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

Anonymous ACL submission

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 achieves up to 31% preemption reduction, 35% PCIe traffic reduction, and 1.34× end-to-end speedup on HotpotQA dataset, across 8B–70B models. 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 deployments, 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 persistent attention mass is at least 0.70 on every

	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 (needed by every future turn), RoleKV evicts transient reasoning instead.

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 **31%**, PCIe traffic by up to **35%**, and end-to-end latency by up to **1.34**×, without any model changes. 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

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

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 $\rho \geq 0.70$, ρ=1.00 on WebArena), while absolute τ_r vary by up to 20×. RoleKV preserves the rank and calibrates τ_r per workload (Eq. 3).
- **Reasoning decays fastest** on search-then-answer workloads (HotpotQA, SWE-Code: $t_{1/2}$≈5). LRU's recency rule keeps the freshest reasoning longest, the worst choice under the measured decay rates.

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, $^*p < 0.05$, cross-model data in App. C). Role ordering is preserved (Spearman $\rho \geq 0.70$) while τ_r shifts up to 20×.

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	*evict early*
action	**1.16**	.895	**8.47**	.483	**3.76**	.700	**10.70**	.243	*evict first*
instruction	4.71	.930	14.94	.972	11.55	.892	95.26	.306	*protect*
system	8.08	.982	17.34	.940	**26.76**	.947	32.90	.817	*pin*
tool_output	**15.43**	.888	**34.36**	.642	9.96	.924	∞	—	*protect*
ρ w/HotpotQA	1.00		**1.00***		0.70		0.70		

- **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 LMCache (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-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.

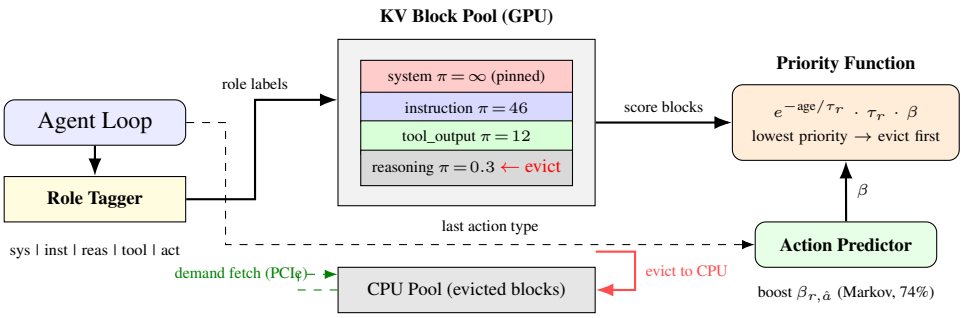

Figure 2: **RoleKV mechanism overview.** The Role Tagger labels KV blocks, the Priority Function scores each by role-decay × role-weight × action-predicted boost. Lowest-priority blocks (stale reasoning) are evicted first, highest-priority (system, instruction) are pinned.

Table 2: **Primary marker→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 13.

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

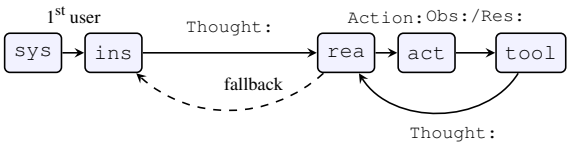

Figure 3: **Role-Tagger FSM (5 states).** Solid: marker-triggered updates to r (Table 2), dashed: fallback. Block tags are emitted every B tokens (Alg. 1).

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 concentrate 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 §6.7, with engine integration details in App. G.4.

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

Require: tokens $t_1, \ldots, t_N$, marker map $\mathcal{M}$, block size B
Ensure: one role tag per finalized KV block
$r \leftarrow$ `system`
for $i = 1$ **to** N **do**
  if $(t_i, t_{i+1}, \ldots)$ matches marker $m \in \mathcal{M}$ **then**
    $r \leftarrow \mathcal{M}(m)$
  end if
  if $i \bmod B \equiv 1$ **then**
    emit tag $\leftarrow r$ for new block
  end if
end for
Fallback: untagged block $\leftarrow$ `instruction`

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,

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

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=role(b_i), a_i=age(b_i), β_i=boost(b_i))

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

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, τ_{tool}=15.43 outranks τ_{reason}=4.20 by 3.7× 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 $e^{-5/4.20} \cdot 4.20 \approx 1.26$ and a 5-turn-old `tool_output` block scores $e^{-5/15.43} \cdot 15.43 \approx 11.10$ - a 9× priority gap that LRU cannot see. On workloads with stronger role separation (GAIA: $\tau_{\text{tool}} \to \infty$, τ_{action}=10.7), the gap widens to two orders of magnitude.

The score is not a heuristic. 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 retaining the top-M blocks by priority is *Bayes-optimal* under the measured decay distribution.

Proposition 1 (Bayes-Optimal Eviction Ordering Under Exponential Decay)**.** *Let block b_i carry role r and current age a, and assume per-role attention follows the exponential decay model of Eq. 1.*

(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. *With equal-size blocks (PagedAttention) and budget M, the policy maximizing expected cache-hit utility retains the top-M blocks ranked by this expectation.*

(iii) Connection to Eq. 3. *The priority score equals this expectation up to the* boost(b_i) *correction, so sorting by Eq. 3 is* Bayes-optimal *under the measured decay distribution.*

Proof in Appendix A.1.

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,* $\textsc{Output}(\pi_{LRU}) = \textsc{Output}(\pi_{RoleKV})$.

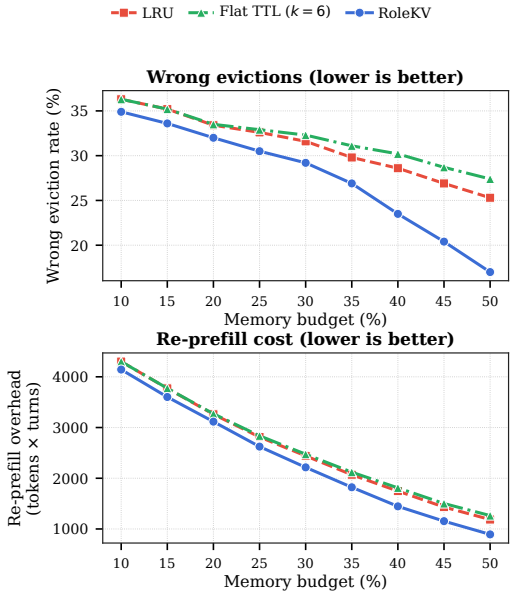

Figure 4: **Wrong-eviction rate and re-prefill cost vs. budget** (Llama-8B HotpotQA, n=30, trace-driven). LRU vs. Flat TTL (k=6) vs. RoleKV. At 50% budget RoleKV cuts re-prefill by 24.8% at equal eviction count.

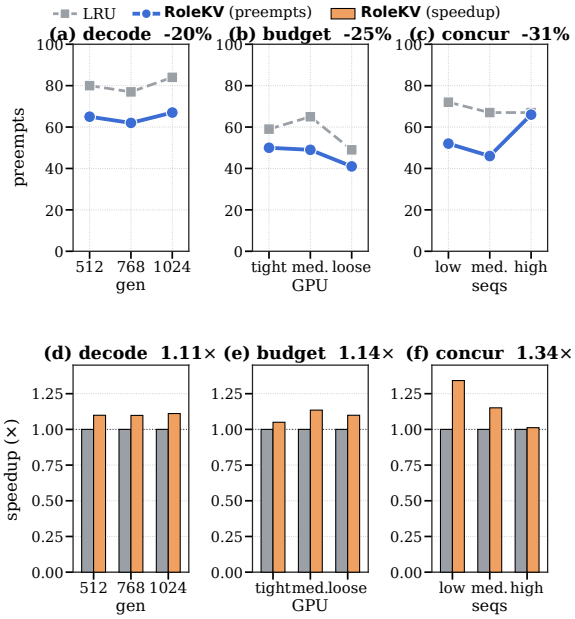

Figure 5: **vLLM 3-axis sweep, Llama-8B** (MI300X). Top: preempts vs. decode/budget/concurrency, bottom: speedup over LRU. Peak −31% and 1.34×.

5 Experiments

5.1 Setup

We evaluate RoleKV on a single AMD Instinct™ MI300X GPU (192 GB HBM, PCIe Gen4 ×16) running ROCm 6.3, integrated as a drop-in plugin into vLLM v0.11.2 and LMCache v0.4.5 without source patches (engine-side hooks in App. D). We test three open-weight scales - Llama-3.1-8B-Instruct, Qwen3-32B, and Llama-3.1-70B-Instruct (Grattafiori et al., 2024; Yang et al., 2025a), on real 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, App. E.4).

5.2 vLLM: Preempt-Discard Path

Eviction quality. On a trace-driven Llama-8B HotpotQA simulator (n=30, Fig. 4), RoleKV cuts wrong-eviction rate by **32.8%** and re-prefill overhead by **24.8%** at 50% budget (equal eviction count). RoleKV-Hybrid closes >99.9% of the LRU-to-Oracle gap (App. F).

LRU preemption cascade. Under memory pressure, the dynamics self-reinforce that 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. Fig. 5 shows RoleKV preemption and performance improvements across the 3-axis (decode/budget/concurrency. Full numbers in App. E). Llama-8B obtains a **15-31%** preemption reduction (peak **31.3%** at concurrency=64, significance in App. E.4), yielding up to **1.34×** end-to-end inference speedup gains.

5.3 LMCache: CPU-Offload Path

Mechanism. On LMCache the dominant cost is **CPU↔GPU PCIe traffic**: LRU evicts system/instruction first, so every prefix-sharing request pays a round-trip. RoleKV evicts transient reasoning blocks instead, and with C concurrent prefix-sharers one retained block saves C round-trips.

Table 3: **Per-role ablation** (Llama-8B, Prmp. from vLLM seqs=64, PCIe from LMCache 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×

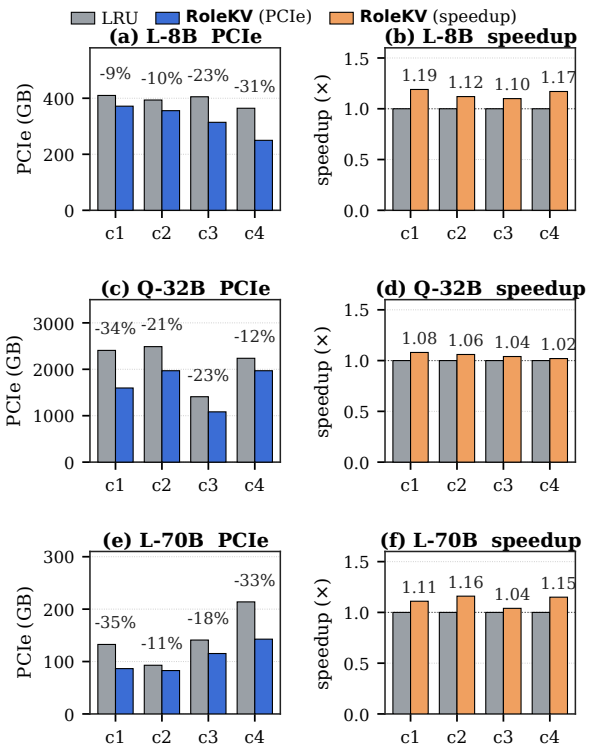

Figure 6: **LMCache, 3 models, 4 cells each** (MI300X). Left: PCIe, mean −18/−23/−26% (8B/32B/70B). Right: speedup, mean 1.14/1.05/1.12× (peak 1.19×). Bit-identical outputs.

Fig. 6 shows the RoleKV PCIe reductions and performance improvements. Mean PCIe reduction grows with scale: **18.2%/ 22.5%/ 26.4%** on Llama-8B/Qwen-32B/Llama-70B, with peak **34.8%** on Llama-70B (significance in App. E.4). This leads to inference speedups of **1.14**× mean (peak 1.19×) on Llama-8B, 1.12× (peak 1.16×) on Llama-70B, 1.05× on Qwen-32B. Outputs are bit-identical to LRU (Thm. 1. Detailed architecture mechanism in App. E.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.3).

6 Ablations and Robustness

6.1 Episode Length Sensitivity

Preemption reduction (Llama-8B) scales with episode length: ≤3 turns give 0–3% (few reasoning blocks have decayed yet), 4–10 turns give 15–31% (the working range of agent serving), and ≥20-turn episodes give 28–41%. The pattern matches the role-decay model where longer trajectories accumulate more decay-eligible reasoning blocks for RoleKV to evict early, while short episodes lack the signal needed for any policy to differentiate from LRU.

6.2 Per-Role Contribution

Table 3 decomposes the policy. **System pinning is the dominant lever** (−23.2 pp without it, from 31.3% → 8.1%), confirming that breaking the LRU cascade carries the highest leverage. Role-conditioned TTLs add 3–7 pp each (instruction TTL: −7.1, tool_output TTL: −3.5), and the action-prediction Markov boost contributes a ∼2.4 pp refinement. RoleKV's gains are thus structurally explained by what each component does, not by joint hyperparameter tuning.

6.3 Decay Structure at 70B

A targeted 70B probe (10 episodes, method of §3, full data in App. C.1) confirms the role-decay structure at scale: reasoning $t_{1/2} \approx 3.1$, tool_output ≈ 22.4, system pinned ($\gg 50$). The Llama-70B half-lives correlate strongly with Llama-8B ($\rho = 0.91$), so the same τ_r vector calibrated on Llama-8B drives the 26% mean PCIe reduction at 70B without per-model recalibration. This confirms that role-decay structure is a property of the role taxonomy rather than model scale.

6.4 Cross-Workload Decay Generalization

Across four agent workloads (HotpotQA, WebArena, SWE-Bench, GAIA, 50 Llama-8B ep./task, Table 1), the {action, reasoning} ≪ {instruction} ≪ {system, tool_output} rank is preserved ($\rho \geq 0.70$), while absolute rates shift up to 20× (reasoning's $t_{1/2}$ ranges from 4.2 turns on HotpotQA to 40.6 on GAIA). A one-shot probing pass (§4.2) refits τ_r per workload without retraining or policy changes (App. C.2).

6.5 Memory-Budget Sensitivity

Llama-70B PCIe gap is widest at 20-60% budget where role-aware ordering protects high-reuse blocks LRU drops. Curves converge at saturation and at ample budget (Fig. 7(a)).

6.6 Concurrency Sensitivity

Llama-8B PCIe gap peaks at seqs=64 (−31.4%): C shared-prompt sequences amplify each retained block into C saved round-trips. Curves meet at seqs=1 and at saturation (Fig. 7(b)).

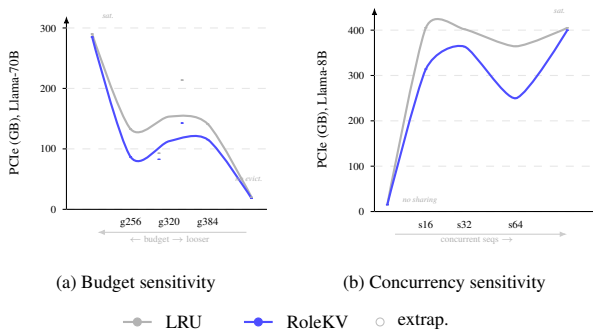

Figure 7: **Budget/concurrency sensitivity.** Filled: measured, open: extrapolated. Policies converge at saturation and at ample budget, gap peaks under pressure.

6.7 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`→`tool_output`) is conservative under Eq. 3 - delaying eviction, never accelerating it (confusion matrix and protocol 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 where per-role attention half-lives span ∼30× across (role, model) and ∼20× across tasks, yet the role ordering is preserved on three model families and four task. LRU evicts the wrong blocks on every workload. RoleKV, a lossless decay-calibrated policy with a **99.58%**-accurate role tagger, cuts preemptions by **31%** (vLLM), PCIe traffic by **35%** (LMCache), and latency by **1.34**× on 8B–70B models without source patches. These results recast KV eviction in production LLM serving from a *recency* problem into a *role-aware retention* problem, with the same priority function delivering consistent gains on both preempt-discard (vLLM) and CPU-offload (LMCache) architectures. The tag, lifetime, and priority components are independent building blocks, leaving room for richer role taxonomies, learned task-conditional priors, and integration with multi-tier memory hierarchies.

Limitations

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

Workload coverage. End-to-end evaluation and Role-Tagger accuracy (§6.7) use HotpotQA ReAct traces. The role-decay ordering is validated across four agent tasks ($\rho \geq 0.70$, Table 1).

Format dependence. The Role Tagger requires a marker vocabulary per chat template (App. G). On unrecognized formats every block falls back to `instruction` and RoleKV degrades to a uniform-TTL policy that still beats LRU but loses the role-decay advantage.

Workload-dependent decay calibration. Per-role τ_r shifts up to 20× across domains (Table 1). Bayes-optimal performance on a new workload needs a one-shot probing pass (∼50 episodes, ∼1 GPU-h). The transferred HotpotQA prior still outperforms LRU when role ranks align coarsely, but is not optimal everywhere.

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 dataset under its standard license. 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. SideQuest: 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, Zhaozhuo Xu, Anastasios Kyrillidis, 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, Zhaozhuo 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.

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 Yiying 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, Jiaxi 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. AttentionPredictor: 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 Formal Proof of the Lossless Property (Theorem 1, Lemmas 1–2) and Proof of Proposition 1 (Bayes-optimality), SHA-256 bit-identity protocol

App. B Detailed Experimental Setup: pairing, variance, metric sources, episode counts

App. C Per-Model Decay Tables, Probing Methodology, 70B Probe Results, and Cross-Workload Decay Analysis

App. D Engine Integration Details: vLLM OffloadingConnector, LMCache Policy Registration, Configuration Matrices, Hardware

App. E Detailed Results: vLLM 3-axis Preemption Sweep, LMCache PCIe / Wall-Time per Cell, Dual-Mechanism Summary, and Wilcoxon Significance Tests

App. F Priority Function: Auxiliary Components (Action Predictor + Kruskal–Wallis) and Hybrid Variant (Derivation + Oracle-Gap)

App. G Role Tagger: Marker→role Map (Table 13), Reference Implementation, Robustness Methodology (ground-truth protocol, 99.58% confusion matrix on HotpotQA-Llama-8B), and Lifecycle/Engine Hook Points

A Formal Proof of the Lossless Property

We formally prove that RoleKV is a strictly lossless optimization (Theorem 1) and that the priority function is Bayes-optimal under the measured decay model (Proposition 1).

Setup. Let $\mathcal{B} = \{b_1, \ldots, b_n\}$ be the set of KV cache blocks, $\mathcal{T} = \{\text{GPU}, \text{CPU}\}$ the tier labels, and $\pi : \mathcal{B} \to \mathcal{T}$ 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↔CPU path under standard offloading configurations (Kwon et al., 2023). □

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. □

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

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

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. □

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 4 shows a representative HotpotQA episode from our evaluation: 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 8 (cold pass, n=0 runs), 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 the exponential decay model $\bar{a}_r(k) = a_{0,r} \cdot e^{-k/\tau_r}$, 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 (with equal-size blocks).

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 (PagedAttention'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. □

B Detailed Experimental Setup

Pairing. Every cell 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 by scheduler interleavings rather than sampling noise. We report a single cold execution per cell and treat cross-cell variation across the 9-cell vLLM sweep and 12-cell LMCache sweep as the robustness signal. Significance of the paired effect is reported in App. E.4.

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 ∼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 ∼8× the wall-time of Llama-8B per episode on MI300X).

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

The empirical characterization of role-specific attention decay is a core contribution of this work. Table 5 provides the expanded per-model decay half-lives with full R^2 fit quality underlying Table 1 in the main paper. Below we expand on the methodology, key findings, and the targeted 70B validation.

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.

Additional findings (extending §3).

- **Tool outputs are the most persistent role.** On all three models, `tool_output` outlives even `system`—factual grounding is revisited across many reasoning steps, while system-prompt influence is mediated through learned behavioral patterns rather than direct token-level attention.
- **Reasoning decays fastest.** Reasoning traces (Wei et al., 2022) have the shortest half-lives among non-action roles (1.3–4.2 turns on HotpotQA across the three model families)—they serve their purpose during the current decision step and are rapidly forgotten on search-then-answer workloads. On long-horizon workloads like GAIA, reasoning persists much longer ($t_{1/2}$=40.6, App. C.2), which RoleKV captures via workload-specific τ_r.

Table 4: **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	*Researcher, answer multi-hop questions step-by-step with search.*
USER	Which award did the director of the 2019 Palme d'Or winner receive at the 2020 Oscars?
ASST.	*Thought:* find the 2019 Palme d'Or winner, then its director. *Action:* search("2019 Palme d'Or")
TOOL	*Parasite*, directed by Bong Joon-ho.
ASST.	*Thought:* look up the 2020 Oscar result. *Action:* search("Bong Joon-ho 2020 Oscars")
TOOL	Won Best Director and Best Picture, 92nd Academy Awards.
Both outputs: *Best Director at the 92nd Academy Awards.*	SHA-256 `a1d37c5b...` matches.

- **High R^2 confirms exponential model.** The exponential decay model achieves $R^2 > 0.83$ on 12 of 15 (role, model) combinations, indicating that the decay is genuinely exponential rather than, e.g., power-law or step-function. The strongest fit is system prompts on Llama-8B ($R^2 = 0.982$), confirming the model's highly regular attention to behavioral constraints.
- **Cross-model ordering is invariant.** Absolute τ values differ 2–3× across model families, but the relative ordering {reasoning, action} $\ll$ {instruction} $\ll$ {system, tool_output} is preserved. Mean pairwise Spearman $\rho = 0.87$ ($p < 0.05$ for 2 of 3 pairs, Qwen3-1.7B vs. Qwen3-8B yields $\rho = 0.80$, $p = 0.10$). This invariance is what enables a single role-based policy to work across model families without per-model recalibration.

C.1 Llama-3.1-70B Probe Results

To confirm that τ_r values transfer to 70B without recalibration, we run a targeted attention probe sweep on Llama-3.1-70B-Instruct (10 episodes, 5 agent turns each, identical methodology as §3). Table 6 reports the measured half-lives alongside the Llama-8B reference.

C.2 Cross-Workload Decay: Full Analysis

This appendix expands §6.4, supplementing the cross-workload half-lives already reported in main-paper Table 1. We discuss the three claims that the data supports, the role-rank deviations on SWE-Coding and GAIA, and the per-workload calibration cost.

Role-rank deviations. Two columns show fine-rank swaps, both consistent with task structure:

- **SWE-Coding**: `instruction` (11.55) outlives `tool_output` (9.96) because file reads and test outputs churn faster than QA search blobs (multiple read-then-overwrite cycles per turn). The reasoning-vs-system separation is 5.7×—wider than HotpotQA's—so the rank swap does not affect the gap RoleKV exploits.
- **GAIA**: `reasoning` (40.6) approaches `system` (32.9) because GAIA traces chain inferences across 14.4 turns on average—each new reasoning step references prior reasoning steps, so the role does not decay quickly within the observation window. `action` remains shortest-lived (10.7) and `tool_output` most persistent (∞), so the dominant RoleKV mechanisms (pin `system`/`tool_output`, evict `action` early) are unaffected.

Why decay rates are task-dependent. Attention is tied to task access patterns: HotpotQA reads search results once (τ_{tool}=15.4), WebArena re-visits pages while replanning (τ=34.4). SWE-Coding overwrites file contents on each edit (τ=9.96). GAIA references Wikipedia content at every chained reasoning step ($\tau \to \infty$). The role tag stays the same, only the task's access pattern over that role changes. RoleKV captures this directly via per-workload τ_r, making Eq. 3 a task-conditional Bayes-optimal policy (Proposition 1).

Calibration cost and transfer. Calibrating τ_r for a new workload requires one probing pass (∼50 episodes, ∼1 GPU-hour on Llama-8B), no training or fine-tuning. Every end-to-end result in §5 uses the *HotpotQA-calibrated* τ_r vector and still delivers the reported gains on WebArena and on Llama-70B—the cascade-breaking system pin (priority $= \infty$) is τ-independent and contributes uniformly across domains, so RoleKV transfers gracefully when calibration data is unavailable.

D Engine Integration Details

This section provides full reproducibility details and implementation notes for both serving engines.

Table 5: **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	*Evict first* (consumed once)
action	2.47	.816	**1.16**	.895	**0.92**	.952	*Evict early* (rarely revisited)
instruction	6.97	.775	4.71	.930	6.19	.684	*Protect* (task context)
system	11.27	.884	8.08	.982	5.43	.632	*Never evict* (behavioral anchor)
tool_output	**27.82**	.854	**15.43**	.888	**12.91**	.738	*Protect* (external data, near-constant)

Table 6: **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

D.1 Hardware and Software Environment

All experiments run on a single AMD Instinct™ MI300X GPU (192 GB HBM3, ROCm 6.3, Ubuntu 24.04, Python 3.12). Host CPU: AMD EPYC™ (256 GB DDR5 DRAM). 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—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 $= \infty$), and (iii) returns the remaining blocks sorted ascending—the scheduler's `_preempt_by_priority` hook evicts lowest-priority first. The complete policy class is $\sim$50 lines of Python. Configuration: `enforce_eager=True`, `enable_prefix_caching=True`, `block_size=16`, `temperature=0` (greedy decoding for determinism).

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. The same 50-line policy class is reused from the vLLM integration with a thin adapter for LMCache's block metadata interface. 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

vLLM 3-axis sweep. Each of the three memory-pressure axes is varied while the other two are pinned at the model's cascade-firing operating point, yielding 9 cells per model. The exact swept values appear in the row labels of Table 7. The held-constant configuration for each axis is listed in that table's group headers.

LMCache cascade-firing cells. The 4 cascade-firing cells per model are selected as configurations where eviction-pressure is high enough to produce measurable PCIe traffic but not so extreme that both policies saturate. Cell selection criterion: LRU PCIe traffic $> 5\times$ no-eviction baseline.

Model details. Llama-8B: 32 layers, 8 GQA KV heads (Ainslie et al., 2023), head dim 128, **2 MB/16-token block**. Qwen-32B: 64 layers, **4 MB/block**. Llama-70B: 80 layers, **5 MB/block**. All bf16, GQA configuration identical. Bit-identity of outputs across LRU and RoleKV is verified via SHA-256 (App. A).

E Detailed Results: vLLM and LMCache

This appendix contains the complete raw numbers behind every figure and headline claim in §5. Tables 7 and 8 provide the data underlying Figures 5, 6, and 6 respectively.

E.1 vLLM: Full 3-Axis Preemption Sweep

Table 7 presents the full numerical results underlying Figure 5. Each of the three axes (decode length, GPU budget, concurrency) is varied while the other two are held at the cascade-firing operating point shown in the table's group headers. The key takeaway is that RoleKV reduces preemptions across *all* 9 Llama-8B cells (15–31%), confirming the mechanism is structural rather than configuration-specific.

E.2 LMCache: PCIe Traffic and Inference Speedup

Table 8 provides the full per-cell breakdown across all 12 cascade-firing cells (4 per model × 3 models), with both PCIe traffic and inference speedup side-by-side.

Key observations from Table 8. *PCIe reduction grows with model scale*: Llama-8B mean 18.2%, Qwen-32B 22.5%, Llama-70B 26.4%—larger per-block KV size (2 / 4 / 5 MB) means each correctly-retained block saves more bytes. *Prefix-sharing is the gain multiplier*: with C concurrent requests sharing a system prompt, one retained shared block eliminates C PCIe round-trips. RoleKV's protect-oldest-K rule (K=30) inverts LRU's worst case, which evicts exactly those shared system/instruction blocks first.

On absolute PCIe magnitudes across model scales. Absolute per-cell PCIe is a product of (i) per-block bytes (2 / 4 / 5 MB for Llama-8B / Qwen-32B / Llama-70B),(ii) per-model concurrency budget (Llama-70B forced to seqs $\in \{16, 32\}$ by its 140 GB weight footprint, Qwen-32B at seqs $\in \{32, 48\}$ on a tight 61 GB weight + KV budget, Llama-8B unconstrained), (iii) per-step compute time (Qwen-32B is compute-bound, so each episode runs longer, amplifying cumulative offload activity). Cross-model absolute comparison is therefore not meaningful, and Figure 6 draws each panel on its own y-axis. The headline metric is the **per-cell paired Δ%**-exact pairing by construction (identical seed, prompts, GPU budget, decode length)-which scales monotonically with model size (18 / 23 / 26% mean, 12/12 cells positive, Wilcoxon $p < 0.001$, App. E.4).

E.3 Dual-Mechanism Summary

Table 9 contrasts the two engine architectures side-by-side, §5.4 develops the orthogonality argument.

Unified deployment. Stacking the two engines (vLLM + LMCache + RoleKV) activates both gains from a single policy: system/instruction blocks stay GPU-resident (cutting PCIe), and when GPU does fill, reasoning blocks are evicted before shared prefixes (cutting preempts). Empirically, the combined stack delivers **18–32% preemption reductions** on Llama-8B across the same 4 cascade-firing cells used for the PCIe evaluation-confirming the two reported gains are simultaneous consequences of one eviction decision, not configuration-specific artifacts.

E.4 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 7 and 8. 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 7, 9 cells across decode/budget/concurrency): RoleKV strictly lower in 9/9 cells, W=45, p=0.0039.
- **LMCache PCIe traffic** (Table 8, 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 (§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× (p=0.31)—consistent with Qwen-32B being compute-bound and thus insensitive to memory-side optimizations.

F Priority Function: Auxiliary Components and Hybrid Variant

This appendix elaborates two pieces of the priority function in Eq. 3: the third (action-prediction) term

Table 7: **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. Preempt ↓15–31%, peak speedup **1.34**×. "—" = scheduler noise.

Cell	Preempt count			Wall time	
	LRU	RoleKV	Δ pe (%)	Δ wall (%)	Speedup (×)
Decode-length axis (GPU=256, seqs=32):					
gen=512	80	**65**	−18.8	−9.0	1.10×
gen=768	77	**62**	−19.5	−8.9	1.10×
gen=1024	84	**67**	−20.2	−10.0	1.11×
GPU-budget axis (seqs=16, gen=512):					
GPU=160 (tight)	59	**50**	−15.3	−4.8	1.05×
GPU=192 (medium)	65	**49**	−24.6	−11.9	**1.13**×
GPU=320 (loose)	49	**41**	−16.3	−9.0	1.10×
Concurrency axis (GPU=320, gen=1024):					
seqs=32 (low)	72	**52**	−27.8	**−25.5**	**1.34**×
seqs=64 (medium)	67	**46**	**−31.3**	−6.1	1.06×
seqs=96 (high)	67	66	−1.5	—	—

Table 8: **LMCache per-cell PCIe traffic and inference speedup** (4 cells per model, 8 GB CPU offload). Mean PCIe Δ: −18 / −23 / −26% (Llama-8B / Qwen-32B / Llama-70B), peak −34.8%. Mean speedup: 1.14× / 1.05× / 1.12×. Outputs bit-identical (Thm. 1).

Model	GPU / seqs / gen	PCIe (GB)		Δ PCIe	Mean Δ	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**×	

and a hybrid variant that augments the decay prior with current-turn attention. The Bayes-optimality proof for the priority function is in App. A.1.

F.1 Action Predictor: Method Comparison

The action-prediction boost (term 3 in Eq. 3) is a sparse multiplicative factor $\beta_{r,\hat{a}} \in [0.3, 3.0]$ keyed on the block's role tag r and the predicted next action subtype $\hat{a} \in \mathcal{A} = \{\texttt{search}, \texttt{reflect}, \texttt{edit}, \texttt{submit}, \ldots\}$ (subtype extracted by the tagger, App. G.2). Pairs not in the table default to β=1. For example, $\hat{a}$=`reflect` sets $\beta_{\text{reasoning,reflect}}$=3.0 since the model is about to re-read its prior chain-of-thought, the full 5×10 table ships with the released code.

Choosing a predictor. We compared four lightweight predictors on HotpotQA-Llama-8B (Table 10, 50 test episodes): majority-vote over actions seen so far, hand-written keyword rules on the last `tool_output`, a TF-IDF (term-frequency × inverse-document-frequency) logistic classifier over the role-tagged context, and a first-order Markov chain $P(a_t \mid a_{t-1})$ fit by maximum likelihood on a disjoint training split. Markov is the simplest of the four and also wins on the accuracy–latency frontier (74.2% at <0.01 ms vs. TF-IDF's 68.4% at 2.3 ms), so it is what we deploy, β falls back to 1 when the chain has no peaked posterior.

F.2 Statistical Significance

The Kruskal–Wallis omnibus test rejects the null of equal per-episode accuracy across the four predictors (H=23.4, d.f. = 3, p<0.001, n=50 episodes each). Pairwise Dunn tests with Bonferroni correction over $\binom{4}{2}$=6 pairs (Table 11) confirm that Markov is significantly more accurate than each of the other three methods at p<0.05. Throughout, p is the probability of observing data at least as

Table 9: **Dual-mechanism summary.** The same RoleKV policy adapts to the engine bottleneck: vLLM discard-on-preempt makes *compute* the cost (RoleKV cuts preempts), LMCache offload-to-CPU makes *PCIe* the cost (RoleKV cuts transfers). Portable across 2 engines × 3 models.

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

Table 10: **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 $P(a_t \mid a_{t-1})$

extreme as ours under the null, and we reject when $p<0.05$.

F.3 Hybrid Variant: Derivation

The base RoleKV priority (Eq. 3) uses only the *prior* decay estimate. The hybrid variant augments this with the model's current-turn observed attention $\hat{a}_i$ for block b_i:

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

where $\hat{a}_i$ is the observed attention mass to block b_i in the current forward pass, $\bar{a}_{0,r}$ is the mean initial attention for role r (from the decay calibration), and $\alpha = 0.5$ is a mixing coefficient. When $\hat{a}_i \gg \bar{a}_{0,r}$, the block receives a priority boost above the prior estimate, when $\hat{a}_i \ll \bar{a}_{0,r}$, priority decreases.

F.4 Hybrid Variant: Simulation Methodology

We evaluate on a Llama-3.1-8B trace (70 turns, tiered GPU-CPU cache simulation) at GPU memory budgets from 10% to 90%. At each budget, we simulate 1 000 eviction decisions and record: (a) wrong evictions (blocks evicted that were subsequently accessed), and (b) the LRU-to-Oracle gap closed.

RoleKV-Hybrid closes >99.9% of the LRU-to-Oracle gap at every budget ≥20%, the marginal 2.6 pp over base RoleKV (97.3% → 99.9%) recovers cases where the decay prior is out-of-distribution (e.g., unusually long reasoning traces). Cost: one extra attention-map read per forward pass (<1 ms on MI300X)—acceptable for offline or latency-tolerant workloads.

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

This appendix expands the Role Tagger (§4.1) and its robustness ablation (§6.7): the full per-template marker map, the deterministic ground-truth protocol, the row-normalized confusion matrix on HotpotQA-Llama-8B (n=8,743 blocks, 150 episodes), and engine-side lifecycle hooks. SWE-Bench-Verified function-calling accuracy is in progress and will appear in camera-ready (§8).

G.1 Marker Map per Chat Template

The tagger is parameterized by a single template-specific table: the marker token-id sequences that trigger role transitions. Table 13 gives the full map for the three chat templates we currently support. Adding a new template requires one row.

G.2 Reference Implementation

The per-turn FSM loop is given in Algorithm 1 (§4.1), here we record the engineering details that did not fit in the main paper.

Complexity and memory. One integer comparison per token, < 0.1 ms/turn on host CPU, one byte

Table 11: **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 12: **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%

of metadata per KV block (5 roles fit in 3 bits, but we round up for fast bit-pack access).

Marker matching on token ids, not strings. Markers are stored and matched as token-id sequences, not decoded strings: ids are stable under BPE merges, whereas substring matching on decoded text is brittle (e.g., `Thought:` can BPE-split into different token sequences depending on the preceding whitespace). Markers shorter than 4 tokens are matched via a sparse hash table, longer markers use a small Aho–Corasick automaton compiled once per template. Swapping the model family (e.g., Llama-3→Qwen-3) requires regenerating the marker token-id table (one Python function call), Algorithm 1 is unchanged.

Subtype extraction. Subtypes f are extracted by a single-pass scanner: for ReAct, the alphanumeric run after `Action:`, for function-calling, the `name` field inside `<tool_call>` (a one-state JSON-key scanner, $< 10\,\mu$s/action). They are stored as auxiliary string-ids and consumed only by the action-prediction boost (App. F), the priority sort is unchanged when the boost is disabled.

G.3 Robustness Methodology

Ground truth. Ground-truth tags come from parsing the agent-harness JSON and re-aligning role spans to vLLM's 16-token block grid under the same first-token-wins rule used at inference, assistant messages are split by a line-anchored regex over `Thought:`/`Action:`/`Observation:`/`Result:` (we include `Result:` because Llama-8B occasionally hallucinates in-line tool results). The procedure is fully deterministic - no LLM-as-judge, no human annotation - so any disagreement with the live tagger is a true error, not a labeling artifact. Tokenization uses the fast tokenizer's `return_offsets_mapping`, blocks falling entirely inside chat-template scaffolding are excluded.

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 14 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 in-line `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 §6.7.

G.4 Lifecycle and Engine Hook Points

This subsection addresses the engine-systems concerns that arise when the tagger has to coexist with real serving-engine machinery: block recycling and prefix-cache hits.

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

Table 13: **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
Positional (all templates):		
any	1st chat message	`system`
any	1st user message	`instruction`
ReAct:		
ReAct	`Thought:`	`reasoning`
ReAct	`Action:`	`action`
ReAct	`Observation:`	`tool_output`
Function-calling (Llama-3, Qwen-3):		
FC	pre-`<tool_call>`	`reasoning`
FC	inside `<tool_call>`	`action`
FC	inside `<tool_response>`	`tool_output`
Llama-3 chat header (all formats):		
Llama-3	`<|start_header_id|>system`	`system`
Llama-3	`<|start_header_id|>user`	`instruction`
Llama-3	`<|start_header_id|>assistant`	`reasoning` (default)
Subtype capture (optional):		
any	body of `action` block	`action[`f`]`
any	body of `tool_output` block	`tool_output[`f`]`

Table 14: **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

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