters how attention is computed, trading output fidelity for memory. These methods are role-agnostic. *Agent-aware serving systems* (Pan et al., 2025; Li et al., 2025; Bian et al., 2025) preserve fidelity but score blocks with age- or step-based proxies that ignore measured attention behavior. Neither grounds retention in *per-role* attention dynamics (§7).

To close this gap, we conduct the first measurement study of per-role KV attention decay in multi-turn LLM agents. Across three model families and four agent tasks, we find that KV-block importance is **inverted with age** (Fig. 1). The oldest blocks, system prompts and tool outputs, account for the bulk of decode-step attention even after dozens of turns, while the newest blocks, chain-of-thought reasoning, decay fastest. The effect is consistent: the spearman correlation between block age and

	LRU	RKV
System (t=0)	✗	✓
Instruction (t=0)	✗	✓
Tool output (t=2)	✗	✓
Reasoning (t=3)	✓	✗
Reasoning (t=5)	✓	✗
Tool output (t=6)	✓	✓
Reasoning (t=7)	✓	✗

✗ = evicted ✓ = kept in GPU

Figure 1: **LRU vs RoleKV under memory pressure.** LRU evicts the oldest blocks, which every future turn still needs. RoleKV evicts transient reasoning instead.

persistent attention mass is at least 0.70 on every task we measure (Table 1). Recency-based eviction therefore systematically discards exactly the blocks the next turn will reattend to.

This points to a simple fix. We propose RoleKV, a lossless, training-free eviction policy that scores each KV block by combining its semantic role with a measured per-role attention-decay curve. A deterministic, host-CPU Role Tagger identifies block roles online, reaching 99.58% block accuracy on HotpotQA with Llama3.1 8B inference. Its residual error only over-retains blocks, never under-retains. Crucially, RoleKV operates entirely at the system layer: it changes only *where* a block lives, GPU or CPU/disk. Hence outputs are bit-identical to a no-eviction baseline.

We integrate RoleKV as a drop-in plugin for vLLM and LMCache without source patches, and evaluate on 8B-70B models. Compared to LRU, RoleKV reduces preemptions by up to **64.3%**, PCIe traffic by up to **63.8%**, and end-to-end latency by up to **1.41×**, without any model changes. The gains hold across HotpotQA, GAIA, and WebArena, grow with agent trace length, and persist from 1 to 8 GPUs. Concretely, this paper contributes (i) the first measurement study of per-role KV attention decay in multi-turn agents (§3) (ii) RoleKV, a role-aware eviction policy and Role Tagger (§4, §4.1) and (iii) a drop-in integration with vLLM and LMCache demonstrating the gains above (§5). In short, we recast KV eviction in production LLM serving from a *recency* problem to a *role-aware retention* problem.

2 Background: The Agent Workload

An LLM agent executes a *turn loop* (Yao et al., 2023b; Shinn et al., 2023; Packer et al., 2023)

where at each step the model reads the accumulated history, emits a reasoning trace, selects an action, calls an external tool, and appends the response. This yields five semantic roles that persist across turns, SYSTEM (set once), INSTRUCTION (user task), REASONING (per-turn chain-of-thought), ACTION (tool call), and TOOL_OUTPUT (response). Production engines store this context as PagedAttention (Kwon et al., 2023) blocks of 16 tokens. Unlike chat, the cache grows *monotonically*, as every turn appends new blocks without removing old ones.

Agentic workloads differ from single-turn inference in three ways. (i) *Heterogeneous re-access* - system prompts and early tool outputs are read at every subsequent turn, while recent reasoning may never be revisited. (ii) *Tool stalls* - external calls introduce 300 ms–10 s pauses during which the KV cache stays fully resident without compute benefit (Jimenez et al., 2024; Zhou et al., 2024). (iii) *Cross-session prefix sharing* - concurrent agent sessions share identical system prompts and instruction prefixes, so retention decisions affect many requests at once. Together, these properties make the eviction policy, rather than model-level compression, the dominant lever on memory pressure.

3 Motivation: Two Observations

3.1 Observation 1: Inverted Age-Importance

Probing. At each agent turn we measure the attention paid by the decode-step token to all prior KV blocks, averaged across the final L_{probe} layers and heads, and partitioned by role tag. We probe four agent tasks (HotpotQA, WebArena, SWE-Bench, GAIA) on three model families and fit per-role exponential decay

$$\bar{a}_r(k) = a_{0,r} \cdot e^{-k/\tau_r} \quad (1)$$

by weighted least squares. R^2 in Table 1 reports the goodness of Eq. 1 per cell (higher $R^2 \rightarrow$ more reliable $t_{1/2}$, probe counts and per-model breakdowns in App. C).

Three patterns stand out from Table 1.

- **Same pattern, different rate.** The role rank {action, reasoning} $\ll$ {instruction} $\ll$ {system, tool_output} is preserved on every task (Spearman $\rho \geq 0.70$, $\rho=1.00$ on WebArena), while absolute τ_r vary by up to 20 $\times$. RoleKV preserves the rank and calibrates τ_r per workload (Eq. 3).

Table 1: **Per-role attention half-lives ($t_{1/2}$, turns) and exponential-fit R^2 across four agent workloads** (Llama-8B, 50 ep./task, bold: shortest- and longest-lived role per column, cross-model data in App. C). Role ordering is preserved (Spearman $\rho \geq 0.70$) while τ_r shifts up to $20\times$. Each ρ ranks the five role half-lives against HotpotQA and is tested by exact permutation ($n=5$). * marks $p < 0.05$: WebArena ($\rho=1.00$, $p=0.017$) is significant, while SWE-Bench and GAIA ($\rho=0.70$, $p=0.23$) are directionally consistent but underpowered at five roles.

Role	HotpotQA		WebArena		SWE-Code		GAIA		Decision
	$t_{1/2}$	R^2	$t_{1/2}$	R^2	$t_{1/2}$	R^2	$t_{1/2}$	R^2	
reasoning	4.20	.835	9.95	.471	4.68	.669	40.64	.608	<i>evict early</i>
action	1.16	.895	8.47	.483	3.76	.700	10.70	.243	<i>evict first</i>
instruction	4.71	.930	14.94	.972	11.55	.892	95.26	.306	<i>protect</i>
system	8.08	.982	17.34	.940	26.76	.947	32.90	.817	<i>pin</i>
tool_output	15.43	.888	34.36	.642	9.96	.924	∞	—	<i>protect</i>
ρ w/HotpotQA	1.00		1.00*		0.70		0.70		

- **Reasoning decays fastest** on search-then-answer workloads (HotpotQA, SWE-Code: $t_{1/2} \approx 5$). LRU’s recency rule keeps the freshest reasoning longest, the worst choice under the measured decay rates.
- **Tool outputs and system prompts persist:** near-constant attention across many turns, yet LRU evicts them first because they are the oldest.

LRU is role-blind and task-blind. LRU uses a single signal, block age and applies the same rule on every workload. It cannot distinguish a reasoning block from a tool output (role-blind), and it cannot adapt the eviction horizon to whether the workload is HotpotQA or GAIA (task-blind). The two failures compound as even when LRU happens to align with the decay rate of one role, it still mis-evicts whenever the workload shifts. RoleKV resolves both within a single functional form: the role tag breaks the role-blindness, and per-workload τ_r calibration (Eq. 3) breaks the task-blindness.

Cross-model invariance. The role ordering also holds across model families (pairwise spearman $\rho \geq 0.80$, App. C), and end-to-end gains transfer to Llama-3.1-70B without per-model retuning (§5).

3.2 Observation 2: The Cost of Wrong Eviction

To quantify the cost of these wrong evictions, we simulate a 70-turn Llama-3.1-8B-Instruct trace on HotpotQA at a 25% memory budget. LRU produces 445 cache misses, while an oracle that retains only the most-attended blocks produces 7. RoleKV closes $> 99\%$ of this gap (Fig. 4, detailed in §5.2). This gap is not a tuning artifact. It is structural. LRU’s recency heuristic systematically evicts the highest-reuse blocks because they happen to be the oldest.

Every wrong eviction is pure waste. In a preempt-discard engine, the evicted content must be recomputed from scratch leading to wasted GPU cycles. In a CPU-offload engine, the same content must be fetched back from CPU memory across the PCIe bus leading to wasted PCIe bandwidth. Both costs flow from the same wrong decision. The engine architecture determines only the currency in which the waste is paid. §5 quantifies this on vLLM (preempt-discard) and LMCash (CPU-offload).

4 RoleKV: System Design

The measurements of §3 establish two facts: KV blocks differ in attention persistence by semantic role, and per-role decay constants vary across workloads while preserving their relative ordering. RoleKV converts these facts into a deployable eviction policy in three stages, illustrated in Fig. 2. A role tagger labels each KV block with one of five semantic roles as the request enters the engine (§4.1). A lifetime calibration converts the decay constants of Table 1 into per-role thresholds that gate eviction eligibility (§4.2). A priority function ranks the eligible candidates by their expected future attention (§4.3). The resulting policy operates entirely at the system layer and preserves output fidelity by construction (§4.4).

4.1 Role Tagger

The role of a KV block is determined entirely by its position in the chat-template token stream. RoleKV exploits this with a deterministic single-pass tagger that runs on the host CPU before tokens enter the engine subprocess. The tagger maintains one register r ranging over the five roles of §2 (Table 1). As tokens arrive, the tagger updates r whenever the incoming sequence matches one of a small set of marker token-id sequences (Table 2). Each B -

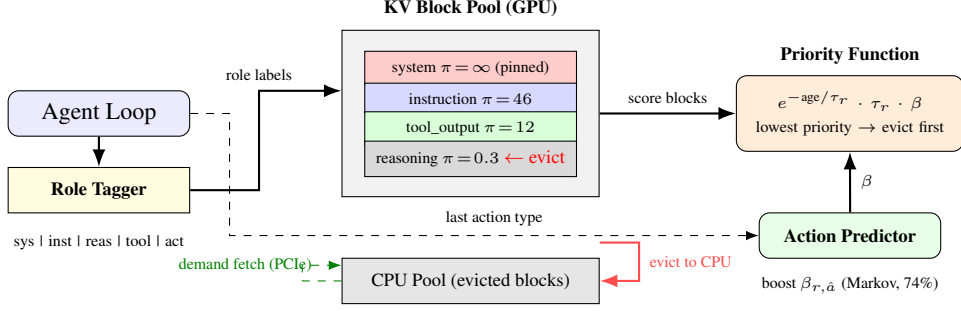


Figure 2: **RoleKV mechanism overview**. The Role Tagger labels KV blocks, the Priority Function scores each by role-decay $\times$ role-weight $\times$ action-predicted boost. Lowest-priority blocks (stale reasoning) are evicted first, highest-priority (system, instruction) are pinned.

Table 2: **Primary marker $\rightarrow$ role map (ReAct)**. Markers are token-id sequences, not strings (BPE-stable). Function-calling tags and the Llama-3 chat-header markers are in App. Table 18.

Marker	$\rightarrow$ Role
1st chat message	system
1st user message	instruction
Thought:	reasoning
Action:	action
Observation:/Result:	tool_output
no marker matched	instruction (fallback)

token KV block is then tagged with the value of r at the block’s first token, a *first-token-wins* rule that resolves block-boundary ambiguity in favor of the older role, a property we exploit in §4.3. With $B=16$ the procedure performs only integer comparisons on the host CPU and requires no model call, GPU work, or learned parameters.

The marker map is the only template-specific input. Each chat template - ReAct, OpenAI tool calls, Llama-3 chat headers, compiles to a fixed map from marker sequences to roles, and adding a new template adds rows to the map without changing policy code. Markers are stored as token-id sequences rather than strings, so the procedure is BPE-stable and survives tokenizer differences across model families.

The resulting state machine has five states and at most six transitions. Algorithm 1 formalizes the per-turn procedure, and Fig. 3 visualizes the transition structure.

Two block-boundary cases require explicit handling. When no marker fires within a block’s token span, the block is tagged *instruction*, the dominant role for unstructured user content. When a role transition straddles a block boundary, first-token-wins assigns the older role. In practice, *system* and *instruction* markers con-

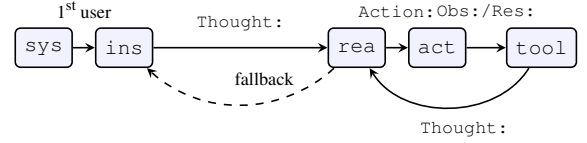


Figure 3: **Role-Tagger FSM (5 states)**. Solid: marker-triggered updates to r (Table 2), dashed: fallback. Block tags are emitted every B tokens (Alg. 1).

Algorithm 1 Role Tagger (per turn, deterministic, $O(N)$).

Require: tokens $t_1, \dots, t_N$, marker map $\mathcal{M}$, block size B
Ensure: one role tag per finalized KV block

```

1:  $r \leftarrow \text{system}$ 
2: for  $i = 1$  to  $N$  do
3:   if  $(t_i, t_{i+1}, \dots)$  matches marker  $m \in \mathcal{M}$  then
4:      $r \leftarrow \mathcal{M}(m)$ 
5:   end if
6:   if  $i \bmod B \equiv 1$  then
7:     emit tag  $\leftarrow r$  for new block
8:   end if
9: end for
10: Fallback: untagged block  $\leftarrow \text{instruction}$ 

```

centrate at the start of a turn, so straddled blocks are biased toward retention rather than premature eviction. Both rules are conservative under the priority function of §4.3. Any residual tagging error retains a block the policy would otherwise drop, never the reverse. Block-tag accuracy and runtime are reported in §6.5, with engine integration details in App. G.4.

4.2 Lifetime Calibration from Measured Decay

The decay constants τ_r measured in §3 give each role an attention half-life. We convert these into *time-to-live* (TTL) thresholds by asking how many turns must pass before a block’s expected attention falls below a fraction ϵ of its initial value. Under

the exponential decay model of Eq. 1,

$$\text{TTL}(r) = \lceil \tau_r \cdot \ln(1/\epsilon) \rceil, \quad \epsilon = 0.01, \quad (2)$$

so a block becomes *eligible* for eviction once its age exceeds $\text{TTL}(r)$. Substituting the HotpotQA values from Table 1 yields TTLs of (reasoning: 20, action: 6, instruction: 22, tool_output: 72) turns, with system pinned. The thresholds are not hand-tuned. Each is a direct ϵ -truncation of the measured decay rate, and a single probing pass on a new workload re-derives the entire TTL vector with no policy or model changes.

4.3 Priority Function

When the cache exceeds budget M , RoleKV evicts in two steps. First, all blocks within their TTL window are masked from eviction, with `system` masked unconditionally. Second, among the remaining eligible candidates, blocks are ranked by the priority score (with $r=\text{role}(b_i)$, $a_i=\text{age}(b_i)$, $\beta_i=\text{boost}(b_i)$)

$$\text{priority}(b_i) = \underbrace{e^{-a_i/\tau_r}}_{\text{decay}} \cdot \underbrace{\tau_r}_{\text{weight}} \cdot \underbrace{\beta_i}_{\text{boost}} \quad (3)$$

and evicted lowest-first. The first factor in Eq. 3 is the role-conditioned decay indicating how fresh the block is relative to its role’s natural lifetime. The second factor weights persistent roles higher. For example, on HotpotQA, $\tau_{\text{tool}}=15.43$ outranks $\tau_{\text{reason}}=4.20$ by $3.7\times$ at equal age. The third is a bounded multiplicative boost in $[0.3, 3.0]$ supplied by an action-prediction model conditioned on the most recent tool call (App. F). A Markov predictor on the last action achieves 74% accuracy at sub-millisecond cost. Where LRU compresses each block to its age alone, this score normalizes age against a role-specific lifetime and is therefore sensitive to which semantic role each block plays.

The combination produces a sharp separation between transient and persistent content. For example, a 5-turn-old reasoning block scores $e^{-5/4.20} \cdot 4.20 \approx 1.28$ and a 5-turn-old `tool_output` block scores $e^{-5/15.43} \cdot 15.43 \approx 11.16$, a $9\times$ priority gap that LRU cannot see. On workloads with stronger role separation (GAIA: $\tau_{\text{tool}} \rightarrow \infty$, $\tau_{\text{action}}=10.7$), the gap widens to two orders of magnitude.

The score follows from the decay model rather than from hand tuning. Under the exponential decay model of Eq. 1, the posterior expected total future attention to a block of role r at age a is

$a_{0,r} \cdot \tau_r \cdot e^{-a/\tau_r}$, where $a_{0,r}$ is the initial attention amplitude. Eq. 3 equals this expectation up to the boost term, so under the assumptions listed below it induces the Bayes-optimal *ordering* of eviction candidates.

Proposition 1 (Bayes-Optimal Ordering under the Stated Assumptions). *Assume (a) per-role attention follows the exponential decay model of Eq. 1, (b) blocks are equal-size, as under PagedAttention, (c) cache utility is the expected future attention retained, and (d) the cost of a future access is uniform across blocks. Let block b_i carry role r and current age a .*

(i) Posterior expectation. *The expected total future attention to b_i is*

$$\mathbb{E}[\text{future attention}(b_i) \mid r, a] = a_{0,r} \cdot \tau_r \cdot e^{-a/\tau_r},$$

where $a_{0,r}$ is the initial attention amplitude for role r (fitted in §3).

(ii) Optimal ordering. *Under (b)–(d), the policy maximizing expected cache-hit utility at budget M retains the top- M blocks ranked by this expectation.*

(iii) Connection to Eq. 3. *The priority score equals this expectation up to the $\text{boost}(b_i)$ correction, so Eq. 3 induces the Bayes-optimal ordering of candidates under (a)–(d). The claim concerns this ordering, not the absolute utility values.*

Proof in Appendix A.1.

What the policy consumes from this result is the *role ordering*, namely which roles outrank which at a given age. App. H.2 shows that this ordering is stable across wide variation of the fitted constants, and the exponential fit and the Markov boost act as refinements on top of it.

The two natural alternatives to this design lie on either side of it. Positional rules of the form “pin positions 0–30” use hand-picked thresholds with no connection to measured attention and do not adapt across workloads. Learned or RL eviction policies adapt but require per-workload gradient updates and add predictor-query overhead at inference time. RoleKV’s only parameters are the τ_r vector itself, fit once from attention probes (§3) and transferred zero-shot across model families.

Two refinements complete the policy: pinning `system` blocks prevents the LRU preemption cascade (§5.2), and a hybrid variant augments the decay prior with current-turn attention (App. F).

4.4 Lossless Guarantee

RoleKV operates entirely at the system layer, choosing where each KV block resides (GPU HBM, CPU memory, or rematerialized on demand) but never altering which tokens the model attends to. This is a fundamentally different *level of intervention* from model-level KV compression (Zhang et al., 2023; Li et al., 2024; Liu et al., 2024; Xiao et al., 2024), which approximates attention itself and accepts a quality–efficiency trade-off that the downstream application must absorb. Because RoleKV preserves the full KV state from the model’s perspective, its outputs are bit-identical to a no-eviction baseline and to any other budget-respecting system-layer policy (Theorem 1, SHA-256 verification in App. A).

Theorem 1 (Lossless eviction). *For any two budget-respecting policies π_{LRU} and π_{RoleKV} on the same model, prompt, and decoding parameters, $\text{OUTPUT}(\pi_{LRU}) = \text{OUTPUT}(\pi_{RoleKV})$.*

5 Experiments

5.1 Setup

We evaluate RoleKV on AMD InstinctTM MI300X GPUs (192 GB HBM, PCIe Gen4 $\times 16$) running ROCm 6.3, integrated into vLLM v0.11.2 and LMCACHE v0.4.5 as a drop-in plugin without source patches (App. D). We test Llama-3.1-8B-Instruct, Qwen3-32B, and Llama-3.1-70B-Instruct (Grattafiori et al., 2024; Yang et al., 2025a) on HotpotQA (Yang et al., 2018) ReAct traces, with 50/20/30 episodes per scale (mean 6 turns, 4-5K tokens per episode, full protocol and significance tests in App. B and App. E.7). §5.5–§5.7 extend the evaluation to GAIA and WebArena, to traces of up to 100 turns, and from one GPU to 8.

Measured vs. simulated. Unless stated otherwise, every reported number is an end-to-end measurement on a real serving engine. The only simulated results are the eviction-quality study of Fig. 4 and the hybrid-variant analysis of App. F, both labelled as trace-driven where they appear.

Block re-access and the cost chain. A block is *re-accessed* when a later request’s prefix overlaps context that was already computed. That re-access avoids recomputation if the block is GPU-resident (a GPU hit) or still held in CPU offload storage (a

¹AMD, AMD InstinctTM MI300X, and AMD ROCmTM are trademarks of Advanced Micro Devices, Inc. © 2026 Advanced Micro Devices, Inc.

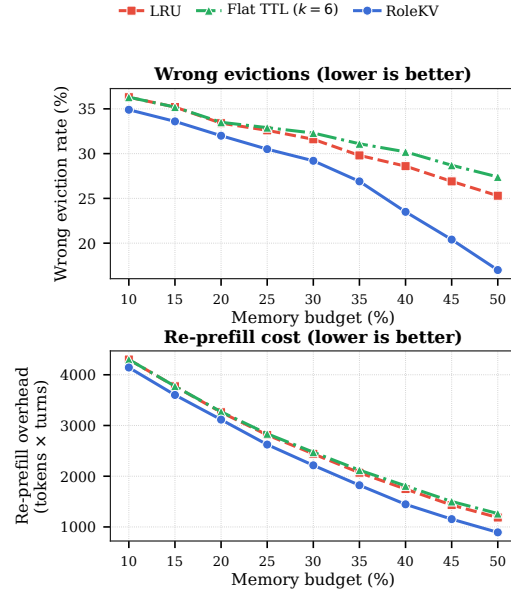


Figure 4: **Wrong-eviction rate and re-prefill cost vs. budget** (Llama-8B HotpotQA, $n=30$, trace-driven). LRU vs. Flat TTL ($k=6$) vs. RoleKV. At 50% budget RoleKV cuts re-prefill by 24.8% at equal eviction count.

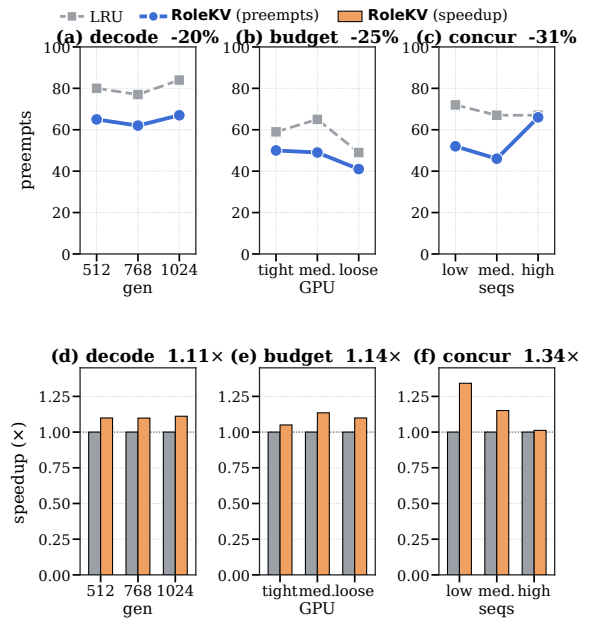


Figure 5: **vLLM 3-axis sweep, Llama-8B**. Top: preempts vs. decode/budget/concurrency, bottom: speedup over LRU. Peak -31% and $1.34\times$.

reload hit), otherwise the block is recomputed during prefill. This defines the chain we measure end to end: role-aware retention raises the residency of reused blocks, higher residency cuts recomputation and PCIe traffic, and that reduction lowers latency. GPU hit rate is therefore the intermediate signal that links the role signal (§3.1) to wall-clock gain.

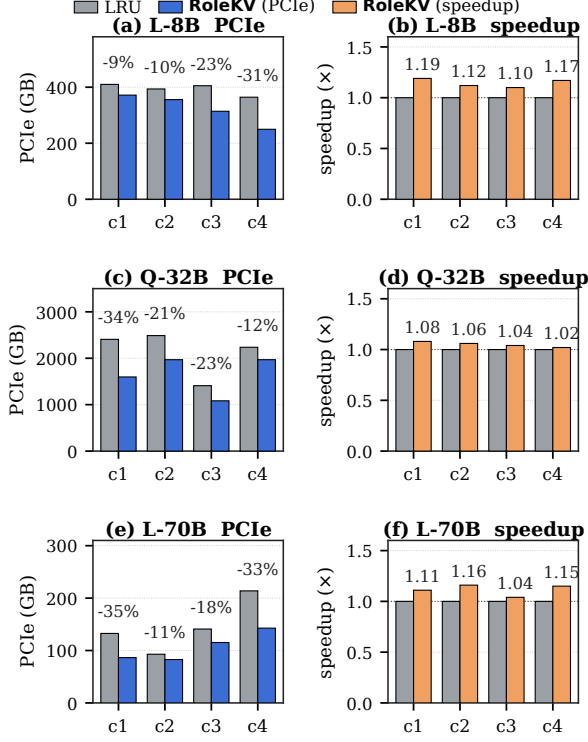


Figure 6: **LMCache, 3 models, 4 cells each.** Left: PCIe, mean $-18/-23/-26\%$ (8B/32B/70B). Right: speedup, mean $1.14/1.05/1.12\times$ (peak $1.19\times$). Bit-identical outputs.

5.2 vLLM: Preempt-Discard Path

Eviction quality. On a trace-driven Llama-8B simulator ($n=30$, Fig. 4), RoleKV cuts wrong evictions **32.8%** and re-prefill **24.8%** at 50% budget and equal eviction count (Hybrid variant in App. F).

LRU preemption cascade. LRU drops the *oldest* blocks, feeding the next preemption. RoleKV pins them, breaking the cycle (App. E.5). Preemptions fall **15-31%**, with speedup of up to **1.34 $\times$** (Fig. 5).

5.3 LMCache: CPU-Offload Path

Mechanism. The cost here is **PCIe traffic**, not recompute. LRU evicts system and instruction first, so each prefix-sharer pays a round-trip. With C sharers one retained block saves C round-trips (App. E.5).

Mean PCIe reduction grows with scale, **18.2/22.5/26.4%** on Llama-8B/Qwen-32B/Llama-70B and peaking at **34.8%**, worth mean speedups of **1.14 $\times$ /1.05 $\times$ /1.12 $\times$** and peak speedup of up to **1.19 $\times$** , (Fig. 6, significance in App. E.7). Outputs stay bit-identical to LRU (Thm. 1).

WebArena (short traces). Short traces create less memory pressure, so gains are smaller but consistent (Table 3). Under a tight budget RoleKV

Table 3: **WebArena, Llama-8B** (short traces). Gains track memory pressure and shrink as the budget loosens.

Budget	vLLM		LMCache	
	Prmp. $\downarrow$	Spd.	PCIe $\downarrow$	Spd.
Tight	64.3%	1.15 $\times$	15.2%	1.12 $\times$
Mid	50.0%	1.12 $\times$	12.8%	1.07 $\times$
Loose	25.0%	1.05 $\times$	11.9%	1.03 $\times$

Table 4: **Effect of retaining durable tool outputs** (GAIA, Llama-70B, vLLM). The workflow-structure policy (KVFlow) protects the fixed prompt. RoleKV additionally retains durable `tool_output` blocks, and its advantage widens with trace length.

Turns	GPU hit rate (%)			Speedup ($\times$)	
	LRU	KVFlow	RoleKV	KVFlow	RoleKV
55	18.7	42.0	45.2	1.22	1.27
75	21.7	45.4	50.0	1.29	1.36
100	24.7	46.1	52.1	1.25	1.41

cuts preemptions **64.3%** and PCIe traffic **15.2%**, worth **1.15 $\times$** and **1.12 $\times$** speedup, and lifts GPU hit rate from **18.4% to 32.3%**. Gains shrink as the budget loosens, matching the memory-pressure trend of App. H.3.

5.4 One Policy, Two Bottlenecks

The vLLM and LMCache gains arise from the same priority function (Eq. 3) at two transport layers, discard vs. offload. Hence policy and transport are orthogonal levers (side-by-side in App. E.6).

5.5 Generalization and Impact of Longer Trace Lengths

GAIA (long-horizon traces). We run GAIA (Mialon et al., 2024) traces of 22–100 turns on both models with two concurrent sequences ($\text{seqs}=2$, Fig. 7). RoleKV lifts GPU hit rate from **24.7% to 52.1%** at 100 turns, and the speedup *grows with trace length*, **1.24 $\times$** to **1.41 $\times$** on Llama-70B ($1.11\times$ to $1.21\times$ on 8B), because longer trajectories deepen intra-agent reuse of durable `tool_output`. On Llama-8B, preemptions fall **42.9%** and PCIe traffic **63.8%** at 22 turns ($18.2\%/19.1\%$ at 100, App. E.3). The residency advantage is proportionally largest early, $6.7\times$ over LRU at 22 turns against $2.1\times$ at 100, yet the speedup still grows, because each retained point protects more durable context as the trace lengthens. Llama-70B converts a given residency into more speedup than Llama-8B (**1.41 $\times$** vs $1.21\times$) because its per-block KV is larger.

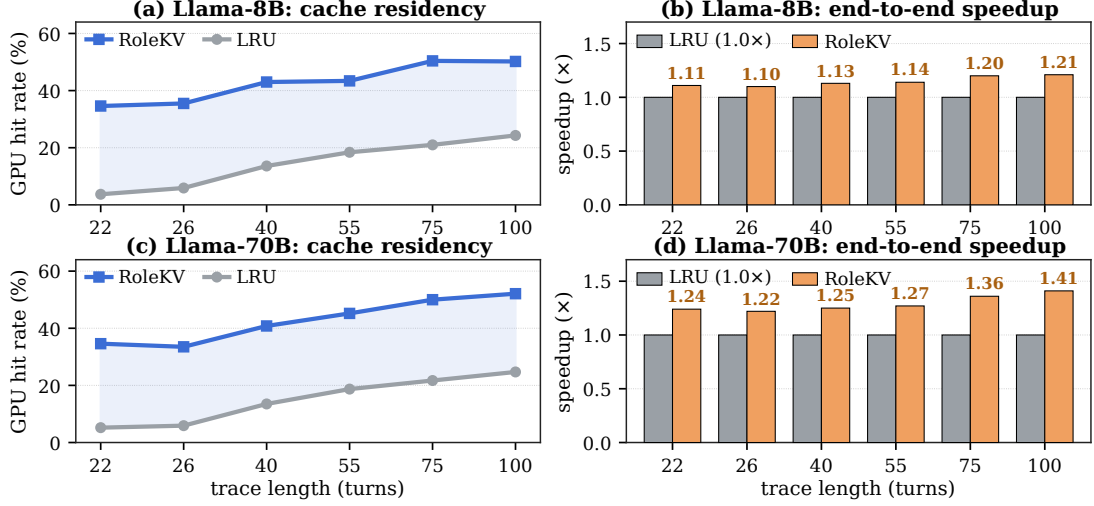


Figure 7: **GAIA generalization and trace-length scaling** (vLLM, seqs=2). Top row Llama-8B, bottom row Llama-70B, with shared y-limits so the two models can be compared directly. RoleKV raises GPU KV-cache hit rate at every trace length (a, c), and the resulting speedup over LRU grows with trace length (b, d), reaching 1.41 $\times$ on Llama-70B at 100 turns.

Table 5: **Blocks evicted by role** (GAIA, Llama-70B, vLLM, 100-turn trace). Both policies pin the fixed prompt. RoleKV additionally protects durable `tool_output`.

Policy	sys	instr	tool	reason	act	total
LRU	all	all	all	all	all	356,697
KVFlow	17	254	81,558	205,763	760	288,352
RoleKV	4	84	31,791	196,859	748	229,486

5.6 Impact of Tools over Longer Traces

Tool outputs are re-read many turns after they are written, so pinning the prompt alone is insufficient. Against a workflow-structure policy (Pan et al., 2025), RoleKV lifts residency **46.1 $\rightarrow$ 52.1%** and speedup **1.25 $\times \rightarrow$ 1.41 $\times$** (Table 4). That advantage widens with the trace length, from 3.2 pp in GPU hit rate and 0.05 $\times$ speedup at 55 turns to **6.0 pp** and **0.16 $\times$** at 100 turns. Table 5 shows why. Both drive system and instruction evictions to near zero and discard a similar volume of reasoning (196.9K vs 205.8K), so RoleKV is not simply evicting less. `tool_output` alone accounts for **85%** of the 58.9K-block gap between them. The designs are complementary: *which agent’s* cache to keep versus *which blocks* within it.

5.7 Multi-GPU Tensor-Parallel Serving

Every shard retains the same durable context, so the priority applies shard-locally. On Llama-70B gains hold at TP=1/2/4/8, **1.41 $\times$ to 1.23 $\times$** at 100 turns and 1.24 $\times$ to 1.08 $\times$ at 22, largest where per-GPU pressure peaks (Fig. 8, App. E.4). Residency stays flat across degrees, **51.6–52.0%** against 24.4–24.9% for LRU, so the effect survives sharding

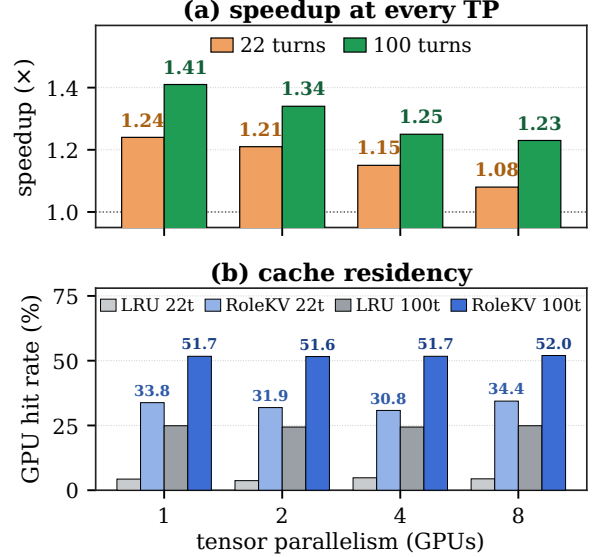


Figure 8: **Multi-GPU tensor-parallel scaling** (GAIA, Llama-70B, vLLM). Gains hold at every TP degree and are largest where per-GPU memory pressure is highest. Panel (b) reports both trace lengths: the residency gap over LRU is near 8 $\times$ at 22 turns and near 2 $\times$ at 100 turns.

intact. Speedup declines only because sharding relieves the per-GPU memory pressure, leaving fewer evictions for RoleKV to prevent.

6 Ablations and Robustness

6.1 Trace Length Sensitivity

Preemption reduction (Llama-8B) grows with trace length: 0–3% at ≤ 3 turns, 15–31% at 4–10, 28–41% beyond 20. GAIA sits entirely above 20 turns, where gains keep growing (§5.5). Across 22 to 100 turns the Llama-70B speedup rises **1.24 $\times$**

to $1.41\times$ and residency 34.6% to **52.1%**, against LRU’s 5.2% to 24.7%. The trend has not saturated at the longest trace we ran.

6.2 Decay Structure at 70B

A 70B probe (App. C.1) correlates with Llama-8B at $\rho = 0.91$, so τ_r transfers unchanged. All GAIA results use this model.

6.3 Cross-Workload Decay Generalization

The four workloads span **short** (HotpotQA, WebArena) and **long** (GAIA, SWE-Bench) regimes. The transient-below-durable rank holds throughout ($\rho \geq 0.70$, Table 1) while rates shift $20\times$. Only the constants move, and one probing pass refits them (App. C.2).

6.4 Reuse Regimes: Inter- and Intra-Agent

Short traces (HotpotQA, WebArena) gain from *inter-agent* prefix reuse, one block serving many requests. Long traces (GAIA) gain from *intra-agent* reuse, so speedup grows $1.24\times$ to $1.41\times$ at seqs=2 from 22 to 100 turns. RoleKV serves both.

6.5 Role-Tagger Format Robustness

On HotpotQA-Llama-8B the tagger reaches 99.58% block accuracy (Wilson 95% CI [99.42, 99.69], $n=8,743$). The residual 3.1% confusion (action $\rightarrow$ tool_output) is conservative under Eq. 3, delaying eviction, never accelerating it (confusion matrix in App. G.3).

7 Related Work

Model-level KV compression (Zhang et al., 2023; Li et al., 2024; Tang et al., 2024; Liu et al., 2023; Adnan et al., 2024; Cai et al., 2024; Ge et al., 2024; Singhania et al., 2024; Ye et al., 2024) approximates away unimportant tokens within a single inference call, forcing the downstream application to absorb a quality-efficiency trade-off. Position-based variants (StreamingLLM (Xiao et al., 2024), LM-Infinite (Han et al., 2024)) keep attention sinks alongside a sliding window (a coarse proxy for the per-role structure we measure), and quantization (KIVI (Liu et al., 2024)) reduces per-block memory rather than choosing what to retain, leaving it orthogonal and composable with RoleKV. RoleKV operates outside the single-call horizon, drawing on role information visible only across turns.

Predictive eviction (Jie et al., 2025; Yang et al., 2025b; Ahn et al., 2026; Dong et al., 2026; Galim et al., 2026) trains auxiliary models to score future

importance within a single forward pass, sharing the model-level family’s single-call horizon and requiring per-workload tuning. RoleKV needs no training and conditions on a cross-turn semantic signal.

Agent-aware serving systems are the closest line of work: KVFlow (Pan et al., 2025) models agent step graphs, Continuum (Li et al., 2025) applies a flat TTL, Tokencake (Bian et al., 2025) offloads during tool stalls, MemGPT (Packer et al., 2023) manages long-term memory at the prompt level, SideQuest (Kariyappa and Suh, 2026) compresses with an auxiliary LLM, LRAgent (Jeon et al., 2026) combines prefix reuse with reactive eviction, and PreAct (Fu et al., 2025) pre-populates the cache by predicting future tool calls. Their retention signals are age- or step-based. RoleKV conditions on the measured per-role decay dynamic of §3.1 and is lossless by construction (§4.4).

Serving infrastructure- DistServe (Zhong et al., 2024), Splitwise (Patel et al., 2024), Preble (Srivatsa et al., 2024), Sarathi (Agrawal et al., 2024), and PromptCache (Gim et al., 2024) optimizes scheduling, disaggregation, and schema-level reuse. These contributions are orthogonal to eviction-policy choice and stack with RoleKV.

Production stacks default to LRU. vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), Orca (Yu et al., 2022), and LMCache (Liu et al., 2025) all delegate KV-cache eviction to a recency rule. RoleKV integrates with vLLM and LMCache as a plugin without source patches (§5), showing that role-aware retention is deployment-ready, not research-only.

8 Conclusion

We identify an *inverted age-importance* pattern in multi-turn LLM agents. Per-role attention half-lives differ by more than an order of magnitude, while their ordering holds across models and tasks, causing LRU to evict the wrong blocks. RoleKV, a lossless decay-calibrated policy, cuts preemptions by up to 64.3% (vLLM), PCIe traffic by up to 63.8% (LMCache), and latency by up to $1.41\times$ while remaining applicable across model scales. Gains hold across short- and long-horizon agent workloads, grow on longer agent traces, and persist across multi-GPU setups. This recasts KV eviction from a *recency* problem into a *role-aware retention* problem, with one priority function serving both preempt-discard and offload architectures.

Limitations

Memory-pressure-contingent gains. Gains concentrate in the 20-60% GPU-budget range. Above $\sim 80\%$ the cache fits and no eviction policy matters. Per-eviction overhead is < 0.1 ms.

Non-shared prompts for short traces. For short traces, a substantial portion of RoleKV’s benefit comes from inter-agent prefix reuse across agents sharing the same system prompt. We do not evaluate short-trace workloads without shared prefixes, as all short-trace experiments use ≥ 16 concurrent sequences. Long traces are less sensitive to this limitation because their gains arise mainly from intra-agent reuse of durable context within a single trajectory, measured at only 2 concurrent sequences (seqs=2) (§5.5).

Role taxonomy and format. The tagger needs a marker vocabulary per template (App. G). Unrecognized formats degrade it to uniform TTL, which beats LRU but loses the role-decay gain. Roles beyond our five are untested.

Workload-dependent decay calibration. Per-role τ_r shifts up to $20\times$ across domains (Table 1), so exact ordering needs a probing pass (~ 50 ep., ~ 1 GPU-h). The transferred HotpotQA prior beats LRU but is not optimal everywhere.

Acknowledgments

We gratefully acknowledge the support of AMD Research and Advanced Development (RAD) and the collaborative environment that facilitated the experimental evaluation and interpretation of the results. Finally, we thank anonymous reviewers for their detailed feedback and suggestions, which strengthened the rigor and presentation of this paper.

Ethics Statement

RoleKV is a lossless inference-time optimization that does not modify model behavior, training data, or output. Evaluation uses the publicly available HotpotQA, WebArena, and GAIA benchmarks under their respective licenses and terms of use. No human subjects were involved and no new data was collected. No new model weights are released. RoleKV is a software-only contribution applicable to any pre-trained LLM serving stack.

References

- Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. Keyformer: KV cache reduction through key tokens selection for efficient generative inference. In *Conference on Machine Learning and Systems*.
- Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134.
- Jinwoo Ahn, Ingyu Seong, Akhil Kedia, Junhan Kim, Hyemi Jang, Kangwook Lee, and Yongkweon Jeon. 2026. LookaheadKV: Fast and accurate KV cache eviction by glimpsing into the future without generation. In *International Conference on Learning Representations*.
- Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.
- Zhuohang Bian, Feiyang Wu, Zhuoran Li, Teng Ma, and Youwei Zhuo. 2025. TokenCake: A KV-cache-centric serving framework for LLM-based multi-agent applications. *arXiv preprint arXiv:2510.18586*.
- Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.
- Zican Dong, Peiyu Liu, Junyi Li, Zhipeng Chen, Han Peng, Shuo Wang, and Wayne Xin Zhao. 2026. ForesightKV: Optimizing KV cache eviction for reasoning models by learning long-term contribution. *arXiv preprint arXiv:2602.03203*.
- Dayuan Fu, Jianzhao Huang, Siyuan Lu, Guanting Dong, Yejie Wang, Keqing He, and Weiran Xu. 2025. PreAct: Prediction enhances agent’s planning ability. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 1–16.
- Kevin Galim, Ethan Ewer, Wonjun Kang, Minjae Lee, Hyung Il Koo, and Kangwook Lee. 2026. Draft-based approximate inference for LLMs. In *International Conference on Learning Representations*.
- Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. 2024. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*.

- In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt Cache: Modular attention reuse for low-latency inference. In *Conference on Machine Learning and Systems*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. 2024. LM-Infinite: Zero-shot extreme length generalization for large language models. In *Conference of the North American Chapter of the Association for Computational Linguistics*.
- Hyesung Jeon, Hyeongju Ha, and Jae-Joon Kim. 2026. LRAgent: Efficient KV cache sharing for multi-LoRA LLM agents. *arXiv preprint arXiv:2602.01053*.
- Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. 2025. SpeCache: Speculative key-value caching for efficient generation of LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 27917–27928.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-Bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*.
- Sanjay Kariyappa and G. Edward Suh. 2026. Side-Quest: Model-driven KV cache management for long-horizon agentic reasoning. *arXiv preprint arXiv:2602.22603*.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.
- Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Alvin Cheung, Joseph E. Gonzalez, and Ion Stoica. 2025. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*.
- Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*.
- Yuhan Liu, Jiayi Yao, Yihua Cheng, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. 2025. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyriklidis, and Anshumali Shrivastava. 2023. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhao Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, pages 32332–32344.
- Grégoire Mialon, Clémentine Fourier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*.
- Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*.
- Zaifeng Pan, Ajjkumar Patel, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. 2025. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. In *Advances in Neural Information Processing Systems*.
- Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative LLM inference using phase splitting. In *51st ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132.

- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.
- Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. 2024. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*.
- Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiyang Zhang. 2024. Preble: Efficient distributed prompt scheduling for LLM serving. *arXiv preprint arXiv:2407.00023*.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, pages 47901–47911.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Qingyue Yang, Jie Wang, Xing Li, Zhihai Wang, Chen Chen, Lei Chen, Xianzhi Yu, Wulong Liu, Jianye Hao, Mingxuan Yuan, and Bin Li. 2025b. Attention-Predictor: Temporal patterns matter for KV cache compression. In *Advances in Neural Information Processing Systems*.
- Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.
- Lu Ye, Ze Tao, Yong Huang, and Yang Li. 2024. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In *Annual Meeting of the Association for Computational Linguistics*.
- Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*.
- 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. 2024. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*.
- Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210.
- Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue

Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*.

Appendix Overview

App. A Losslessness (Thm. 1), the ordering result of Prop. 1 under its assumptions, SHA-256 protocol

App. B Measurement protocol: pairing, variance, metric sources, episode counts

App. C Per-model decay tables with fit quality, probing methodology, the 70B probe, and cross-workload analysis

App. D Engine integration: vLLM connector, LM-Cache policy registration, configuration matrices, environment

App. E Result tables: GAIA and multi-GPU sweeps (Tables 11, 13), HotpotQA cells, significance

App. F Priority function: action-predictor comparison and the hybrid variant that adds current-turn attention

App. G Role Tagger: marker map (Table 18), implementation, ground truth, confusion matrix, engine hooks

App. H Additional ablations: per-role contribution, decay-calibration robustness, budget and concurrency sensitivity

A Formal Proof of the Lossless Property

We formally prove that RoleKV is a strictly lossless optimization (Theorem 1) and that the priority function induces the Bayes-optimal ordering under the assumptions of Proposition 1.

Setup. Let $\mathcal{B} = \{b_1, \dots, b_n\}$ be the set of KV cache blocks, $\mathcal{T} = \{\text{GPU}, \text{CPU}\}$ the tier labels, and $\pi : \mathcal{B} \rightarrow \mathcal{T}$ an eviction policy subject to $|\{b : \pi(b) = \text{GPU}\}| \leq M$.

Lemma 1 (Tier transparency). *For any block b_i , the value of b_i in GPU HBM is bit-identical to the value of b_i in CPU DRAM, modulo the deterministic copy operation `cudaMemcpyAsync`.*

Proof. DMA (`cudaMemcpyAsync` / `hipMemcpyAsync`) is a byte-for-byte copy. No quantization, lossy compression, or rounding is applied along the GPU $\leftrightarrow$ CPU path under standard offloading configurations (Kwon et al., 2023). $\square$

Lemma 2 (Demand-fetch completeness). *For any policy π satisfying the budget constraint, when the model attempts to read block b_i during attention, b_i is brought back to GPU HBM (if not already resident) before being consumed.*

Proof. This is the contract of both production engines evaluated: vLLM’s `OffloadingConnector` and LMCache’s `LMCacheConnectorV1` both implement synchronous demand-driven fetch (Liu et al., 2025). The engine stalls until the block is resident. $\square$

Theorem 2 (Lossless property of RoleKV (Theorem 1)). *For any two budget-respecting policies π_{LRU} and π_{RoleKV} :*

$$\text{ATTN}(\mathcal{B}, \pi_{\text{LRU}}) = \text{ATTN}(\mathcal{B}, \pi_{\text{RoleKV}})$$

Consequently, the next-token logits and sampled output are bit-identical.

Proof. By Lemma 1, every block’s value is preserved across tiers. By Lemma 2, the model accesses the same set of blocks $\mathcal{B}$ during attention regardless of π , with each b_i resident at the moment of access. Since attention is a deterministic function of $(\mathcal{B}, Q)$ where Q depends only on the model state (itself a deterministic function of $\mathcal{B}$ and input), the output is a function purely of $\mathcal{B}$, not of π . $\square$

What π changes. Only the *cost*:

$$\text{COST}(\pi) = \sum_{b_i: \pi(b_i) = \text{CPU}} \mathbb{1}[b_i \text{ accessed}] \cdot c_{\text{DMA}}$$

RoleKV minimizes COST by keeping high-access-probability blocks (system, instruction) on GPU. Output is identical. Only the data-movement bill differs.

Qualitative verification. Table 6 shows a representative HotpotQA episode from our evaluation where the same input prompt produces word-for-word identical answers under both LMCache (default LRU eviction) and LMCache+RoleKV. Across all 12 evaluation cells in Table 10 (cold-pass evaluation), SHA-256 hashes of generated token sequences were verified to be 100% identical between the two policies.

A.1 Proof of Proposition 1

Claim. Under assumptions (a)–(d) of Proposition 1, the expected total future attention to a block of role r at current age a is $a_{0,r} \cdot \tau_r \cdot e^{-a/\tau_r}$, and retaining blocks in decreasing order of this quantity maximizes expected cache utility under a fixed budget M . The result fixes the eviction *ordering* and does not claim optimality outside these assumptions.

Proof. The expected future attention at age a is obtained by integrating the decay model forward:

$$\begin{aligned} \mathbb{E}[\text{future attention}(b_i)] &= \int_a^\infty a_{0,r} \cdot e^{-k/\tau_r} dk \\ &= a_{0,r} \cdot \tau_r \cdot e^{-a/\tau_r}. \end{aligned}$$

Given a budget of M equally-sized blocks (PageAttention’s fixed-size paging ensures this), the problem of maximizing total expected future attention subject to retaining at most M blocks is a linear selection problem. Since all blocks have equal size, the optimal solution retains the M blocks with the highest values of $a_{0,r} \cdot \tau_r \cdot e^{-a/\tau_r}$ —i.e., blocks are sorted in decreasing order of this quantity and the top M are kept. $\square$

B Detailed Experimental Setup

Pairing. Every evaluation configuration is a paired LRU/RoleKV comparison sharing seed (42), prompts, GPU budget (`num_gpu_blocks_override`), and decode length—only the eviction policy varies. We perform three runs for each experiment and report the mean result.

Variance and replication. Episodes run closed-loop under greedy decoding (`temperature=0`, `enforce_eager=True`) with deterministic scheduling, so within-cell variance is dominated

Table 6: **Lossless guarantee, qualitative example.** LMCache (LRU) and LMCache+RoleKV produce bit-identical answers, SHA-256 match across all 12 cells (cold pass).

Role	Content
SYSTEM	<i>Researcher; answer multi-hop questions step-by-step with search.</i>
USER	Which award did the director of the 2019 Palme d’Or winner receive at the 2020 Oscars?
ASST.	<i>Thought:</i> find the 2019 Palme d’Or winner, then its director. <i>Action:</i> search(“2019 Palme d’Or”)
TOOL	<i>Parasite</i> , directed by Bong Joon-ho.
ASST.	<i>Thought:</i> look up the 2020 Oscar result. <i>Action:</i> search(“Bong Joon-ho 2020 Oscars”)
TOOL	Won Best Director and Best Picture, 92nd Academy Awards.
Both outputs: <i>Best Director at the 92nd Academy Awards.</i> SHA-256 a1d37c5b... matches.	

by scheduler interleavings rather than sampling noise. We report mean results across three runs for each evaluation configuration and treat variation across the 9 vLLM configurations and 12 LMCache configurations as an additional robustness signal. Significance of the paired effect is reported in App. E.7.

Metric sources. Preempt counts: vLLM’s `num_preemptions_total` Prometheus counter. PCIe traffic: sum of LMCache’s per-request `Stored` and `hit_tokens` log entries. TTFT / end-to-end wall-time: client-side, from `add_request` to first decoded token / last response. We report cell means. P95 tracks the mean within $\sim 5\%$ under deterministic scheduling.

Episode counts. 50/20/30 episodes for Llama-8B/Qwen-32B/Llama-70B, set by per-scale wall-clock budget (Llama-70B takes $\sim 8\times$ the wall-time of Llama-8B per episode on the same GPU).

C Per-Model Decay Tables, Probing Methodology, and 70B Probe Results

Table 7 expands main-paper Table 1 with per-model half-lives and fit quality.

Probing methodology. The exponential model $\bar{a}_r(k) = a_{0,r} e^{-k/\tau_r}$ (§3) is fitted by weighted least-squares, with each data point weighted by token count to avoid bias from sparse roles at high ages.

Findings. `tool_output` outlives even `system` on all three models, since factual grounding is revisited across reasoning steps. Chain-of-thought reasoning (Wei et al., 2022) decays fastest among non-action roles (1.3–4.2 turns on HotpotQA), though it persists far longer on long-horizon GAIA ($t_{1/2}=40.6$, App. C.2). The exponential model fits at $R^2 > 0.83$ on 12 of 15 (role, model) cells. Absolute τ differs $2\text{--}3\times$ across families while the ordering is preserved (mean pairwise Spearman $\rho = 0.87$), which is what lets one

role-based policy serve all three without per-model recalibration.

C.1 Llama-3.1-70B Probe Results

A targeted probe on Llama-3.1-70B-Instruct (10 episodes, 5 turns each, methodology of §3) confirms τ_r transfers without recalibration.

C.2 Cross-Workload Decay: Full Analysis

Role-rank deviations. Two columns show fine-rank swaps, both consistent with task structure and both leaving the exploited gap intact. On SWE-Coding `instruction` (11.55) outlives `tool_output` (9.96) because file reads churn faster than QA search results, while the reasoning-to-system separation widens to $5.7\times$. On GAIA `reasoning` (40.6) approaches `system` (32.9) because inference chains persist across longer traces, yet `action` stays shortest-lived (10.7) and `tool_output` most persistent, so pinning durable roles and evicting `action` early is unaffected.

Decay rates follow task access patterns. The rates track how each task re-reads its context: HotpotQA reads search results once ($\tau_{\text{tool}}=15.4$), WebArena revisits pages while replanning (34.4), SWE-Coding overwrites files on each edit (9.96), and GAIA re-reads retrieved content at every chained step ($\tau \rightarrow \infty$). The *role* of a block is unchanged by this. Only the constant attached to it moves, which is exactly the quantity a one-shot probing pass refits. Hence, the same policy form transfers across tasks without redesign.

Calibration cost and transfer. A new workload needs one probing pass (~ 50 episodes, ~ 1 GPU-hour on Llama-8B), with no training. In practice, RoleKV remains effective when the exact decay constants are imperfect, since the policy relies primarily on the separation between transient and

Table 7: **Per-model attention-decay half-lives ($t_{1/2}$, turns) with exponential-fit R^2 on HotpotQA ReAct traces.** Ordering {reasoning, action} $\ll$ {instruction} $\ll$ {system, tool_output} is invariant across families. $N=299/300/199$ probes per column. Compact form: Table 1.

Role	Qwen3-1.7B		Llama-3.1-8B		Qwen3-8B		Eviction priority
	$t_{1/2}$	R^2	$t_{1/2}$	R^2	$t_{1/2}$	R^2	
reasoning	1.29	.881	4.20	.835	2.24	.757	<i>Evict first</i> (consumed once)
action	2.47	.816	1.16	.895	0.92	.952	<i>Evict early</i> (rarely revisited)
instruction	6.97	.775	4.71	.930	6.19	.684	<i>Protect</i> (task context)
system	11.27	.884	8.08	.982	5.43	.632	<i>Never evict</i> (behavioral anchor)
tool_output	27.82	.854	15.43	.888	12.91	.738	<i>Protect</i> (external data, near-constant)

Table 8: **Decay half-lives, Llama-70B vs. Llama-8B.** Role ordering is preserved ($\rho = 0.91$, $p < 0.05$), no per-model recalibration needed for the 70B LMCache run.

Role	Llama-3.1-8B		Llama-3.1-70B	
	$t_{1/2}$	R^2	$t_{1/2}$	R^2
reasoning	4.20	.835	3.1	.812
action	1.16	.895	0.9	.871
instruction	4.71	.930	5.8	.904
system	8.08	.982	$\gg 50$	.961
tool_output	15.43	.888	22.4	.851

durable roles. RoleKV therefore transfers even when calibration data is unavailable.

D Engine Integration Details

D.1 Hardware and Software Environment

All experiments run on the GPU platform of §5.1 (ROCm 6.3, Ubuntu 24.04, Python 3.12), using one GPU unless stated otherwise and 2/4/8 GPUs for the tensor-parallel study of §5.7. Host CPU: Intel Xeon Platinum 8480+ (2 sockets $\times$ 56 cores). Interconnect: PCIe Gen4 $\times$ 16 (32 GB/s bidirectional). Fresh Docker container per evaluation cell, no warmup passes.

D.2 vLLM OffloadingConnector Plugin

RoleKV integrates into vLLM v0.11.2 via the KVTransferConfig plugin API with no source patches required. The integration registers a custom OffloadingConnector that intercepts the eviction-ordering callback. Role tags produced by the Role Tagger (App. G) are written to a side-channel RoleMetadata map keyed by block_id. On every eviction event the connector (i) evaluates Eq. 3 over all blocks in the GPU pool (< 0.05 ms), (ii) filters out system-tagged blocks (priority = ∞), and (iii) returns the remaining blocks sorted ascending. The scheduler’s _preempt_by_priority hook evicts lowest-priority first.

D.3 LMCache LMCacheConnectorV1 Policy Registration

RoleKV registers with LMCache v0.4.5 via the POLICY_MAPPING dictionary in lmcache/cache_engine.py:

```
POLICY_MAPPING["ROLEKV"] = \
    RoleKVCachePolicy
```

The RoleKVCachePolicy class overrides evict() to return blocks in ascending priority order. LMCache config: cache_policy: "ROLEKV", CPU offload pool 8–16 GB, chunk size 256 tokens. Both engines expose the same sorted-block-list callback contract. RoleKV only consumes (role tag, age), making the policy class portable across engines.

D.4 Configuration Matrices

Each vLLM axis varies one knob with the other two pinned at the cascade-firing point (9 cells per model, values in Table 9). The 4 LMCache cells per model are those where LRU PCIe traffic exceeds $5\times$ the no-eviction baseline, providing measurable PCIe pressure while avoiding configurations where both policies exhibit similar behavior (Table 10). Per-block KV is 2/4/5 MB for Llama-8B/Qwen-32B/Llama-70B (32/64/80 layers, GQA (Ainslie et al., 2023), bf16). Bit-identity is verified by SHA-256 (App. A).

E Detailed Results: vLLM and LMCache

Raw numbers behind every figure and headline claim in §5.

E.1 vLLM: Full 3-Axis Preemption Sweep

RoleKV reduces preemptions by up to 31% across the Llama-8B evaluation sweep (Table 9), indicating that the effect is structural rather than configuration-specific.

Table 9: **Llama-8B vLLM 3-axis sweep: preempts and wall-time speedup** (Fig. 5, 50 ep.). Each axis varies one knob with the other two pinned at the cascade-firing point. ↓ columns are reductions against LRU, so larger is better. Preempt reduction is 15-31%, peak speedup **1.34**.

Varying axis	GPU / seqs / gen	Preempt count		Preempt ↓ (%)	Speedup (×)
		LRU	RoleKV		
Decode length	256 / 32 / 512	80	65	18.8	1.10
	256 / 32 / 768	77	62	19.5	1.10
	256 / 32 / 1024	84	67	20.2	1.11
GPU budget	160 / 16 / 512	59	50	15.3	1.05
	192 / 16 / 512	65	49	24.6	1.13
	320 / 16 / 512	49	41	16.3	1.10
Concurrency	320 / 32 / 1024	72	52	27.8	1.34
	320 / 64 / 1024	67	46	31.3	1.06
	320 / 96 / 1024	67	66	1.5	1.02

E.2 LMCache: PCIe Traffic and Inference Speedup

Table 10 provides the per-configuration breakdown across all 12 cascade-firing cells (4 per model, 3 models).

Two effects drive the numbers. Reduction grows with model scale (18.2/22.5/26.4% mean) because larger per-block KV means each retained block saves more bytes. Prefix sharing multiplies it, since one retained shared block eliminates C round-trips for C concurrent requests, inverting LRU’s worst case.

Absolute magnitudes are not comparable across models. They depend on per-block bytes, the concurrency each model’s weights permit (Llama-70B is capped at seqs $\in \{16, 32\}$ by its 140 GB footprint), and per-step compute time. Figure 6 therefore gives each panel its own axis. The comparable metric is the **paired $\Delta\%$** for each configuration, exact by construction, which increases with model size (12/12 cells positive, Wilcoxon $p < 0.001$).

E.3 GAIA: Trace-Length Sweep

Table 11 gives the per-trace numbers behind Fig. 7. Both models run identical traces for only 2 concurrent sequences (seqs=2), so the comparison isolates trace length from concurrency. Residency rises for both policies as trace length increases, but the RoleKV-to-LRU gap widens, and the speedup grows monotonically on Llama-70B from 1.24 at 22 turns to 1.41 at 100. Table 12 reports the corresponding preemption and PCIe reductions on Llama-8B. The largest relative reductions occur at shorter GAIA traces, while end-to-end speedup continues to grow with trace length as retained

durable context accumulates over longer trajectories.

E.4 Multi-GPU Tensor-Parallel Scaling

Table 13 gives the per-degree numbers behind Fig. 8. Residency and eviction are decided per shard, and every shard must retain the same durable context, so the role priority applies shard-locally without modification. Residency is therefore flat across degrees while speedup declines, because lower per-GPU memory pressure leaves fewer evictions for RoleKV to prevent.

E.5 Preemption Cascade and Offload Mechanism

The LRU preemption cascade (vLLM). Under memory pressure the dynamics self-reinforce. Long decode pushes new KV into GPU, forcing eviction. LRU evicts the *oldest* blocks (system/instruction), breaking the GPU prefix cache for the shared system prompt. Subsequent prefix-sharing requests must re-prefill the prompt, consuming *more* GPU memory and triggering the next preemption. RoleKV pins system blocks (~ 30 deflections per pass), so the prefix cache stays warm, new requests hit it, and the cycle stops self-reinforcing.

Why the same policy pays off on offload (LM-Cache). On LMCache the dominant cost is CPU $\leftrightarrow$ GPU PCIe traffic rather than recomputation. LRU evicts system and instruction blocks first, so every prefix-sharing request pays a round-trip to fetch them back. RoleKV evicts transient reasoning blocks instead. With C concurrent requests sharing a system prompt, one correctly retained block eliminates C round-trips, which is why the reduction

Table 10: **LMCache per-cell PCIe traffic and inference speedup** (4 cells per model, 8 GB CPU offload). ↓ columns are reductions against LRU, so larger is better. Mean PCIe reduction 18.2 / 22.5 / 26.4% (Llama-8B / Qwen-32B / Llama-70B), peak **34.8%**. Mean speedup 1.14 / 1.05 / 1.12.

Model	GPU / seqs / gen	PCIe (GB)		PCIe ↓ (%)	Mean ↓ (%)	Speedup (×)	Mean spd. (×)
		LRU	RoleKV				
Llama-8B	256 / 32 / 1024	410.0	371.7	9.3	18.2	1.19	1.14
	256 / 32 / 512	393.7	355.7	9.6		1.12	
	256 / 32 / 768	405.3	314.1	22.5		1.10	
	320 / 64 / 1024	364.3	249.8	31.4		1.17	
Qwen-32B	256 / 48 / 1024	2 407	1 597	33.6	22.5	1.08	1.05
	192 / 32 / 1024	2 488	1 969	20.9		1.06	
	320 / 32 / 1024	1 409	1 083	23.2		1.04	
	192 / 48 / 1024	2 237	1 969	12.0		1.02	
Llama-70B	256 / 16 / 1024	132.6	86.5	34.8	26.4	1.11	1.12
	256 / 16 / 768	92.9	82.8	10.9		1.16	
	320 / 16 / 1024	141.0	115.3	18.2		1.04	
	320 / 32 / 768	213.8	142.7	33.3		1.15	

Table 11: **GAIA trace-length sweep: cache residency (GPU hit rate %) and end-to-end speedup** (vLLM, seqs=2, Fig. 7). Percentage columns are GPU KV-cache hit rate, speedup is measured against LRU on the identical trace. Both models see the same traces.

Regime	Turns	Llama-8B			Llama-70B		
		LRU (%)	RoleKV (%)	Speedup (×)	LRU (%)	RoleKV (%)	Speedup (×)
Medium	22	3.7	34.6	1.11	5.2	34.6	1.24
	26	5.9	35.5	1.10	5.9	33.5	1.22
	40	13.6	43.0	1.13	13.5	40.8	1.25
Long	55	18.4	43.4	1.14	18.7	45.2	1.27
	75	21.0	50.4	1.20	21.7	50.0	1.36
	100	24.3	50.2	1.21	24.7	52.1	1.41

Table 12: **GAIA reductions by trace length** (Llama-8B). Larger is better. Short GAIA traces sit under the tightest relative memory pressure, so the per-trace reduction is largest there even though the end-to-end speedup is largest on long traces.

Turns	Reduction vs. LRU ↓ (%)	
	vLLM preempt	LMCache PCIe
22	42.9	63.8
75	25.0	26.5
100	18.2	19.1

scales with both concurrency and per-block KV size.

E.6 Dual-Mechanism Summary

Table 14 contrasts the two architectures.

Unified deployment. Stacking both engines activates both gains from one policy: durable blocks stay GPU-resident (cutting PCIe), and when GPU fills, reasoning is evicted before shared prefixes (cutting preempts). The combined stack delivers **18–32%** preemption reductions on Llama-8B over the same cells, so the two gains are simultaneous

consequences of one decision.

E.7 Statistical Significance of the Policy Effect

To test whether RoleKV’s reduction over LRU is significant rather than configuration-specific noise, we apply the paired one-sided Wilcoxon signed-rank test on the per-cell LRU–RoleKV differences in Tables 9 and 10. The pairing is exact by construction: identical seed, prompts, GPU budget, and decode length within each LRU/RoleKV cell pair (§5.1).

- **vLLM Llama-8B preempt count** (Table 9, 9 cells across decode/budget/concurrency): RoleKV strictly lower in 9/9 cells, $W=45$, $p=0.0039$.
- **LMCache PCIe traffic** (Table 10, 12 cells across Llama-8B/Qwen-32B/Llama-70B): RoleKV strictly lower in 12/12 cells, $W=78$, $p<0.001$.

The sign-test agreement across every sweep axis indicates the policy effect is structural rather than restricted to the headline cells. Wall-time speedups inherit the same per-cell pairing but compress through the engine’s compute/memory ratio

Table 13: **Tensor-parallel scaling** (GAIA, Llama-70B, vLLM, Fig. 8). Percentage columns are GPU KV-cache hit rate. Gains hold at every degree and are largest at TP=1/2 where per-GPU memory pressure is highest. TP=1 is an independent re-run of the single-GPU configuration of Table 11, so its residency differs by under one point.

TP (GPU _s)	Medium trace (22 turns)			Long trace (100 turns)		
	LRU (%)	RoleKV (%)	Speedup (×)	LRU (%)	RoleKV (%)	Speedup (×)
1	4.3	33.8	1.24	24.9	51.7	1.41
2	3.7	31.9	1.21	24.4	51.6	1.34
4	4.8	30.8	1.15	24.4	51.7	1.25
8	4.4	34.4	1.08	24.9	52.0	1.23

Table 14: **Dual-mechanism summary.** The same RoleKV policy adapts to the engine bottleneck: vLLM discard-on-preempt makes *compute* the cost (RoleKV cuts preempts), LMCatch offload-to-CPU makes *PCIe* the cost (RoleKV cuts transfers). Portable across 2 engines × 3 models.

	vLLM (preempt-discard)	vLLM + LMCatch (CPU-offload)
<i>What happens on memory pressure?</i>	V1 scheduler <i>preempts</i> a request and discards its KV cache.	LMCach <i>offloads</i> evicted blocks to a CPU memory pool.
<i>What does eviction cost?</i>	Re-prefill compute: ~0.5 s (Llama-8B), ~1.5 s (Qwen-32B) per preempt.	PCIe data movement: ~10 μ s setup + bytes/bandwidth per fetch.
<i>What does LRU do wrong?</i>	Evicts system/instruction blocks $\Rightarrow$ GPU prefix-cache breaks $\Rightarrow$ preemption cascade.	Evicts system/instruction blocks $\Rightarrow$ re-fetch from CPU on every prefix-share $\Rightarrow$ massive PCIe traffic.

(§5.3). The Wilcoxon test on Llama-8B/Llama-70B speedups yields $p < 0.05$, while Qwen-32B speedups are not significantly distinguishable from $1.0 \times$ ($p = 0.31$).

F Priority Function: Auxiliary Components and Hybrid Variant

Two pieces of Eq. 3 are elaborated here: the action-prediction term and a hybrid variant using current-turn attention. The ordering proof is in App. A.1.

F.1 Action Predictor

The boost is a sparse factor $\beta_{r,\hat{a}} \in [0.3, 3.0]$ keyed on the role tag r and the predicted next action $\hat{a}$, defaulting to $\beta = 1$ for pairs outside the table (e.g. $\hat{a} = \text{reflect}$ sets $\beta_{\text{reasoning}, \text{reflect}} = 3.0$, since the model is about to re-read its chain-of-thought). Table 15 compares four lightweight action predictors on HotpotQA-Llama-8B. Markov is both the simplest and the best on the accuracy-latency frontier (74.2% at <0.01 ms against TF-IDF’s 68.4% at 2.3 ms), so it is what we deploy. Table 16 reports the corresponding Bonferroni-corrected Dunn post-hoc analysis. A Kruskal-Wallis test rejects equal accuracy across the four methods ($H = 23.4$, $p < 0.001$), and the Dunn tests place Markov above each alternative at $p < 0.05$.

F.2 Hybrid Variant

Eq. 3 uses only the *prior* decay estimate. The hybrid variant adds the current-turn observed attention $\hat{a}_i$:

$$\text{priority}_{\text{hyb}}(b_i) = \underbrace{e^{-\text{age}(b_i)/\tau_r}}_{\text{decay prior}} \tau_r \cdot \underbrace{(1 + \alpha \hat{a}_i / \bar{a}_{0,r})}_{\text{current-turn corr.}} \cdot \text{boost}(b_i) \quad (4)$$

where $\bar{a}_{0,r}$ is the mean initial attention for role r and $\alpha = 0.5$ mixes the two, so blocks attended above their role’s prior gain priority and those below lose it. We evaluate on a trace-driven Llama-8B simulation (70 turns, 1 000 eviction decisions per budget, 10–90% of GPU memory). Table 17 reports the 25%-budget result.

Hybrid closes >99.9% of the LRU-to-Oracle gap at every budget $\geq 20\%$. The marginal 2.6 pp over base RoleKV recovers cases where the decay prior is out-of-distribution, at the cost of one extra attention-map read per forward pass (<1 ms), which suits latency-tolerant workloads.

G Role Tagger: Algorithm, Marker Map, and Robustness Methodology

The marker map, ground-truth protocol, confusion matrix on HotpotQA-Llama-8B ($n = 8,743$ blocks,

Table 15: **Action predictor comparison** (HotpotQA ReAct, 50 ep., Llama-8B). Markov wins on accuracy–latency, TF-IDF adds >2 ms/turn for only 5.8 pp.

Method	Accuracy	Latency	Description
Majority vote	42.1%	<0.01 ms	Most frequent action in session
Keyword matching	61.3%	<0.01 ms	Rule-based on last tool response
TF-IDF context	68.4%	2.3 ms	TF-IDF over role-tagged context
Markov (last act)	74.2%	<0.01 ms	Transition matrix $P(a_t a_{t-1})$

Table 16: **Dunn post-hoc** (Bonferroni-corrected over $\binom{4}{2}=6$ pairs). Markov beats every other method at $p < 0.05$.

Pair	Method A vs B	z-score	p (corrected)
Markov vs Keyword	74.2% vs 61.3%	4.21	<0.001
Markov vs TF-IDF	74.2% vs 68.4%	2.14	0.032
Markov vs Majority	74.2% vs 42.1%	6.77	<0.001
TF-IDF vs Keyword	68.4% vs 61.3%	2.07	0.038

Table 17: **LRU-to-Oracle gap closure** (Llama-8B, 70 turns, 25% budget). Hybrid closes >99.9%.

Policy	Wrong	Gap closed
LRU	445	0%
RoleKV	12	97.3%
RoleKV-Hybrid	1	99.9%
Oracle	0	100%

150 episodes), and engine-side lifecycle hooks for the Role Tagger (§4.1).

G.1 Marker Map per Chat Template

The tagger is parameterized by one template-specific table of marker token-id sequences (Table 18). Adding a template requires one row.

G.2 Reference Implementation

The per-turn FSM loop is Algorithm 1 (§4.1). It costs one integer comparison per token, < 0.1 ms/turn on host CPU, and one byte per KV block. Markers are matched as *token-id sequences* rather than decoded strings, because ids are stable under BPE merges while substring matching is brittle (`Thought :` splits differently depending on preceding whitespace). Short markers use a hash table, longer ones a small Aho–Corasick automaton compiled once per template. Changing model family regenerates the marker table but leaves the algorithm unchanged. Subtypes f come from a single-pass scanner and feed only the action boost, so the priority sort is unaffected when it is disabled.

G.3 Robustness Methodology

Ground truth. Tags come from parsing the agent-harness JSON and re-aligning role spans to the 16-token block grid under the same first-token-

wins rule used at inference, with assistant messages split by a line-anchored regex over `Thought:/Action:/Observation:/Result:.` The procedure is fully deterministic, with no LLM-as-judge and no human annotation, so any disagreement with the live tagger is a true error rather than a labeling artifact.

Sampling and confidence intervals. We score every block in all 150 held-out HotpotQA-Llama-8B episodes ($n=8,743$, mean 58 blocks/episode). The 95% Wilson binomial CI on accuracy is 99.58% [99.42, 99.69]. We use Wilson rather than the Wald (normal-approximation) interval because 0.9958 is too close to the upper boundary for Wald to retain correct coverage. The block distribution is reasoning-dominated (71%, with 14/10/3/2% action/system/tool_output/instruction), reflecting HotpotQA’s `Thought:-heavy` ReAct shape.

Confusion matrix and per-role metrics. Table 19 gives the row-normalized confusion matrix. All off-diagonal mass concentrates in one cell, `action→tool_output` (3.1% of action blocks, driven by Llama-8B’s hallucinated inline `Result:/Output:/Return:` markers), per-role precision/recall is 100/100 for system, instruction, reasoning, 100/96.92 for action, and 87.99/100 for tool_output. Why this single-cell confusion is conservative under Eq. 3 is argued in §6.5.

G.4 Lifecycle and Engine Hook Points

When and where the tagger runs. The tagger executes on the *request-submission thread* (host

Table 18: **Marker→role map per chat template.** Markers are pre-tokenized once per template, per-turn cost is one linear token-id scan. Subtype f (parsed from the action body, e.g., `search`) feeds the action predictor only.

Template	Marker (decoded)	→ Role
<i>Positional (all templates):</i>		
any	1st chat message	system
any	1st user message	instruction
<i>ReAct:</i>		
ReAct	Thought:	reasoning
ReAct	Action:	action
ReAct	Observation:	tool_output
<i>Function-calling (Llama-3, Qwen-3):</i>		
FC	pre-<tool_call>	reasoning
FC	inside <tool_call>	action
FC	inside <tool_response>	tool_output
<i>Llama-3 chat header (all formats):</i>		
Llama-3	< start_header_id >system	system
Llama-3	< start_header_id >user	instruction
Llama-3	< start_header_id >assistant	reasoning (default)
<i>Subtype capture (optional):</i>		
any	body of action block	action[f]
any	body of tool_output block	tool_output[f]

Table 19: **Role-Tagger confusion matrix** (row-normalized %, HotpotQA-Llama-8B, $n=8,743$, 99.58% accuracy). All off-diagonal mass in one cell (action→tool_output, 3.1%), conservative under Eq. 3.

GT ↓ / Pred →	sys	ins	rea	act	tool	n
system	100	0	0	0	0	900
instruction	0	100	0	0	0	164
reasoning	0	0	100	0	0	6206
action	0	0	0	96.9	3.1	1202
tool_output	0	0	0	0	100	271

CPU), *before* the request enters the engine subprocess. Its only inputs are (a) the chat template, (b) the tokenized prompt, and (c) (for subsequent turns) any newly committed tokens since the previous turn. Output: an in-process RoleMetadata map (Python dict[int → uint8]) keyed by the engine-assigned block_id. The engine subprocess sees this map only through the Offloading-Connector callback at eviction time (App. D), so no GPU-side cost is incurred.

Block recycling. When vLLM frees a block (on request completion or preemption), its RoleMetadata entry is dropped on the same call-path, the FSM re-runs from scratch on re-allocation, so a recycled block carries the new request’s tag with no cross-request leakage.

Prefix-cache hits. On a prefix-cache hit the existing block and its RoleMetadata entry are reused as-is, tag consistency across sharers is au-

tomatic because two requests with a shared token prefix have identical marker positions, so the tagger would emit identical tags anyway.

H Additional Ablations

H.1 Per-Role Contribution

Table 20 decomposes the policy. **System pinning is the dominant lever** (−23.2 pp without it, 31.3% → 8.1%), role-conditioned TTLs add 3–7 pp each, and the Markov boost contributes ~2.4 pp, so the gains follow from what each component does rather than joint tuning. This decomposition is measured on **HotpotQA**, whose short episodes leave most blocks in system or instruction context, so pinning carries almost all the benefit. The balance shifts on long traces, where durable tool_output becomes decisive (Table 5). This ranking should be interpreted as specific to short-trace workloads.

Table 20: **Per-role ablation (HotpotQA, Llama-8B, Prmp. from vLLM seqs=64, PCIe from LMChe peak).** System pinning dominates, role TTLs add gains, Markov refines.

Variant	Prmp.↓	PCIe↓	Spd.
Full RoleKV	31.3%	31.4%	1.34×
– system pin	8.1%	11.2%	1.07×
– instr. TTL=reason	24.2%	23.5%	1.21×
– tool TTL=reason	27.8%	26.4%	1.28×
– Markov boost	28.9%	28.6%	1.29×
LRU (role-blind)	0%	0%	1.00×

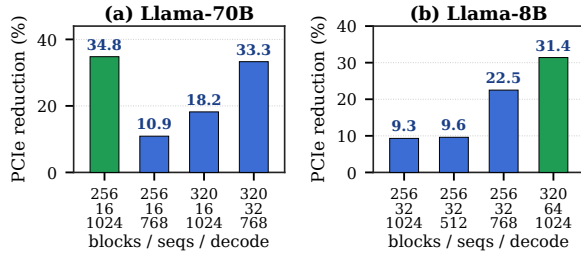


Figure 9: **Per-cell PCIe reduction** (LMCache, measured cells of Table 10). Green marks the largest reduction per model. Cells vary GPU block budget, concurrent sequences and decode length jointly, so these are joint memory-pressure effects rather than a single-axis sweep. Discussed in App. H.3 and App. H.4.

H.2 Robustness to Decay Calibration

RoleKV consumes the decay fit only through the *role ordering* it induces (Prop. 1), and that ordering is stable. Under $\pm 50\%$ perturbation of the measured constants the eviction order is essentially unchanged (Spearman $\rho \geq 0.94$) and end-to-end gains move little. Transient roles sit below durable ones by a wide margin at every age measured (Table 1), so the rank separation absorbs sizeable fit error. Calibration refines the policy, the ordering decides what is dropped.

H.3 Memory-Budget Sensitivity

Across the four measured Llama-70B cells the PCIe reduction spans 10.9–34.8% (Fig. 9(a) and Table 10, both in App. E). Two pairs of cells isolate a single knob. Loosening the budget from 256 to 320 blocks at fixed concurrency and decode length shrinks the reduction from 34.8% to 18.2%, matching the expectation that the eviction policy matters less as the cache approaches fitting. Shortening decode from 1024 to 768 tokens at a fixed 256-block budget shrinks it from 34.8% to 10.9%, because a shorter trace produces fewer evictable reasoning blocks.

H.4 Concurrency Sensitivity

On Llama-8B the reduction is 9.3% and 9.6% at 32 concurrent sequences and peaks at **31.4%** in the 64-sequence cell (Fig. 9(b)). C shared-prompt sequences amplify each retained block into C saved round-trips. That cell also raises the budget from 256 to 320 blocks, so it reflects joint memory pressure rather than a controlled concurrency sweep.