AGENTKV: PREDICTIVE ADMISSION PREVENTS KV-CACHE COLLAPSE IN AGENTIC LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon language-model agents place unique demands on LLM serving systems because they repeatedly return from tool calls with increasingly long contexts. At high concurrency, these growing contexts can evict one another's reusable KV cache, forcing repeated prefill, slowing completion, and keeping even more contexts resident. We call this positive-feedback failure *KV-cache collapse*. We show that the onset of collapse can be predicted from the relationship between available KV capacity and the average context demand of each instance. We introduce AGENTKV, a predictive admission controller that forecasts short-horizon context growth, maintains a lifecycle-safe working set under an adaptive high-bandwidth memory (HBM) budget, and replans from live cache telemetry. Across ten Qwen3-8B and Qwen3.8-27B campaigns on four RTX PRO 6000 GPUs, the predicted capacity boundary closely matches the observed transition from stable prefix reuse to cache collapse. Beyond this boundary, AGENTKV outperforms vLLM, ThunderAgent, InferCept, Continuum-TTL, and cache-aware vLLM-router in every measured cell, reaching up to $3.7\times$ the throughput of stock vLLM. At 240 concurrent instances, AGENTKV retains roughly 75% prefix reuse for Qwen3-8B workloads while stock vLLM falls to at most 0.4%, and removing predictive admission leaves only 27–77% of full-system throughput. These results show that predictive working-set control can prevent a high-load cache-collapse regime that allocation, routing, and persistence alone do not fully address. Our code is available at https://anonymous.4open.science/r/AgentKV-F6D2/

1 INTRODUCTION

Large language model agents are increasingly used as the reasoning core of downstream applications such as coding, research, and tool use. The shift from single model calls to long-horizon rollouts changes the pattern of LLM serving: a single model call is stateless and ends when its response is returned, whereas an agent rollout alternates between model calls and environment steps, so each call extends the prefix produced by the previous one. Serving an agent efficiently therefore requires the engine to treat the rollout, rather than the individual call, as the unit it maintains state for, since the prefix that each call rebuilds grows with the length of the rollout. Beyond the KV state of a single request, the engine must keep a rollout's state resident across environment steps and route the next call to where that state lives.

This pattern creates a failure mode that is easy to miss from GPU utilization alone. When many agent instances share the same GPU KV-cache capacity, their growing contexts begin to evict one another's reusable KV blocks. An agent resuming inference after a tool call can then no longer reuse its cached prefix efficiently and must repeat much of the prefill. Repeated prefill slows instance completion; slower completion keeps more contexts alive for longer; and the larger resident population causes still more eviction. The GPU may remain fully utilized while useful progress drops sharply. We call this positive-feedback failure *KV-cache collapse*.

Existing serving systems provide increasingly sophisticated mechanisms for managing KV state. Modern engines support efficient allocation, preemption, migration, routing, and cache persistence, while agent-aware schedulers can pause and resume persistent agent instances (Kwon et al., 2023; Abhyankar et al., 2024; Kang et al., 2026; Chen et al., 2026). Agent-aware schedulers also regulate concurrency using program state or cache feedback. We focus on admission under projected context

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

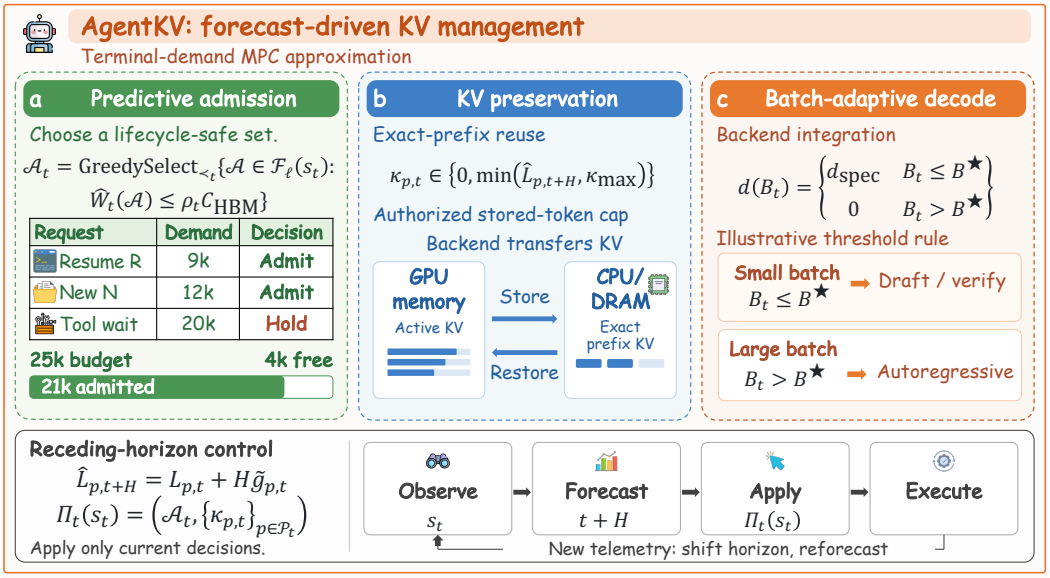

Figure 1: End-to-end AGENTKV pipeline. The controller forecasts future context demand, selects a lifecycle-safe working set under an adaptive HBM budget, and optionally preserves selected KV state in CPU memory. Fresh cache telemetry then closes the receding-horizon control loop. The rightmost batch-adaptive decode block is an optional backend integration point; the reported main configurations use autoregressive decoding.

growth: *how many growing agent instances can advance together while their projected KV demand remains within capacity?*

Current cache occupancy alone is not sufficient to answer this question. A working set that fits in HBM now may exceed capacity after the next few agent steps, so reacting only after memory pressure appears may be too late to prevent prefix eviction and repeated prefill. More generally, whether a working set can remain resident depends on the relationship between available KV capacity C_{HBM} and per-instance context demand. As aggregate demand approaches capacity, admitting additional instances becomes increasingly likely to destroy reusable state. Preventing collapse therefore requires controlling the *future* working set rather than maximizing instantaneous concurrency.

We introduce AGENTKV, a predictive admission controller for agentic LLM serving. At each control epoch, AGENTKV observes the lifecycle of each instance and the state of the serving backend, forecasts how individual contexts will grow over a short horizon, and selects a lifecycle-safe working set whose projected demand fits within an HBM target. Only instances in this working set are allowed to issue their next model requests. The controller applies the current decision, observes fresh cache telemetry, and continuously replans as contexts and system conditions change.

The HBM target itself is adaptive. Rising uncached-prompt traffic or falling prefix reuse indicates that the current resident set is becoming unstable, so AGENTKV tightens its target and reduces the number of simultaneously advancing instances. When cache behavior recovers, the target is relaxed gradually. This feedback mechanism allows the controller to respond to workload and backend conditions without relying on a fixed concurrency limit.

CPU KV persistence provides a complementary mechanism (Yu et al., 2025; Qin et al., 2025; Liu et al., 2025; Xia et al., 2026). Rather than treating offloading as an independent policy, AGENTKV uses the same context forecast to decide when preserving an instance's KV state outside HBM is worthwhile. Admission and persistence therefore address two different parts of the same problem: admission controls which instances form the active working set, while persistence determines whether selected reusable state should be retained outside GPU memory when residency cannot be maintained.

Our contributions are:

1. **A characterization of KV-cache collapse in agentic serving.** We identify a positive-feedback failure in which growing agent contexts cause prefix eviction, repeated prefill, slower completion, longer context residency, and further eviction. We characterize the operating boundary through the relationship between aggregate context demand and available KV capacity.
2. **Predictive working-set admission.** We formulate agentic serving as a dynamic working-set control problem and introduce AGENTKV, which forecasts short-horizon context growth and admits only a lifecycle-safe set of instances whose projected demand fits within GPU KV capacity.
3. **Cache-aware closed-loop control.** AGENTKV continuously replans from live cache telemetry, tightening its HBM target when recomputation pressure rises and coordinating admission with selective CPU KV persistence. The resulting controller is lightweight, deterministic, and directly compatible with existing LLM serving backends.

2 RELATED WORK

2.1 SERVING AND AGENT SCHEDULING

Orca introduced iteration-level scheduling (Yu et al., 2022), PagedAttention made dynamic KV allocation practical (Kwon et al., 2023), and FastServe added token-granular preemption (Wu et al., 2023). DeepSpeed-FastGen and Sarathi-Serve compose prefill and decode work (Holmes et al., 2024; Agrawal et al., 2024); DistServe, Splitwise, and TetriInfer disaggregate those phases (Zhong et al., 2024; Patel et al., 2024; Hu et al., 2024); and Llumnix live-migrates requests across instances (Sun et al., 2024). These systems optimize requests already at the engine. AGENTKV instead controls which persistent agent instances may issue their next request.

Application-aware systems expose higher-level structure: Parrot represents dataflow (Lin et al., 2024), and InferCept manages state across tool and human interruptions (Abhyankar et al., 2024). ThunderAgent, CONCUR, and Agentix (formerly Autellix) schedule persistent agent instances (Kang et al., 2026; Chen et al., 2026; Luo et al., 2026); Continuum manages cache lifetimes (Li et al., 2025), ConServe places conversations using observed state (Ding et al., 2026), and TokenCake and MORI coordinate agent state with tiered KV movement (Bian et al., 2025; Xia et al., 2026). AGENTKV's distinction is admission against projected *next* contexts, cache-miss feedback, and a measured prediction of when that distinction pays.

2.2 KV REUSE, TIERING, AND DEMAND REDUCTION

RadixAttention, Prompt Cache, and ChunkAttention reuse prompt segments (Zheng et al., 2024; Gim et al., 2024; Ye et al., 2024), while Preble co-optimizes distributed reuse and load balance (Srivatsa et al., 2025). Pensieve caches multi-turn history across GPU and CPU (Yu et al., 2025); Mooncake disaggregates KV-centric serving (Qin et al., 2025); LMCache provides a cross-engine cache (Liu et al., 2025); and CacheGen compresses KV for transfer (Liu et al., 2024a). These supply storage, routing, or movement. AGENTKV is complementary: it chooses the resident instance set and delegates exact storage and restoration to the backend.

Other work raises effective capacity. vAttention changes physical allocation (Prabhu et al., 2025); H_2O, SnapKV, and PyramidKV retain selected state (Zhang et al., 2023; Li et al., 2024; Cai et al., 2025); KIVI quantizes it (Liu et al., 2024b); and InfiniGen selectively prefetches host KV (Lee et al., 2024). These methods move the collapse boundary but do not remove admission once aggregate demand exceeds it. Speculative and suffix decoding target serial decode cost instead of residency (Leviathan et al., 2023; Oliaro et al., 2025).

3 PROBLEM FORMULATION

3.1 SERVING STATE AND CONTROL INTERFACES

Let $\mathcal{P}_t$ denote the live persistent agent instances at control epoch t. Lifecycle and execution mode are distinct: an instance is active or paused, and an active instance is either executing a model call

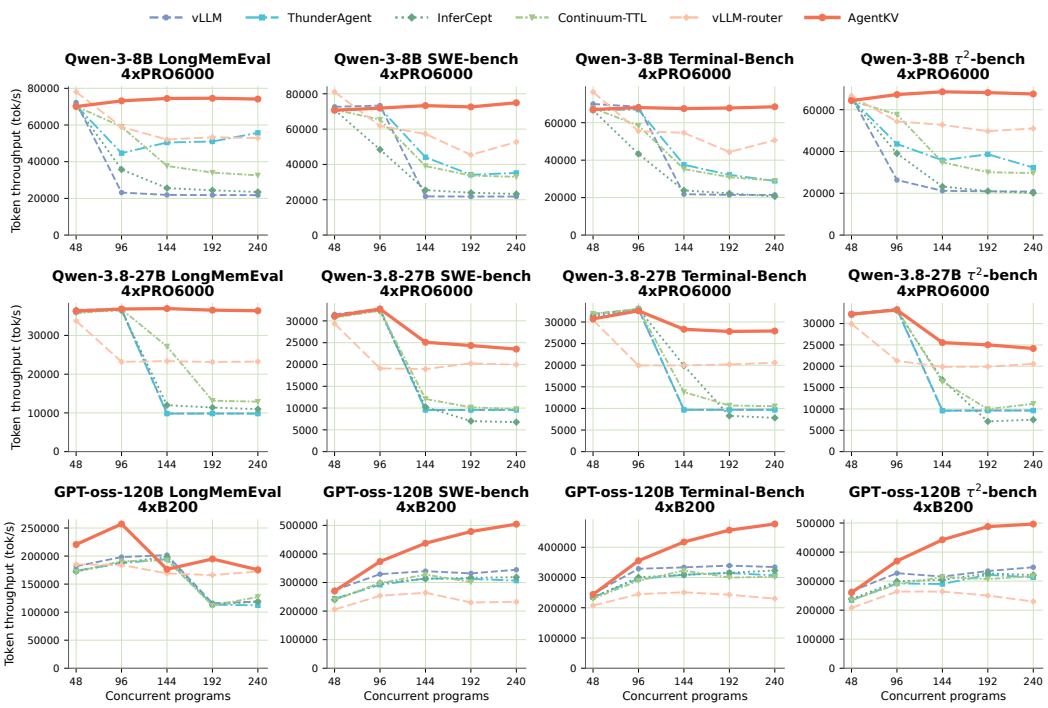

Figure 2: Load sweeps on four RTX PRO 6000 GPUs (top two rows) and four B200 GPUs (bottom row): aggregate token throughput against concurrent instances, measured as input plus output tokens divided by active campaign seconds. The PRO 6000 rows show Qwen3-8B and Qwen3.8-27B; the B200 row shows GPT-oss-120B. Each row contains LongMemEval and the SWE-bench, Terminal-Bench, and τ^2-bench serving replays. On B200, "AgentKV" is the four-replica deployment; the old tensor-parallel AgentKV and ThunderAgent-replicas series are omitted.

or waiting on a tool. AGENTKV maintains the per-instance state

$$x_{p,t} = \left(L_{p,t}, \bar{g}_{p,t}, n^g_{p,t}, \bar{r}_{p,t}, \ell_{p,t}, m_{p,t}, j_{p,t}, e_{p,t}\right), \tag{1}$$

where $L_{p,t}$ is the latest total-token count, $\bar{g}_{p,t}$ and $n^g_{p,t}$ are the EMA and sample count for context growth, $\bar{r}_{p,t}$ is the EMA of time between requests, $\ell_{p,t}$ is the lifecycle state, $m_{p,t}$ is the execution mode, $j_{p,t}$ is the agent-step index used to distinguish initial requests from later agent steps, and $e_{p,t}$ indicates a just-resumed instance. The backend and feedback state is

$$y_t = (C_{\mathrm{HBM}}, C_{\mathrm{DRAM}}, U_t, R_t, q^u_t, q^p_t, h_t, T_t, \beta_t, \rho_t), \tag{2}$$

where C_{HBM} and C_{DRAM} are the configured GPU and CPU KV capacities, U_t is GPU KV occupancy, and R_t is vLLM's inferred shared-token count. The recent uncached-prompt rate, prefix-query rate, and prefix-hit rate are q^u_t, q^p_t, and h_t; T_t is the available telemetry history; β_t is CPU KV transfer occupancy; and ρ_t is the current HBM target. All quantities come from the proxy, controller state, or vLLM telemetry.

At their respective decision points, the two AGENTKV policy outputs are

$$\Pi_t(s_t) = (\mathcal{A}_t, \{\kappa_{p,t}\}_{p\in\mathcal{P}_t}). \tag{3}$$

Here $\mathcal{A}_t$ is the logical working set permitted to issue model requests and $\kappa_{p,t}$ is a per-request CPU-persistence cap. The outputs are event driven rather than simultaneous: admission is updated at a periodic control epoch and persistence is gated before each request. Together they define the AGENTKV runtime policy, while vLLM retains responsibility for physical KV-block placement, storage, and restoration.

3.2 COORDINATED RECEDING-HORIZON POLICY

Future tool duration, output length, and context growth are unknown. Rather than roll out a sequence of future actions, AGENTKV predicts each instance's demand at the end of an H-step horizon and

selects only the current working set. Let $\widehat{W}_t(\mathcal{A})$ be that terminal-demand estimate and let $\mathcal{F}_\ell(s_t)$ contain sets permitted by the lifecycle state. The implemented decision has the form

$$\mathcal{A}_t = \mathsf{GreedySelect}_{\prec_t}\left\{\mathcal{A} \in \mathcal{F}_\ell(s_t) : \widehat{W}_t(\mathcal{A}) \le \rho_t C_{\mathrm{HBM}}\right\}, \tag{4}$$

where $\prec_t$ orders lifecycle classes first and projected context size second. The deterministic selector retains or admits as many instances as possible within each priority class, preferring shorter projected contexts.

For every admitted request, AGENTKV derives $\kappa_{p,t}$ from the same forecast together with observed HBM and transfer pressure. The two outputs are coupled: admission controls projected HBM demand, and persistence trades future recomputation against CPU transfer work for the instances admission has chosen to hold.

After execution, the controller observes fresh completions and telemetry and repeats the policy. Its admission stage is a terminal-demand, receding-horizon approximation to model-predictive control: the horizon enters the context forecast, but the implementation neither rolls out an H-action sequence nor fits stage and terminal costs. CPU persistence uses an event-specific rule within the same closed-loop AGENTKV policy.

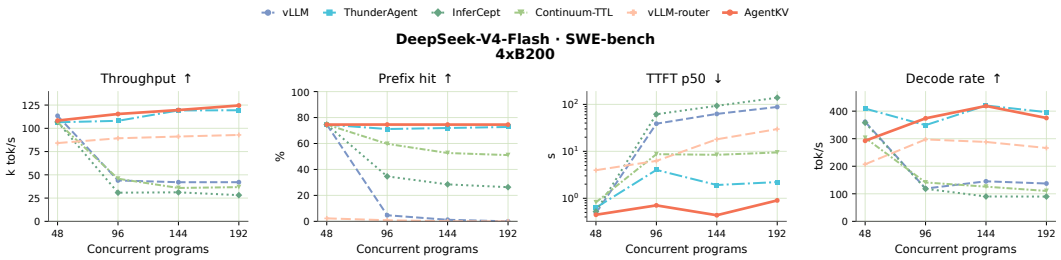

Figure 3: DeepSeek-V4-Flash on the SWE-bench serving replay at 48, 96, 144, and 192 concurrent instances on four B200 GPUs. The four panels report aggregate token throughput, prefix-cache hit rate, median TTFT, and decode rate. We select one AgentKV deployment before plotting: the tensor-parallel configuration has the higher mean aggregate token throughput across this sweep (117.0k versus 110.4k tokens/s for four replicas), and the same configuration is used in every panel.

4 METHODOLOGY

4.1 AGENTKV CONTROL PIPELINE

Every control period, AGENTKV updates per-instance growth statistics, forecasts horizon-end context demand, updates the HBM target from cache telemetry, and enforces the resulting admission budget. Before each admitted model request, it uses the forecast, current occupancy, predicted reuse interval, and recent transfer occupancy to set the request's CPU KV persistence cap. Execution produces new token counts, cache telemetry and transfer counters, closing the loop for the next decision (Figure 1). Appendix Algorithm 1 consolidates these periodic and request-time decisions into one executable control loop.

Lifecycle safety precedes capacity: model calls are never interrupted, while instances waiting on tools can be paused immediately. Within each lifecycle class, projected context size provides the deterministic tie-breaking order. The following subsections describe the forecast, admission and persistence stages of this pipeline.

4.2 SHORT-HORIZON CONTEXT FORECAST

After each completed model call, let $\Delta L_{p,t} = [L_{p,t} - L_{p,t-1}]_+$. Once the growth estimator has been initialized, AGENTKV updates

$$\bar{g}_{p,t} = \alpha_g \Delta L_{p,t} + (1 - \alpha_g)\bar{g}_{p,t-1}. \tag{5}$$

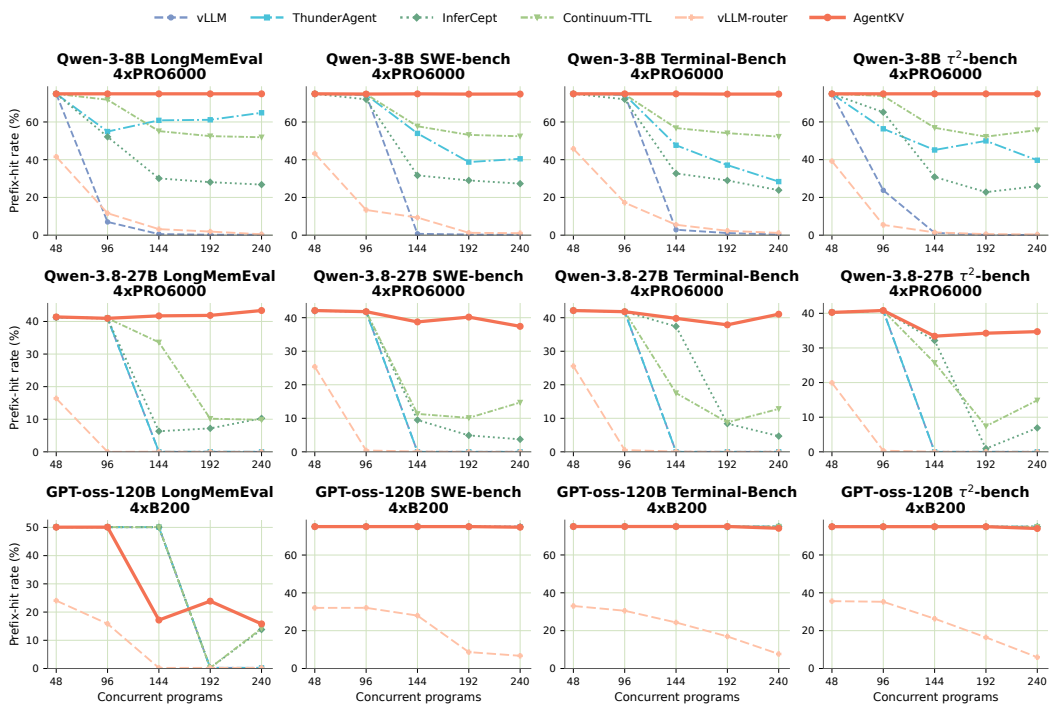

Figure 4: Engine-reported prefix-cache hit rate for the same twelve panels as Figure 2: eight PRO 6000 panels followed by four GPT-oss-120B B200 panels. The B200 naming and exclusions are identical to Figure 2.

The first observed increment initializes the EMA directly. To avoid treating an unobserved instance as having zero growth, the effective estimate is

$$\widetilde{g}_{p,t} = \begin{cases} g_0, & n^g_{p,t} < n_g, \\ \bar{g}_{p,t}, & n^g_{p,t} \geq n_g, \end{cases} \tag{6}$$

where g_0 is the warmup estimate and n_g is the required sample count. The H-step context forecast is

$$\widehat{L}_{p,t+H} = L_{p,t} + H\widetilde{g}_{p,t}. \tag{7}$$

For a candidate working set $\mathcal{A}$, projected backend demand is

$$\widehat{W}_t(\mathcal{A}) = \sum_{p \in \mathcal{A}} \widehat{L}_{p,t+H} + b|\mathcal{A}|, \tag{8}$$

where b is a per-instance safety buffer. The forecast does not subtract vLLM's inferred shared-token counter R_t: that counter cannot distinguish resident prefix sharing from blocks moved to CPU, so crediting it could treat offloaded state as free HBM. Equation 8 is intentionally a short-term memory-demand model; it does not predict task success, remaining agent steps, or future tool outputs.

4.3 Feedback Target and Admission

Admission applies Equation 4 at a target fraction ρ_t of GPU KV capacity. For live population $N_t = |\mathcal{P}_t|$, the base target is

$$\rho^{\text{base}}(N_t) = \begin{cases} \rho_{\text{low}}, & N_t \leq N_0, \\ \rho_{\text{high}}, & N_t > N_0, \end{cases} \tag{9}$$

where N_0 separates the low- and high-load regimes. Every Δ_{ctrl}, the controller derives a recomputation-pressure indicator from its recent telemetry window:

$$c_t = \mathbb{I}[T_t \geq T_{\text{hist}}]\, \mathbb{I}[q^u_t \geq q_{\text{miss}} \ \vee \ (q^p_t \geq q_{\text{miss}} \wedge h_t \leq h_{\min})]\,. \tag{10}$$

The first branch detects high uncached-prompt traffic. The second requires both substantial prefix-query traffic and a low prefix-hit rate, avoiding a low-hit alarm when almost no reusable prefix tokens were requested. Define

$$\rho^{\mathrm{em}}(N_t) = \max\big(\rho_{\min}, \rho^{\mathrm{base}}(N_t) - \Delta_\rho\big)\,. \tag{11}$$

The hysteretic target update is

$$\rho_{t+1} = \begin{cases} \min(\rho_t, \rho^{\mathrm{em}}(N_t))\,, & c_t = 1, \\ \min\big(\rho^{\mathrm{base}}(N_t), \rho_t + \eta_\rho\big)\,, & c_t = 0. \end{cases} \tag{12}$$

Thus pressure tightens the target immediately, while healthy windows relax it gradually toward the base target.

When projected demand exceeds the budget, AGENTKV first pauses tool-running instances in descending projected-context order. If more demand must be removed, in-flight model calls are marked in the same order and paused when they next reach a tool boundary. Because these calls cannot be preempted, the logical post-boundary set can satisfy Equation 4 while instantaneous physical occupancy remains temporarily above the target; the implementation does not introduce or optimize a slack variable.

Resume proceeds by lifecycle priority: previously active reasoning work, then new instances, then other tool-running instances. Within each group, AGENTKV tries the shortest projected contexts first and admits each one only if the resulting working set fits. Sorting dominates the deterministic routine, for $O(N \log N)$ control time.

4.4 PREDICTIVE SELECTIVE CPU KV PERSISTENCE

AGENTKV uses vLLM's native request field, `kv_transfer_params.max_offload_tokens`, to set a persistence cap before every admitted request. The interface also exposes the two static policies used in our ablation:

$$\text{No persistence}: \quad \kappa_{p,t} = 0, \tag{13}$$

$$\text{Persist all}: \quad \kappa_{p,t} = \kappa_{\mathrm{unbounded}}, \tag{14}$$

$$\text{Predictive persistence}: \quad \kappa_{p,t} \in \Big\{0, \min(\widehat{L}_{p,t+H}, \kappa_{\max})\Big\}\,. \tag{15}$$

Here "persist all" means that the request does not impose a practical token cap. The configured vLLM connector enforces the capacity of the CPU tier; AGENTKV neither reserves aggregate CPU space nor solves a C_{DRAM}-constrained packing problem.

Predictive persistence combines forecast and telemetry into an effective pressure score,

$$\pi_{p,t} = \frac{\max(\widehat{W}_t, U_t C_{\mathrm{HBM}})}{C_{\mathrm{HBM}}} + \lambda_r \phi_r(\bar{r}_{p,t}; T_r) + \lambda_e\, \mathbb{I}[e_{p,t}], \tag{16}$$

where $\widehat{W}_t$ denotes projected demand for the current admitted set, U_t is engine-reported GPU KV utilization, $\bar{r}_{p,t}$ is an EMA of time between requests, $e_{p,t}$ indicates a just-resumed instance, and $\phi_r(r; T_r) = \min(r/T_r, 1)$. Let β_t be the fraction of the recent window occupied by CPU KV loads and stores. The policy authorizes persistence only when

$$L_{p,t} \geq L_{\min}, \qquad \pi_{p,t} \geq \pi_{\min}, \qquad \beta_t < \beta_{\max}, \tag{17}$$

and the request is not the final agent step. Otherwise it supplies a zero cap. The inter-request EMA uses coefficient α_r.

The cap authorizes what may be *stored*; it does not assert that a store occurred. On a later matching request, vLLM's connector decides whether stored blocks can be restored. AGENTKV neither commands proactive GPU eviction nor initiates CPU-to-HBM migration. Persistence is a secondary action: it can only add reuse that admission has not already secured, so its value depends on how much of the selected working set HBM can hold (Figure 6).

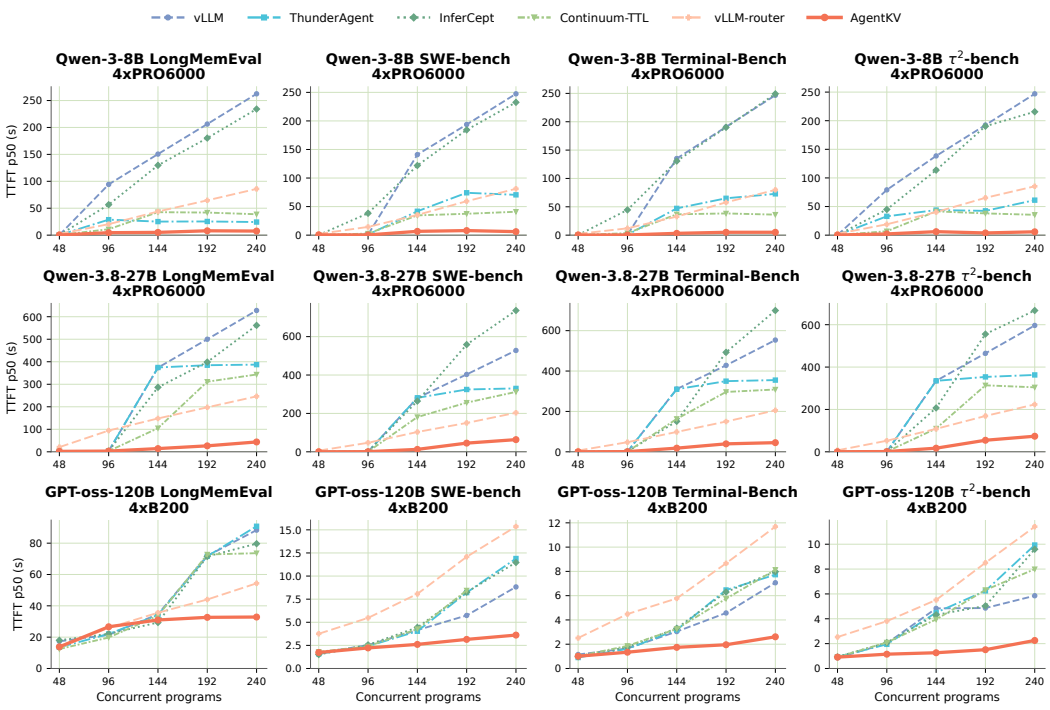

Figure 5: Median time to first token (TTFT p50; lower is better) using the same twelve-panel load-sweep layout as Figures 2 and 4: eight PRO 6000 panels followed by four GPT-oss-120B B200 panels. The B200 naming and exclusions are identical to Figure 2.

5 Experiments

5.1 Experimental setup

Platforms and models. The evaluation uses RTX PRO 6000 GPUs (96 GiB each) and B200 GPUs, running Qwen3-8B, Qwen3.8-27B, GPT-oss-120B, and DeepSeek-V4-Flash. Main load sweeps use four GPUs per platform: either one tensor-parallel engine (TP4) or four single-GPU replicas (4×TP1). A separate Qwen3-8B experiment compares one and four PRO 6000 GPUs. All runs use vLLM 0.26.0, prefix caching, greedy decoding, thinking disabled, and a 90% GPU-memory utilization limit. Qwen models use a 40,960-token context; GPT-oss-120B and DeepSeek-V4-Flash use 65,536 tokens, with `fp8_ds_mla` KV for DeepSeek. Table 1 gives the model and platform settings.

Baselines. We compare against stock vLLM, ThunderAgent, proxy-layer implementations of the InferCept and Continuum-TTL policies, and cache-aware vLLM-router. Each deployment uses its configured KV capacity. For example, vLLM-router runs four single-GPU replicas; on PRO 6000, replicated weights leave about 14% less total KV capacity than TP4. Its results therefore reflect both routing and deployment topology. Appendix E describes these differences and the scope of the policy reimplementations.

Workloads and measurements. We use a 32k-token adaptation of LongMemEval-S (Wu et al., 2025), LOCOMO (Maharana et al., 2024), and serving replays of SWE-bench Verified (Jimenez et al., 2024; Chowdhury et al., 2024), Terminal-Bench (Merrill et al., 2026), and τ^2-bench (Barres et al., 2026). Each instance normally makes four model calls with a growing context. The replays combine benchmark task text with deterministic tool observations to reach 24–25k tokens. They measure serving performance rather than benchmark resolve or pass rate; construction details and timeout reruns are in Appendix D.

Every run begins with a verified empty prefix cache and a fixed number of planned calls. Throughput is total input plus output tokens divided by active campaign time; prefix reuse comes from vLLM's prompt-token statistics, and latency is measured over successful calls. We report only runs passing the checks in Appendix B. Most settings have one validated run, so ratios describe those observations. The main sweeps have no CPU KV tier; a separate persistence experiment adds 256 GiB of CPU storage.

5.2 Performance across workloads and serving configurations

Load sweeps on both four-GPU hosts (Figures 2 and 4) cover 48 to 240 concurrent instances. At 240 instances on PRO 6000, AGENTKV achieves 1.17–$1.42\times$ the throughput of the best other method on the three serving replays. On the same replays using B200 with GPT-oss-120B, it achieves 1.08–$1.46\times$ the best other method from 96 to 240 instances, reaching 477–504k tokens/s at 240. The benefit depends on the workload: GPT-oss-120B LongMemEval ranges from 0.87 to $1.30\times$ the best alternative. The B200 AGENTKV curves in these sweeps use four single-GPU replicas.

At 48 instances, AGENTKV is within 2% of the best other method on the B200 replays. In the PRO 6000 sweeps, methods using TP4 perform similarly below the capacity boundary; the cache-aware router is fastest there, while AGENTKV pays a small proxy-hop overhead. For the ten PRO 6000 combinations of two models and five workloads, $C_{\mathrm{HBM}}/\bar{c}$ predicts the point where the aggregate context exceeds capacity. That estimate and the first measured load at which AGENTKV leads by more than 5% agree within one 48-instance step. The replay thresholds are 95–96 instances for Qwen3-8B and 146–156 for Qwen3.8-27B.

At 240 instances, stock vLLM retains only 0.0–0.4% reuse across the four plotted PRO 6000 workloads. AGENTKV retains 74.8–74.9% with Qwen3-8B and 34.7–43.4% with Qwen3.8-27B; on the three replays, the corresponding ranges are approximately 75% and 35–41%. On the B200 replays, AGENTKV retains 74% reuse at the same load. Preserving these prefixes reduces repeated prefill as concurrency increases.

For Qwen3-8B LongMemEval at 192 instances, moving from one PRO 6000 GPU (TP1) to four (TP4) raises throughput by $1.86\times$ for AGENTKV and $1.69\times$ for stock vLLM, with the planned work held fixed. AGENTKV outperforms stock vLLM at both sizes: added capacity improves throughput, while controlling the active working set remains beneficial.

With DeepSeek-V4-Flash on four B200 GPUs, AGENTKV uses TP4 and exceeds the best other method's throughput at 96–192 instances on the SWE-bench replay, while retaining 74.6% prefix reuse (Figure 3). TP4 is used throughout the sweep because its mean throughput across the sweep exceeded that of the four-replica alternative.

5.3 Latency and completion time

Preserving reuse also shortens the wait for the first token (Figure 5). At 240 instances on PRO 6000, AGENTKV reduces median TTFT from 246.8–262.5 s to 5.1–7.6 s with Qwen3-8B, and from 528.0–627.7 s to 43.8–73.6 s with Qwen3.8-27B, compared with stock vLLM. On the B200 DeepSeek-V4-Flash replay, median TTFT stays below one second through 192 instances.

Faster completion comes with latency tradeoffs. On LongMemEval at 192 instances, AGENTKV reduces completion time relative to stock vLLM by 71% for Qwen3-8B and 61% for Qwen3.8-27B on PRO 6000. Request-latency p95 falls by 17% and 15%, respectively, but inter-token latency p95 rises by 11% and 13%. At the same load on B200, the four-replica AGENTKV deployment cuts LongMemEval completion time by 19% for Qwen3.8-27B and 41% for GPT-oss-120B relative to stock vLLM, again with tradeoffs among latency metrics. Appendix Figure 7 reports the full set of measurements. Appendix B.1 reports the component ablations.

6 Conclusion

AGENTKV uses context-growth forecasts and cache feedback to control the active working set. Across the evaluated campaigns, predictive admission preserves prefix reuse and throughput beyond the capacity boundary; persistence adds little when the working set remains resident.

AI USE STATEMENT

In this work, generative AI tools were employed to support method implementation and translation. We also used these tools to assist with creating and modifying software code, summarizing and analyzing existing literature, and retrieving or searching for relevant information. Generative AI tools were not used to generate or refine research hypotheses, design or evaluate research methodologies or experiments, clean or restructure datasets, conduct qualitative or thematic data analysis, generate synthetic data, develop theoretical models or conceptual frameworks. These applications were not relevant to the present work. All code, written content, and references were independently reviewed. The authors take full responsibility for the final content of this work, including all text, claims, and artifacts produced with the assistance of generative AI tools.

ETHICS STATEMENT

This paper primarily focuses on KV-cache management for agentic LLM serving. There are several potential societal consequences of our work, none of which we feel must be specifically highlighted here.

REPRODUCIBILITY STATEMENT

The system developed in this work is publicly accessible online, and references are provided for all baseline methods considered in our experiments. Further technical details regarding the implementation and experimental setup are provided in the Appendix.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. InferCept: Efficient intercept support for augmented large language model inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 81–95. PMLR, 2024. URL https://proceedings.mlr.press/v235/abhyankar24a.html.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2506.07982.

Zhuohang Bian, Feiyang Wu, Zhuoran Li, Teng Ma, and Youwei Zhuo. TokenCake: A KV-cache-centric serving framework for LLM-based multi-agent applications. *arXiv preprint arXiv:2510.18586*, 2025. doi: 10.48550/arXiv.2510.18586. URL https://arxiv.org/abs/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. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2406.02069.

Qiaoling Chen, Zhisheng Ye, Tian Tang, Peng Sun, Boyu Tian, Guoteng Wang, Shenggui Li, Yonggang Wen, Zhenhua Han, and Tianwei Zhang. CONCUR: High-throughput agentic batch inference of LLM via congestion-based concurrency control. *arXiv preprint arXiv:2601.22705*, 2026. doi: 10.48550/arXiv.2601.22705. URL https://arxiv.org/abs/2601.22705.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho,

Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench verified. OpenAI, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Jianru Ding, Ryien Hosseini, Pouya Mahdi Gholami, Mingyuan Xiang, and Henry Hoffmann. Observation, not prediction: Conversation-level disaggregated scheduling for agentic serving. *arXiv preprint arXiv:2606.01839*, 2026. doi: 10.48550/arXiv.2606.01839. URL https://arxiv.org/abs/2606.01839.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. DeepSpeed-FastGen: High-throughput text generation for LLMs via MII and DeepSpeed-Inference. *arXiv preprint arXiv:2401.08671*, 2024. doi: 10.48550/arXiv.2401.08671. URL https://arxiv.org/abs/2401.08671.

Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. Inference without interference: Disaggregate LLM inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024. URL https://arxiv.org/abs/2401.11181.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Hao Kang, Ziyang Li, Weili Xu, Xinyu Yang, Yinfang Chen, Junxiong Wang, Beidi Chen, Tushar Krishna, Chenfeng Xu, and Simran Arora. ThunderAgent: A simple, fast and program-aware agentic inference system. *arXiv preprint arXiv:2602.13692*, 2026. doi: 10.48550/arXiv.2602.13692. URL https://arxiv.org/abs/2602.13692.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Association for Computing Machinery, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lee.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025. doi: 10.48550/arXiv.2511.02230. URL https://arxiv.org/abs/2511.02230.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-0722. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*. Association for Computing Machinery, 2024a. doi: 10.1145/3651890.3672274. URL https://doi.org/10.1145/3651890.3672274.

Yuhan Liu, Yihua Cheng, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Rui Zhang, Kuntai Du, and Junchen Jiang. LMCache: An efficient KV cache layer for enterprise-scale LLM inference. *arXiv preprint arXiv:2510.09665*, 2025. doi: 10.48550/arXiv.2510.09665. URL https://arxiv.org/abs/2510.09665.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 2024b. URL https://proceedings.mlr.press/v235/liu24bz.html.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Agentix: An efficient serving engine for LLM agents as general programs. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 2443–2459. USENIX Association, 2026. URL https://www.usenix.org/conference/nsdi26/presentation/luo.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13851–13870. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.747. URL https://aclanthology.org/2024.acl-long.747/.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=a7Qa4CcHak.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4211. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b7aea253ab34a773967f1e4cdea9e4fb-Abstract-Conference.html.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024. URL https://www.microsoft.com/en-us/research/publication/splitwise-efficient-generative-llm-inference-using-phase-splitting/.

Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vAttention: Dynamic memory management for serving LLMs without PagedAttention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. Association for Computing Machinery, 2025. doi: 10.1145/3669940.3707256. URL https://doi.org/10.1145/3669940.3707256.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage*

Technologies (FAST 25), pp. 155–170. USENIX Association, 2025. URL https://www.usenix.org/conference/fast25/presentation/qin.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5bc342f48de8264779952fac378f96dc-Abstract-Conference.html.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 173–191. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/sun-biao.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023. doi: 10.48550/arXiv.2305.05920. URL https://arxiv.org/abs/2305.05920.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d813d324dbf0598bbdc9c8e79740ed01-Abstract-Conference.html.

Tian Xia, Hanchen Li, Zhifei Li, Xiaokun Chen, Hao Kang, Yifan Qiao, Yi Xu, and Ion Stoica. Idleness is relative: Exploiting tool-call idle windows for offloading in agentic systems with MORI. *arXiv preprint arXiv:2606.00866*, 2026. doi: 10.48550/arXiv.2606.00866. URL https://arxiv.org/abs/2606.00866.

Lu Ye, Ze Tao, Yong Huang, and Yang Li. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11608–11620. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.623. URL https://aclanthology.org/2024.acl-long.623/.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*. Association for Computing Machinery, 2025. doi: 10.1145/3689031.3696086. URL https://doi.org/10.1145/3689031.3696086.

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. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract.html.

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. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin.

A CONTROLLER ALGORITHM

Algorithm 1 makes explicit the event ordering omitted from the mathematical presentation. Admission is recomputed every control period, while the persistence gate is evaluated only when an admitted instance issues a model request. Pausing remains lifecycle safe: an active model call is allowed to finish and is stopped only at its next tool boundary.

Algorithm 1 AGENTKV receding-horizon admission and request-time KV persistence

Require: Live instances $\mathcal{P}_t$, states $\{x_{p,t}\}$, backend telemetry y_t, horizon H, buffer b, GPU capacity C_{HBM}
Ensure: Admitted set $\mathcal{A}_t$ and per-request caps $\kappa_{p,t}$

while the serving system is active **do**
  Observe completions, token counts, cache counters, occupancy U_t, and transfer occupancy β_t
  for each $p \in \mathcal{P}_t$ with a newly completed model call **do**
    Update $\bar{g}_{p,t}$ and its sample count using Equation 5
  end for
  for each $p \in \mathcal{P}_t$ **do**
    Compute $\widetilde{g}_{p,t}$ and $\widehat{L}_{p,t+H} = L_{p,t} + H\widetilde{g}_{p,t}$
  end for
  Compute collapse signal c_t and update target ρ_{t+1} using Equations 10 and 12
  $\mathcal{A}_t \leftarrow \mathsf{GreedySelect}_{\prec_t}\left(\mathcal{P}_t, \rho_{t+1}C_{\mathrm{HBM}}, \{\widehat{L}_{p,t+H} + b\}\right)$
  for each $p \in \mathcal{P}_t \setminus \mathcal{A}_t$ in decreasing projected size **do**
    if p is waiting on a tool **then**
      Pause p immediately
    else
      Mark p to pause after its active call reaches a tool boundary
    end if
  end for
  Retain or resume instances in $\mathcal{A}_t$ by lifecycle priority, then increasing projected size
  for each model request issued by p before the next control epoch **do**
    if $p \notin \mathcal{A}_t$ **then**
      Defer the request until p is admitted
    else
      Compute pressure $\pi_{p,t}$ using Equation 16
      if p is nonterminal, $L_{p,t} \geq L_{\min}$, $\pi_{p,t} \geq \pi_{\min}$, and $\beta_t < \beta_{\max}$ **then**
        $\kappa_{p,t} \leftarrow \min(\widehat{L}_{p,t+H}, \kappa_{\max})$
      else
        $\kappa_{p,t} \leftarrow 0$
      end if
      Attach $\kappa_{p,t}$ to the native KV-transfer request and submit
    end if
  end for
  Execute until the next Δ_{ctrl} epoch; set $t \leftarrow t + 1$
end while

B EXPERIMENTAL DETAIL

Controller configuration. The control period is 5 s, the forecast horizon is $H = 4$, the default growth estimate is $g_0 = 256$ tokens, and the per-instance buffer is $b = 100$ tokens. The growth EMA uses $\alpha_g = 0.3$ and switches from its default after $n_g = 1$ observed increment. Admission uses $N_0 = 24$ and low/high-load HBM targets of 1.00/0.88. A telemetry window becomes eligible after 10 s; the feedback rule uses an uncached-token threshold of 100,000 tokens/min, a prefix-hit threshold of 0.20, an emergency target drop of 0.10, a minimum target of 0.70, and a relaxation step of 0.01.

Predictive persistence uses $\alpha_r = 0.3$, requires at least 4,096 context tokens, caps a request at 40,960 tokens, and applies a pressure threshold of 0.8. Its reuse horizon is 10 s. The main sweeps run without a CPU KV tier; the dedicated persistence ablation attaches a 256 GiB tier and holds every other setting fixed.

Campaign protocols and measurement. Each load-sweep cell starts from a verified cold prefix cache and runs fixed work: concurrent instances multiplied by the campaign's recorded number of turns. Aggregate token throughput is total input plus output tokens divided by active campaign seconds. Prefix hit rate is taken from vLLM's prompt-token details; TTFT, TPOT, inter-token latency, request latency, prefill rate, and decode rate are computed over successful model calls. The plotting code does not substitute a partially completed cell.

Campaign validators check the live backend command, controller mode and KV policy, external cache reset, expected work, sample continuity, counter monotonicity, and failure rate. A cell is rejected if its cache reset fails, if failures exceed 1% of planned calls, or if fewer than 40% of planned calls complete. The focused ablation is stricter: its comparison load is the largest at which every plotted variant completes the common audit. All published rows retain their source summary path in the consolidated CSV.

B.1 Ablations

We change one control component at a time while keeping the engine fixed, using the highest load at which every variant passes the validation checks (Figure 6). Removing admission leaves 77% of full-system throughput on Qwen3-8B LongMemEval at 192 instances, 46% on Qwen3-8B Terminal-Bench at 192, and 27% on Qwen3.8-27B LongMemEval at 144. In contrast, a one-step horizon and disabling persistence each stay within 2% of the full system.

To test persistence with CPU storage available, we add a 256 GiB tier for Qwen3-8B LongMemEval at 192 instances. Selective persistence, no persistence, and persist-all achieve 74.4k, 74.7k, and 73.4k tokens/s, respectively, compared with 21.4k for stock vLLM. Once admission keeps the selected working set resident, extra persistence adds little in this setting.

C Additional figures

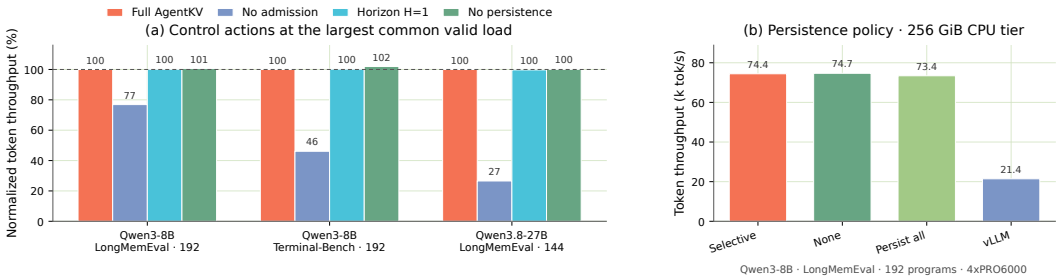

Figure 6: Focused PRO 6000 sensitivity study. (a) Aggregate token throughput normalized to full AgentKV at the largest common audit-valid load in each campaign. Removing predictive admission is the only material degradation; a one-step horizon and no persistence remain within 2%. (b) Aggregate token throughput with a 256 GiB CPU KV tier for Qwen3-8B LongMemEval at 192 instances. The three AgentKV persistence policies are nearly tied, while vLLM remains collapsed. These single audited runs support regime sensitivity.

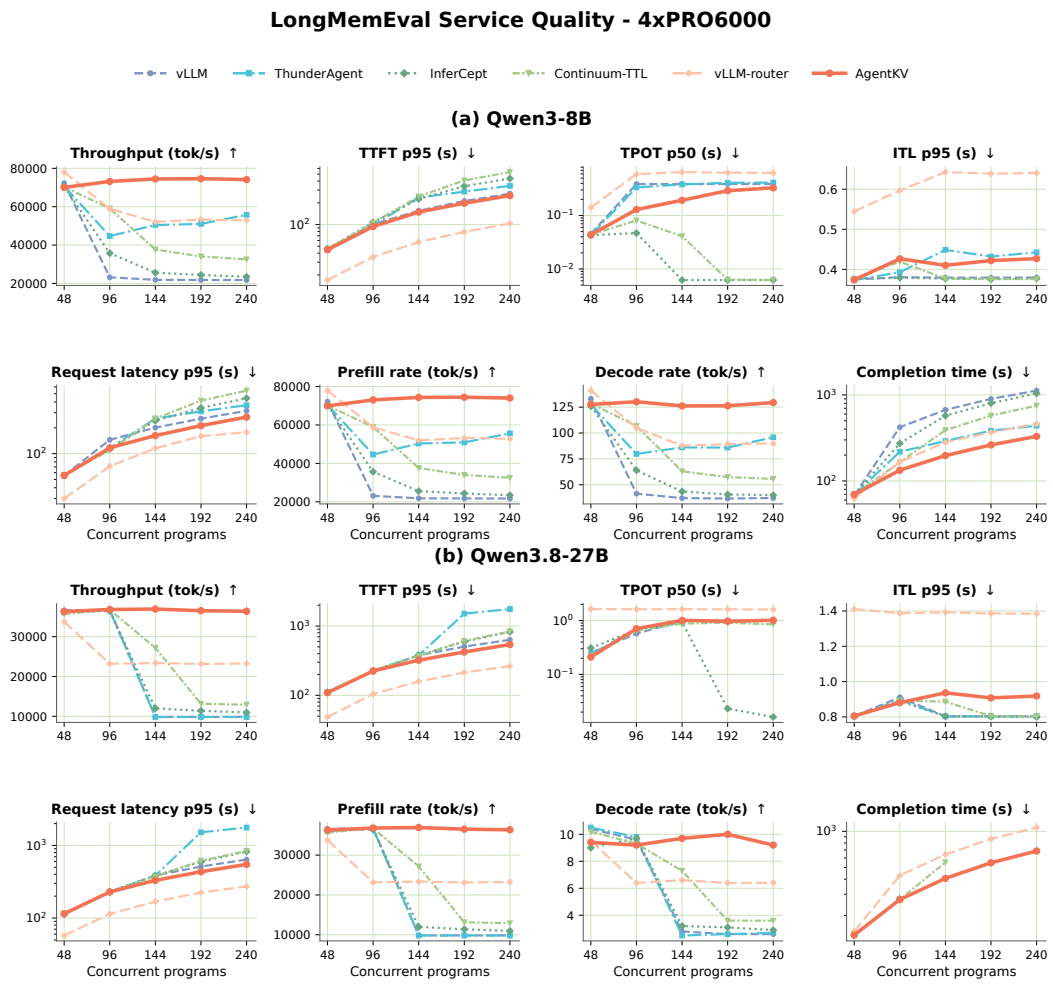

Figure 7: LongMemEval service quality on four RTX PRO 6000 GPUs for (a) Qwen3-8B and (b) Qwen3.8-27B. Each 2-by-4 grid reports aggregate token throughput, TTFT p95, TPOT p50, inter-token latency p95, request-latency p95, prefill and decode token rates, and completion time. Up and down arrows mark whether larger or smaller values are preferable. Figure 2 reports the same throughput metric across all workloads.

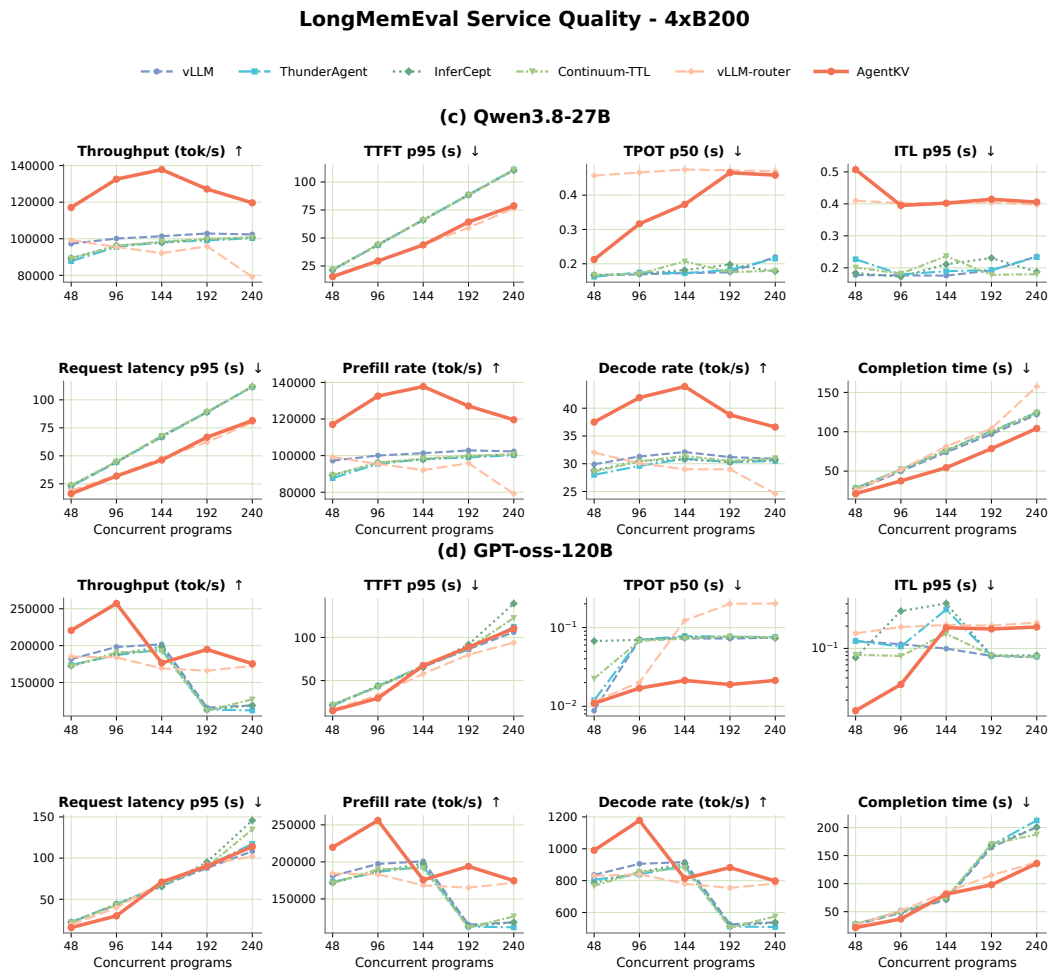

Figure 7: B200 continuation of Figure 7: the same eight LongMemEval metrics for (c) Qwen3.8-27B and (d) GPT-oss-120B using the post-stress campaign. In both B200 grids, AgentKV is the four-replica deployment; the old tensor-parallel AgentKV and ThunderAgent-replicas series are omitted.

D SERVING REPLAYS OF THE AGENT BENCHMARKS

SWE-bench, Terminal-Bench and τ^2-bench evaluate an agent inside a sandbox that the reported hosts cannot run (Section E). Their *serving* side is replayed instead: each instance carries the benchmark's own task text—a SWE-bench Verified problem statement with its patch context, a Terminal-Bench instruction, or a τ^2 domain policy and database with the dataset's scripted user turns—padded with deterministic per-replica shell observations to 24–25k tokens, the context a real agent reaches mid-episode and the same scale as the document workloads above; the loaders are sized by the model's tokenizer, not by characters. These cells measure throughput and latency only and cannot yield a resolve or pass rate. Figure 2 reports both models. On Qwen3-8B the threshold sits at 95–96 instances on all three, so the 96-instance column is the crossing itself: AGENTKV separates from 96 on τ^2-bench, whose instances are largest, and from 144 on the other two, and holds a flat 74.8% prefix-hit rate across the sweep while stock vLLM's falls to 0.1–1.1% at 192 instances. The cache-aware gateway is again the fastest uncontended arm and, with four small independent caches, the first to lose its prefixes under load. On Qwen3.8-27B the same replays collapse harder: past the threshold every baseline is clipped by the cell budget at 0% reuse, and under that collapse their late requests wait behind the queue for tens of minutes. With the harness's original 600 s client timeout those cells lost more than 1% of calls to timeouts and failed the audit; they were rerun with a timeout long enough to drain the queue (1200 s, then 3000 s), after which every cell completes with zero timeouts. A cell that finishes without a timeout is identical under any timeout value, so AGENTKV's cells, which never timed out, were not rerun; the superseded originals are retained in the artifact. At 240 instances the picture is unchanged: AGENTKV holds 56.7/67.4/55.4 calls/min on the three 27B replays against 23.1–23.4 for stock vLLM.

E EXTENDED LIMITATIONS

The PRO 6000 campaign (Section 5.2) has its own caveats. Its InferCept and Continuum-TTL arms are our reimplementations of those policies at the proxy layer, not the authors' code: InferCept's prototype targets vLLM 0.2.x and cannot be built for the sm_120 architecture, and the authors' public Continuum fork is pinned to a vLLM 0.10.2 and transformers 4.x combination that does not recognize the `qwen3_5` and `gpt_oss` architectures, so the real Continuum could not run any model in that campaign. Both reimplementations speak to the policy class, not to the published systems. The vLLM-router arm was built for this campaign as four single-GPU replicas behind vllm-router's cache-aware policy; its aggregate KV capacity is about 14% below the tensor-parallel arms because weights are replicated rather than sharded, so its ratios describe a serving topology as well as a routing policy. The B200 comparison likewise contains both tensor-parallel and four-replica topologies; Figure 3 states the one global selection rule used for the DeepSeek mini figure. Finally, the SWE-bench, Terminal-Bench, and τ^2-bench workloads prepared for that host are serving-side replays of the benchmarks' prompt distributions, because the host is an unprivileged container without the sandboxes those benchmarks execute in; they measure serving behavior and cannot produce a resolve or pass rate.

The evaluation covers two four-GPU hosts and four model configurations, but most method–load cells are single audited observations; ratios are therefore descriptive rather than confidence intervals. Greedy outputs are not fully batch invariant, and the adapted document answer scores are diagnostics rather than proof of end-task quality. Broader hardware, interactive agents and replicated quality evaluation remain necessary. Finally, our primary metric is aggregate completed calls. Predictive admission prioritizes shorter projected contexts within lifecycle classes, so per-instance tail latency and starvation resistance require separate fairness evaluation.

F REPRODUCIBILITY TABLES

Table 1: Reported platform and engine settings. TP4 is one four-GPU tensor-parallel engine; 4×TP1 is four independent one-GPU replicas behind the cache-aware router.

Platform	Reported models	Engine and topology	Memory and cache
4× RTX PRO 6000 (96 GiB each)	Qwen3-8B; Qwen3.8-27B	vLLM 0.26.0, TP4; 4×TP1 for vLLM-router; 40,960-token context	0.90 HBM fraction; native KV dtype; no CPU tier except the 256 GiB ablation
4× B200	Qwen3.8-27B; GPT-oss-120B; DeepSeek-V4-Flash	vLLM 0.26.0, TP4 and 4×TP1 as labeled; 40,960-token Qwen context and 65,536 otherwise	0.90 HBM fraction; `fp8_ds_mla` KV for DeepSeek and the native model setting otherwise

Table 2: Workload and measurement settings used by the plotted campaigns.

Item	Setting
Concurrent instances	Main sweeps: 48, 96, 144, 192, and 240 concurrent instances. Each campaign records its exact load and turn count in `plan.json`.
Work per cell	Fixed planned calls (instances × recorded turns); normally four growing-context calls per instance.
Decoding	Greedy decoding, thinking disabled, prefix caching enabled.
Cache state	Engine restarted as required and external prefix-cache reset verified before each reported cell.
Throughput	Aggregate input plus output tokens divided by active campaign seconds; calls/min is retained as a secondary metric.
Latency and rates	TTFT, TPOT, inter-token latency, request latency, prefill rate, and decode rate over successful model calls.
Cell audit	Reject a failed cache reset, failure count above 1% of planned calls, or completion below 40% of planned work; the ablation additionally uses the largest load valid for every compared arm.
Replay scope	SWE-bench, Terminal-Bench, and τ^2-bench are serving-side transcript replays and do not report task success.

Table 3: Default AGENTKV controller parameters. Ablation arms change only the parameter or action named in their label.

Parameter	Value	Parameter	Value
Scheduler interval	5 s	MPC horizon	4 steps
Acting-token weight	1.0	Growth warm-up	1 sample
Initial growth g_0	256 tokens	Per-instance buffer b	100 tokens
Growth EMA α_g	0.3	Low-load threshold N_0	24 instances
Low/high-load HBM targets	1.00 / 0.88	Emergency target drop	0.10
Emergency uncached traffic	100,000 tokens/min	Minimum target	0.70
Target relaxation step	0.01	Feedback eligibility window	10 s
KV pressure threshold	0.8	Minimum persisted context	4,096 tokens
Maximum offload cap	40,960 tokens	Reuse horizon	10 s
Persistence policy	Dynamic / selective	Persistence EMA α_r	0.3

Table 4: Pinned serving software recorded in the released experiment bundle.

Component	Version	Component	Version
vLLM	0.26.0	vLLM Router	0.1.15
PyTorch	2.11.0+cu130	Transformers	5.15.1
FlashInfer Python	0.6.14	CUDA build target	13.0
Python	3.12	B200 operating system	Ubuntu 24.04