MODEL AS INFERENCE ADVISOR: LEVERAGING LLM SEMANTICS FOR AGENTIC KV CACHE SCHEDULING

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic workloads powered by Large Language Models (LLMs) are becoming increasingly critical; however, their long contexts and concurrent executions generate massive key-value (KV) caches, imposing severe GPU memory pressure. To alleviate this bottleneck, hierarchical KV cache offloading and scheduling are imperative. Existing scheduling policies typically rely on request histories or agent importance, without fully distinguishing task-specific return behavior within the same tool. This semantic ambiguity can cause scheduling mismatches that underutilize hierarchical storage resources and degrade response latency. To bridge this gap, we propose **Model as Inference Advisor (MIA)**, a novel inference system that leverages the LLM itself to expose performance-relevant task semantics as structured hints, enabling highly efficient hierarchical cache management. Experiments on Terminal-Bench demonstrate that MIA reduces mean time to first token (TTFT) and mean job completion time (JCT) by up to 28.7% and 9.1%, respectively, relative to the vLLM baseline.

1 INTRODUCTION

Agentic workloads are becoming an important target for optimizing large language model (LLM) inference systems (Luo et al., 2026; Zhu et al., 2026; Kang et al., 2026). As agents alternate between model inference and tool execution, each tool result extends the conversation, so successive requests share an increasingly long prefix. Reusing the key-value (KV) cache for this prefix avoids repeated prefill and reduces time to first token (TTFT). Preserving this benefit across requests, however, requires keeping the cache available while the session waits for shell commands, retrieval, compilation, or tests. In high-concurrency settings, numerous idle sessions accumulate massive reusable KV caches, even when not actively decoding. This immense demand for GPU high-bandwidth memory (HBM) renders KV cache capacity a primary bottleneck in agentic serving.

Hierarchical KV caching is an important way to accommodate these caches beyond limited HBM, placing inactive caches in host memory or SSD and restoring them on demand (Gao et al., 2024; Qin et al., 2025; Liu et al., 2025; Xie et al., 2026). This capacity extension introduces two coupled scheduling decisions: offloading moves inactive caches out of HBM to free space, while prefetching restores them before reuse to hide transfer latency.

Figure 1(a) illustrates how incoming requests and predicted returns trigger cache rotation between HBM and RAM across competing sessions. The timing of both movements matters (Figure 1(b,c)): offloading too early exposes restoration latency on a cold return, whereas offloading too late delays HBM reclamation. Prefetching too early occupies memory needed by other sessions; prefetching too late leaves transfer on the critical path. Thus, hierarchical caching requires coordinating which caches move and when, rather than managing each request in isolation. Baseline runtimes make these decisions from local state (Figure 2(a)).

Exploration of offloading and prefetching policies for agentic serving is still limited. Some studies estimate reuse from request histories, tool identities, observed durations, and workflow structure to set cache residency or choose victims (Li et al., 2026a; Pan et al., 2025; Zheng et al., 2026), as shown in Figure 2(b). These signals are useful but semantically ambiguous. The same bash tool can execute a status query, a selected test, a repository-wide build, or a background launch, each with different return behavior. This ambiguity can lead to inaccurate return-time estimates: offloading too early or prefetching too late exposes restoration latency, whereas offloading too late or prefetching

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.

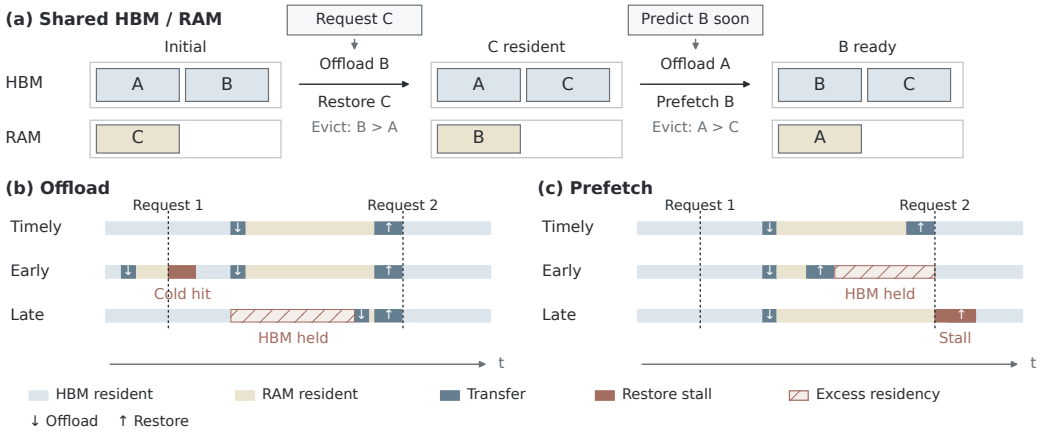


Figure 1: Hierarchical KV scheduling. (a) Requests and predicted returns trigger cache rotation according to eviction priority. (b,c) With identical request arrivals, offload and prefetch timing trade HBM occupancy against exposed restore latency.

too early occupies HBM needed by other sessions. Another approach exposes application-level interfaces (Biswas et al., 2026; Gim et al., 2025), through which applications directly manage KV caches or pass agent orchestration information to the inference engine (Figure 2(c)). These interfaces require workload-specific adaptation, increasing development effort for agent developers while still leaving task-specific semantics difficult to obtain.

We therefore seek a more accurate, low-cost approach to KV-cache management through automated use of task semantics. Inspired by the model's role in controlling the agent harness, we let its understanding of the current operation, scope, and execution mode inform the inference scheduler. As models are not well suited to predicting exact completion times or directly issuing physical scheduling commands, we consider it more appropriate to use model-generated semantics to assist runtime decisions. We term this approach **Model as Inference Advisor (MIA)** (Figure 2(d)): the model provides structured semantic hints, while the runtime retains cache control, calibrating these hints against observed behavior to finalize offloading and prefetching decisions.

Realizing MIA requires addressing three challenges. First, free-form model outputs can be inaccurate and ambiguous, failing to express performance characteristics in a form useful to the inference system. Second, a gap separates semantic estimates from feasible scheduling policies, whose effectiveness depends on infrastructure performance and workload conditions. Third, sessions cannot be scheduled in isolation: live memory pressure, transfer costs, and competing sessions all affect whether a cache movement is beneficial.

MIA addresses these challenges through a model-side Hint protocol and an inference-side scheduling backend. The model emits a grammar-constrained, structured Hint describing performance-relevant facets such as operation, scope, execution mode, and a coarse completion window, without naming cache blocks or prescribing physical actions. The backend fuses this Hint with tool statistics and online calibration to estimate return behavior, then evaluates that estimate together with live cache costs to guide both offloading and prefetching. A two-plane design separates semantic advice from runtime decision making: the advice plane produces and calibrates scheduling evidence, while the decision plane validates current resource state and executes stable actions. Missing or invalid Hints naturally fall back to statistical or native policies.

We make the following contributions:

- We introduce MIA, a novel inference system that conveys task-specific semantics to the inference runtime, expanding the information available to the inference engine for KV-cache scheduling.
- We design a structured Hint protocol and a semantics-aware scheduling backend that convert semantic information into calibrated return-time distributions, enabling better-informed KV-cache offloading and prefetching decisions.

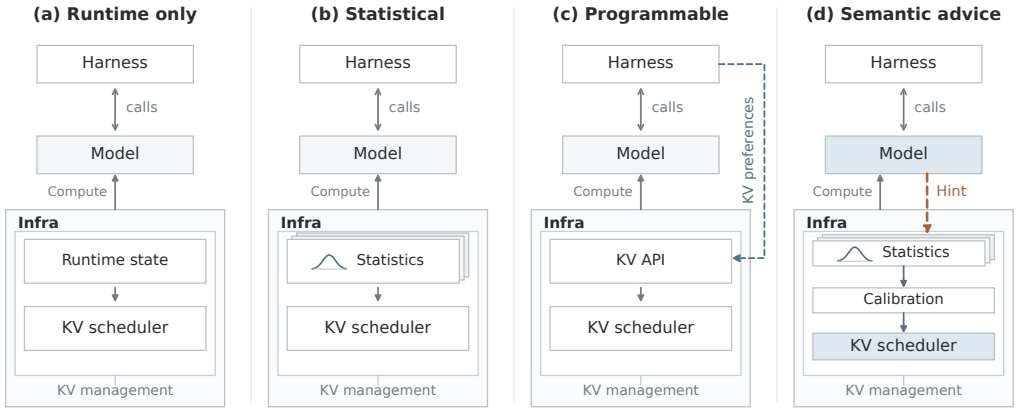


Figure 2: Information paths for KV scheduling. (a) Runtime policies use local state. (b) The inference runtime collects tool statistics to estimate return times. (c) Programmable KV interfaces accept application-defined KV preferences. (d) MIA supplies model-generated semantic advice; the runtime combines it with statistics and calibration and retains cache control.

- We implement MIA on vLLM and evaluate it using agentic workload traces. On Terminal-Bench, MIA reduces mean TTFT and mean JCT by up to 28.7% and 9.1%, respectively, relative to the vLLM baseline, with auxiliary inference cost included.

2 Background

2.1 Agentic Workloads

Agentic workloads repeatedly alternate between LLM inference and tool execution within a persistent session (Luo et al., 2026; Zhu et al., 2026; Kang et al., 2026). Each return from a tool adds observations to the conversation, so subsequent requests reuse an increasingly long prefix. KV caching avoids recomputing that prefix, but the state must remain available across requests rather than only during decoding. As context length and session concurrency grow, the amount of reusable KV state can exceed GPU high-bandwidth memory (HBM), making cache capacity a central concern for agentic serving.

Tool execution creates intervals in which a session is inactive on the GPU but its cache remains useful. Keeping every waiting session in HBM wastes space that active requests could use; discarding every cache forces restoration or repeated prefill when sessions resume. These intervals differ with the task: the same shell tool may perform a short status query or a repository-wide build. The system must therefore accommodate many long-lived caches while deciding which ones need fast access next, rather than optimizing the lifetime of a single request in isolation.

2.2 Hierarchical Cache

Hierarchical KV caching extends cache capacity beyond HBM with host DRAM and, optionally, slower storage (Gao et al., 2024; Qin et al., 2025; Liu et al., 2025; Xie et al., 2026). It allows inactive session prefixes to remain reusable without keeping all of them on the GPU. An HBM-resident prefix can be reused directly; an externally resident prefix must be restored before computation can reuse it. Capacity expansion thus introduces a scheduling problem: which caches should occupy HBM, and when should state move between tiers?

Figure 1 illustrates two coupled decisions. **Offloading** releases HBM occupied by a selected inactive cache while preserving it in an external tier. Executing this prematurely risks a cold cache miss upon an early return, whereas delaying it impedes the reclamation of vital memory space. **Prefetching** brings external state back before the next request: late prefetch leaves transfer latency exposed, while early prefetch occupies HBM that other sessions could use. Because all sessions share a finite cache budget, a movement changes the choices available for subsequent requests. Agentic KV scheduling

must balance this ongoing rotation using both expected reuse times and current memory and transfer costs.

2.3 Existing Methods

Existing cache schedulers estimate future reuse from runtime-visible signals such as recency, request history, cache size, tool identity, observed tool duration, and queue state (Zheng et al., 2024; Li et al., 2026a; Pan et al., 2025; Zheng et al., 2026). These signals are causal, inexpensive, and broadly available, making them strong baselines. However, they group together operations that look identical to the runtime but represent different tasks. As illustrated in Figure 3, the same shell tool can issue a fast status check, a selected test, a background launch, or a long compilation; the same retrieval tool can perform either a narrow lookup or a broad scan.

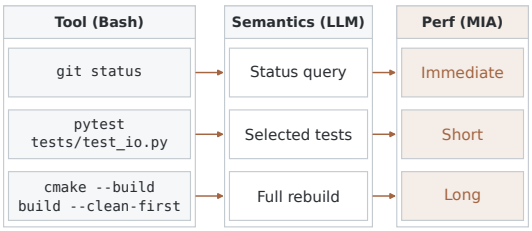


Figure 3: Calls to the same tool can exhibit different performance characteristics because of their task semantics. MIA maps LLM-generated semantics to scheduling features.

The model issuing the tool call already understands much of this distinction: the immediate intent, target scope, execution mode, and current task stage. The inference runtime does not observe these semantics. Our key observation is not that the model can predict an exact completion time, but that its semantic view can reduce ambiguity in runtime cohorts. MIA surfaces this semantic knowledge as probabilistic advice, empowering the inference system to make highly informed, final resource allocation decisions.

3 Method

MIA turns task semantics into evidence for KV-cache scheduling, rather than asking the model to control memory directly. Figure 4 separates model-side Hint generation from runtime-side estimation and scheduling. The key link is a calibrated return-time distribution: it combines the model's interpretation of the current work with observed tool behavior, while leaving resource costs and physical actions to the inference engine.

3.1 Problem Formulation

Consider a session i waiting for its current tool call or batch of calls. Let T_i be the total time from dispatch until all calls in that batch return, and a_i the elapsed wait. The scheduler estimates the remaining wait $R_i = T_i - a_i$, conditioned on the calls not yet having returned. All time quantities are measured in seconds, and the estimate is constructed anew for each tool-wait interval. MIA combines tool history and a semantic Hint to estimate return behavior, then uses the estimate together with memory pressure, reusable KV state, and restoration costs to guide cache movement.

3.2 Model-Generated Semantic Hints

Describe work in performance view. A tool name alone conflates operations with different return behavior. For example, a Bash invocation may inspect a file, run selected tests, or launch a background build. MIA describes each current call through operation, scope, and completion mode, together with a short description and a soft time window $[l_i, u_i]$; Table 1 summarizes the fields used by the runtime. Scope distinguishes one target from a subset or the whole workload; mode distinguishes waiting for completion from returning a background handle. These facets let the model express why two calls to the same tool may take different amounts of time. The window summarizes plausible return times in seconds, allowing the model to convey information with credible temporal bounds, such as timeout limits.

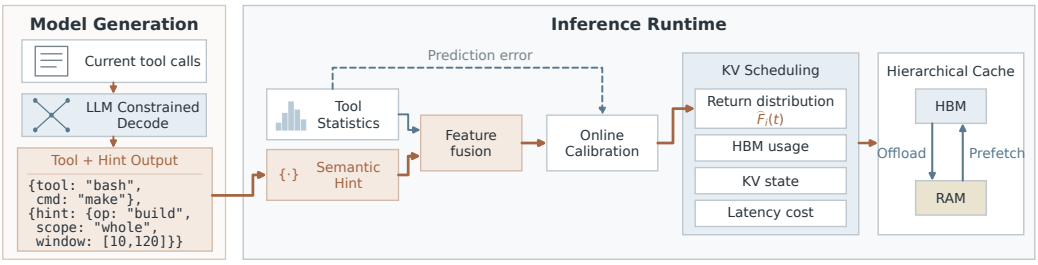


Figure 4: MIA spans model generation and inference runtime. The model provides semantic hints; the engine fuses them with tool statistics and online calibration to estimate return behavior and schedule hierarchical KV-cache movement.

Table 1: Semantic Hint content and usage in MIA.

Field	Example	Runtime use
Operation / scope / mode	test / subset / sync	Select residual calibration group
Soft window	$[1, 30]$ seconds	Set the current prediction scale
Confidence	low or high	Label a calibration group

Constrained invocation-specific generation. MIA combines prompting with constrained generation to guide the same model to produce a structured semantic Hint after generating a tool call. The prompt elicits performance-relevant semantics about the intended work, while a grammar-constrained schema specifies the Hint's output structure and allowable field values. This design captures rich task semantics while preserving the inference engine's standard autoregressive decoding process.

3.3 Semantics-Aware Return-Time Estimation

Constructing the statistical prior. MIA first constructs a statistical prior from completed tool waits. It combines the weighted empirical distribution of all observed waits with that of calls sharing the current tool-name combination. For a single call, this combination is its tool name; for a batch, it comprises the names of its constituent calls. The tool-specific distribution receives more weight as its observation count increases, while sparse or unseen combinations rely more on global history. We denote the resulting prior CDF by $B_i(t)$, which estimates the probability that the total wait T_i is at most t seconds.

Calibrating semantic windows. The numerical contribution of a Hint is its soft window $[l_i, u_i]$, whose endpoints describe plausible lower and upper bounds on the total wait from dispatch, not the remaining wait from the current instant. MIA interprets this window through historical prediction errors. For a completed historical invocation or batch j, let T_j be its observed total wait and $[l_j, u_j]$ its associated Hint window. We transform both endpoints using $\log(1 + t)$, take their midpoint as the center μ_j, and use their half-width, bounded below by a positive constant, as the scale σ_j. The normalized residual is

$$z_j = \frac{\log(1 + T_j) - \mu_j}{\sigma_j}. \tag{1}$$

Here σ_j is a window-derived scale. Its positive lower bound prevents narrow windows from amplifying small errors. Positive residuals indicate returns later than the window center predicts, while negative residuals indicate earlier returns. The log scale accommodates waits spanning different orders of magnitude. Observed durations need not fall within their Hint windows.

Residual history is pooled globally, by tool combination, and by a semantic group that additionally includes the associated operation/scope/mode tuples and confidence label. Groups with fewer observations borrow more evidence from broader groups. For the current session i, MIA maps this pooled residual distribution back to durations using the current window's center μ_i and scale σ_i, defined in the same way as for historical calls. A residual z maps to $\max\{0, \exp(\mu_i + \sigma_i z) - 1\}$

seconds. The resulting CDF $S_i(t)$ is the semantic forecast, where the Hint supplies the current time scale and selects relevant error history and observations supply empirical probabilities. MIA then combines it with the statistical prior:

$$D_i(t) = (1 - \lambda_i)B_i(t) + \lambda_i S_i(t), \qquad t \geq 0. \tag{2}$$

Here $D_i(t)$ is the fused total-wait CDF, and $\lambda_i \in [0, 1]$ is a bounded mixture weight that increases with the amount of available residual history. Without a valid window or residual history, MIA uses B_i alone. A small exponential tail is mixed into D_i to obtain the final total-wait CDF $\widehat{F}_i$, preserving support beyond previously observed durations. Soft windows do not truncate this distribution.

Online correction and remaining time. For each group, calibration combines initial historical observations with a bounded buffer of newly completed waits, weighting recent observations more heavily and increasing their contribution as evidence accumulates. A completed call or batch updates tool history and, when its Hint is available, the corresponding residual groups. The Prediction error feedback in Figure 4 denotes these updates.

The calibrated distribution $\widehat{F}_i$ describes the total wait from tool dispatch, whereas scheduling decisions depend on how soon the tool will return from the current instant. If session i has already waited a_i seconds without a return, we condition this distribution on $T_i > a_i$. The probability of returning within the next r seconds is therefore, for $r \geq 0$ and $\widehat{F}_i(a_i) < 1$,

$$G_i(r \mid a_i) = \frac{\widehat{F}_i(a_i + r) - \widehat{F}_i(a_i)}{1 - \widehat{F}_i(a_i)}. \tag{3}$$

The numerator removes probability mass assigned to returns before the current time, and the denominator renormalizes the remaining mass. Thus, the same Hint can support a different decision as the wait grows, without requiring a new Hint or a completed observation. If the surviving probability mass is too small for reliable numerical conditioning, the forecast is not used to override native scheduling.

3.4 Distribution-Guided KV Scheduling

The return distribution above is scheduling advice. As Figure 4 shows, the runtime combines predicted reuse with HBM usage, current KV residency, and measured restoration costs. The aim is to make room for active requests while retaining or restoring caches whose near-term reuse would otherwise incur substantial delay. Since sessions compete for HBM, a movement must be evaluated together with its effects on other sessions rather than in isolation.

Offloading under competition. When allocation requires space, MIA compares feasible eviction choices with the native policy. It favors choices that release HBM while imposing less expected restoration delay across affected sessions, taking both return timing and the amount of reusable KV state into account. For example, a cache predicted to be reused later can be a better victim than one likely to return soon, but timing alone is insufficient when their restoration costs differ. MIA changes the native choice only when the predicted benefit justifies the additional harm to other sessions and the action passes runtime safety checks; otherwise, it retains the native policy.

Prefetch timing and admission. A return-time quantile and the estimated transfer time determine when to reconsider restoring an external cache. At that point, the runtime checks whether the cache is still missing and whether restoring it early is likely to reduce delay enough to justify displacing other caches. Prefetching too early can waste HBM or cause the imported cache to be evicted before use, while prefetching too late leaves restoration on the request's critical path. MIA therefore rechecks current residency and memory pressure before issuing a transfer, and skips imports that are already unnecessary or insufficiently beneficial. Tool completions update the histories used for subsequent predictions, closing the scheduling loop.

4 Evaluation

We evaluate latency across job arrival rates, latency quantiles, ablations, and performance profiles using 40 Terminal-Bench traces with complete replay information. We additionally conduct a long-

horizon test using 28 Trace Commons traces with longer jobs and tool waits. Supplementary stability, throughput, and cache-capacity experiments are reported in Appendix B.

4.1 SETUP

We construct Terminal40 from 40 non-image Terminal-Bench traces with complete replay information, containing 1,895 business requests (Merrill et al., 2026). This is our primary workload for the load sweep, quantile analysis, ablations, performance profiles, and supplementary experiments. For the long-horizon test, we use Common28, comprising 28 Trace Commons traces with 4,354 business requests and 4,020 tool rounds. These traces contain longer jobs and tool waits, complementing the Terminal40 load sweep (Section 4.3). Both workloads are replayed with recorded business outputs preserved through forced decoding and original tool waits retained, allowing us to evaluate serving performance independently of autonomous task-solving quality.

The deployments are single-GPU Qwen3.5-9B on A100, Qwen3.5-35B-A3B on A100, and Qwen3.5-9B on RTX 4090. We compare vLLM, Continuum (Li et al., 2026a), and MIA. Throughout the paper, vLLM denotes the baseline using Mooncake-backed hierarchical KV caching with minimal restoration-correctness adaptations, rather than an L1-only configuration. Since an open-source implementation of Continuum was unavailable for our evaluation, we reproduce it based on the paper. MIA auxiliary inference shares the business service and is included in end-to-end cost.

We vary the job arrival rate, measured in jobs per second (JPS), to evaluate each method under different load intensities. TTFT measures business-request start to first token; JCT measures scheduled job arrival to completion, including tool waits.

4.2 MAIN EXPERIMENT

Our main experiment measures serving performance on Terminal40 across JPS 0.020/0.025/0.030/0.035, as shown in Figure 5. Across all 12 completed vLLM–MIA pairs, mean TTFT decreases by 1.6–28.7% and mean JCT by 1.7–9.1%.

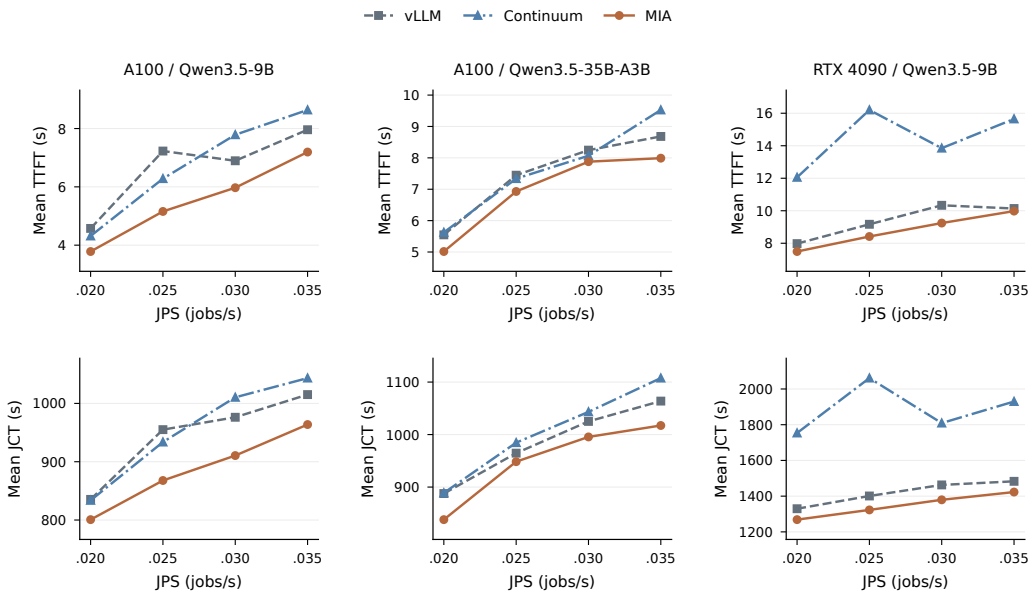


Figure 5: Terminal40 performance across arrival rates. Rows show mean TTFT and JCT; columns show the three deployments.

Table 2: A100 9B / Terminal40 / JPS 0.030. Mean latency in seconds; $\Delta = (\text{variant}/\text{MIA} - 1) \times 100\%$, so positive values indicate higher latency.

Variant	Mean TTFT	Δ vs. MIA	Mean JCT	Δ vs. MIA
vLLM	6.895	+15.48%	976.237	+7.21%
MIA (w/o tool statistics)	6.051	+1.34%	911.594	+0.11%
MIA (w/o LLM hint)	6.359	+6.50%	921.260	+1.17%
MIA (w/o calibration)	6.218	+4.15%	928.510	+1.97%
MIA	5.971	+0.00%	910.568	+0.00%

4.3 Long-Horizon Test

We use Common28 to evaluate prolonged agent sessions, retaining all tool rounds with a 200k context limit and a 160k-token sliding window. At JPS 0.002 on the two A100 deployments, MIA reduces mean TTFT by 18.8–34.0% and mean JCT by 2.0–8.7% relative to vLLM (Figure 6), demonstrating benefits even for jobs lasting thousands of seconds.

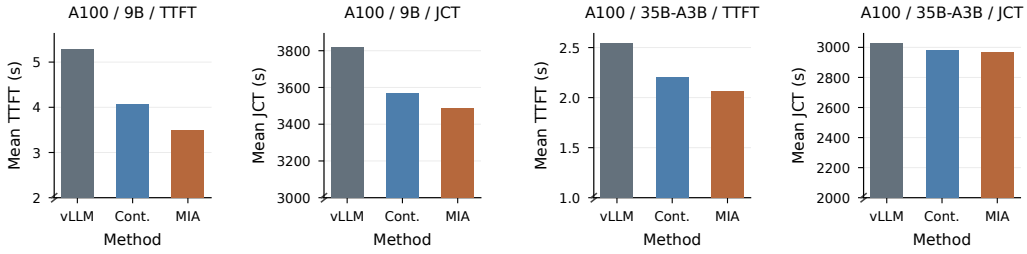


Figure 6: Long-horizon test on Common28.

4.4 Latency Quantiles

We compare vLLM and MIA on Terminal40 with A100 9B across JPS 0.020/0.025/0.030/0.035 (Figure 7). MIA consistently lowers both P50 and P90 of TTFT and JCT across these loads, showing that the mean-latency gains extend to typical and tail requests and jobs in this setting.

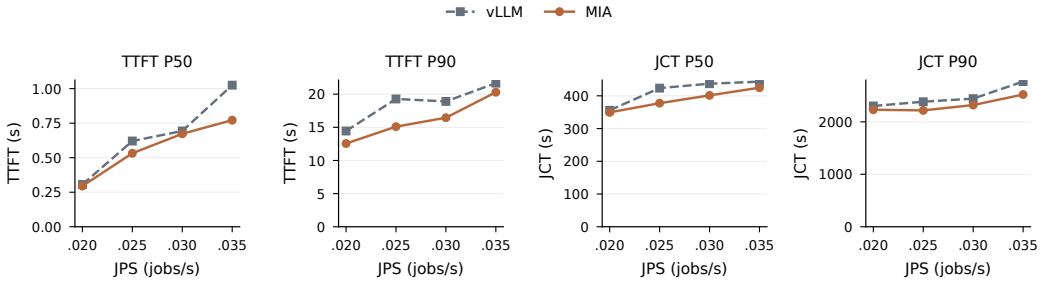


Figure 7: Terminal40 latency quantiles on A100 9B.

4.5 Mechanism Ablation

Table 2 evaluates the contributions of MIA's components on A100 9B / Terminal40 / JPS 0.030. Removing tool statistics disables tool-conditioned history but retains global priors; removing Hints disables semantic advice; removing calibration freezes semantic updates after the same initialization but retains transfer-cost profiling.

Table 3: A100 9B / Terminal40 / JPS 0.030. Service-level GET time can overlap queueing and computation; it is not additive TTFT/JCT loss.

Metric	vLLM	MIA
Successful GET batches	984	945
GET volume (GiB)	1,410.573	1,076.287
Cumulative GET API time (s)	2,387.623	1,948.139
Engine preemptions	103	24
Hint computed tokens	0	180,053
Policy mean (ms)	–	1.555
Policy P50 (ms)	–	1.202
Policy P90 (ms)	–	4.030

Full MIA achieves the lowest mean TTFT and JCT. Removing Hints raises them by 6.50% and 1.17%; freezing calibration raises them by 4.15% and 1.97%. Tool-conditioned history has a smaller marginal effect (1.34% and 0.11%), supporting complementary but unequal contributions in this setting.

4.6 Performance Profile

Table 3 profiles A100 9B / Terminal40 / JPS 0.030. MIA lowers GET volume from 1,410.573 to 1,076.287 GiB and preemptions from 103 to 24, consistent with reduced restoration traffic and memory pressure. The cost includes 180,053 Hint-computation tokens and 1.555/1.202/4.030 ms mean/P50/P90 policy time.

5 Related Work

KV-cache memory and placement. PagedAttention and SGLang improve GPU cache reuse (Kwon et al., 2023; Zheng et al., 2024); CachedAttention, Mooncake, LMCache, and Strata extend caching across memory tiers (Gao et al., 2024; Qin et al., 2025; Liu et al., 2025; Xie et al., 2026), while Preble balances locality and load (Srivatsa et al., 2024). MIA complements these mechanisms with task-specific scheduling evidence.

Agent-aware KV scheduling. Agentix and ThunderAgent schedule agent programs (Luo et al., 2026; Kang et al., 2026). Continuum predicts tool return times, KVFlow exploits workflow graphs, and PBKV predicts workflow reuse (Li et al., 2026a; Pan et al., 2025; Zheng et al., 2026). MIA instead elicits interpretable task semantics and calibrates them with observed tool behavior.

Cross-layer and model-generated hints. SYMPHONY, Sutradhara, and Pie expose cross-layer coordination or programmable cache control (Agarwal et al., 2026; Biswas et al., 2026; Gim et al., 2025). ToolCaching and SelfBudgeter use model-generated metadata or budgets (Zhai et al., 2026; Li et al., 2026b). MIA applies semantic advice to KV scheduling while retaining resource control in the runtime.

6 Conclusion

In this work, we design and implement MIA, a semantics-aware KV-cache management system for agentic inference. MIA represents a significant leap forward in model-advised cache management through structured semantic Hints, statistical–semantic fusion with online calibration, and resource-aware offloading and prefetching. Together, these techniques turn task-specific semantics into informed cache decisions, reducing response latency and job completion time. Extensive evaluations demonstrate MIA's superior performance.

AI USE STATEMENT

We used generative AI tools to assist with writing and polishing, translation, drafting sections of the paper, and retrieving and discovering related work. We also used them for research ideation and execution, including discussing method formulations, providing feedback on experimental design, assisting with code and plotting scripts, and interpreting results. AI-generated drafts and suggestions were revised through author review, and bibliographic entries were checked against original publication sources. The authors remain responsible for the final text, claims, experimental evidence, and artifacts, including all AI-assisted content.

REPRODUCIBILITY STATEMENT

The main paper describes the Hint protocol, return-time estimator, scheduling approach, and evaluation metrics. Appendix A details the replay procedure, Hint acquisition and cost accounting, and calibration and resource settings. Appendix B provides supplementary measurements, example request traces, and the complete Terminal40 latency table, together with the scope and conditions of the comparisons.

REFERENCES

Saurabh Agarwal, Bodun Hu, Anyong Mao, Aditya Akella, and Shivaram Venkataraman. SYMPHONY: Enabling Compute-Memory disaggregation in LLM serving systems. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 2027–2041, Renton, WA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/agarwal.

Anish Biswas, Kanishk Goel, Srivarshinee S, Jayashree Mohan, Alind Khare, Anjaly Parayil, Ramachandran Ramjee, and Chetan Bansal. Sutradhara: An intelligent orchestrator-engine codesign for tool-based agentic inference, 2026. URL https://arxiv.org/abs/2601.12967.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with cachedattention, 2024. URL https://arxiv.org/abs/2403.19708.

In Gim, Zhiyao Ma, Seung seob Lee, and Lin Zhong. Pie: A programmable serving system for emerging llm applications, 2025. URL https://arxiv.org/abs/2510.24051.

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, 2026. 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, 2023. URL https://arxiv.org/abs/2309.06180.

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, 2026a. URL https://arxiv.org/abs/2511.02230.

Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, Kai Jia, and Zhifang Sui. Selfbudgeter: Adaptive token allocation for efficient llm reasoning, 2026b. URL https://arxiv.org/abs/2505.11274.

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, 2025. URL https://arxiv.org/abs/2510.09665.

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, Renton, WA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/luo.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbj"orn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows, 2025. URL https://arxiv.org/abs/2507.07400.

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, Santa Clara, CA, February 2025. USENIX Association. ISBN 978-1-939133-45-8. 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, 2024. URL https://arxiv.org/abs/2407.00023.

Zhiqiang Xie, Ziyi Xu, Mark Zhao, Yuwei An, Vikram Sharma Mailthody, Scott Mahlke, Michael Garland, and Christos Kozyrakis. Strata: Hierarchical context caching for long context language model serving. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, pp. 1–16, Seattle, WA, July 2026. USENIX Association. ISBN 978-1-939133-55-7. URL https://www.usenix.org/conference/osdi26/presentation/xie-zhiqiang.

Yi Zhai, Dian Shen, Junzhou Luo, and Bin Yang. Toolcaching: Towards efficient caching for llm tool-calling, 2026. URL https://arxiv.org/abs/2601.15335.

Haoyu Zheng, Fangcheng Fu, Jia Wu, Binhang Yuan, Yongqiang Zhang, Hao Wang, Yuanyuan Zhu, Xiao Yan, and Jiawei Jiang. Efficient serving for dynamic agent workflows with prediction-based kv-cache management, 2026. URL https://arxiv.org/abs/2605.06472.

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, 2024. URL https://arxiv.org/abs/2312.07104.

Kan Zhu, Mathew Jacob, Chenxi Ma, Yi Pan, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. Tracelab: Characterizing coding agent workloads for llm serving, 2026. URL https://arxiv.org/abs/2606.30560.

A Implementation and Reproducibility

Replay and measurement. The replay preserves recorded business outputs through forced decoding and executes tool-wait intervals at their original durations. Jobs follow fixed Poisson arrival schedules generated with seed 20260905; successive tool rounds are not independently rate-limited. Paired methods use the same session ordering and arrival offsets. The client imposes no additional active-job gate, while the engine admits up to 64 sequences. Measurement follows warmup and cache reset. TTFT is measured at the business-request boundary, and JCT starts at the scheduled job arrival, including queueing and tool waits. Completion checks verify the expected jobs and requests and the recorded output sequence, rather than treating a successful server exit as a complete run.

Hint generation and cost. The deployed model also generates Hints, using a same-server auxiliary request to fit the existing inference interface. This request sees only the current tool names and arguments, not future tool results. The prompt asks for call-ordered groups of operation, scope, completion mode, and a short description, plus one soft return window for the entire batch. It distinguishes executed commands from quoted text and foreground completion from a background launch; a timeout is not treated as the expected duration. JSON-schema-constrained decoding uses temperature zero. Tool-argument input is limited to 2,048 tokens, preserving the first 1,536 and last 512 when necessary; this limit does not truncate business prompts. The output budget starts at 192 tokens and increases for larger batches. Acquisition begins only if the tool is still running after three seconds, with at most one auxiliary request in flight. The statistical prior is available while the Hint is pending. Auxiliary work shares the service's compute and cache resources and is included in end-to-end measurements, including submitted work later abandoned by the client.

Calibration and scheduling controls. Calibration starts from a fixed historical prior and incorporates only completed, received tool observations. A batch supplies one observed wait for all its calls; its duration is not duplicated into artificial subtask observations. Descriptions are retained as metadata rather than parsed into scheduling commands, and confidence is a calibration-group label rather than a probability. The residual estimator adds a 1% exponential tail to retain support beyond observed waits. The evaluated controller uses a 15-second decision horizon, a 5-millisecond minimum predicted benefit, and a 10-millisecond decision-computation budget. Prefetch planning uses the median return-time quantile and rechecks live cache state before admission. These settings distinguish the forecast from the runtime's final resource decision.

Resource and ablation controls. The A100 main configurations allocate 768 KV blocks for 9B and 320 for 35B-A3B, with 128 GiB Mooncake stores and 512 MiB staging per rank; the RTX 4090 vLLM–MIA comparison uses a 64 GiB store. Capacity sweeps vary the KV-block quota rather than assuming that equal block counts imply equal bytes across models. Ablations retain the core backend and selectively remove tool-conditioned history, semantic Hints, or online calibration. The frozen-calibration variant starts from the same fitted prior but does not update the semantic predictor; independent transfer-cost profiling remains enabled. The five-run stability experiment resets service, cache, and prior state between runs while keeping the input and arrival schedule fixed. These repeats measure service-run variability, not sensitivity to different arrival seeds.

B Additional Results

B.1 Independent-run Stability

We evaluate five service-reset repeats per method for vLLM and MIA on A100 9B / Terminal40 / JPS 0.030. Inputs and arrival schedules are fixed; service, KV state, and priors are reset. All five runs are retained. This tests run variability, not variability across Poisson seeds or tool-interaction rounds.

Repeat protocol. Each repeat uses the same input and arrival schedule. Per-run records retain mean/P50/P90 TTFT and JCT for all five runs. We use independent service resets rather than request-level bootstrap samples to characterize run variability.

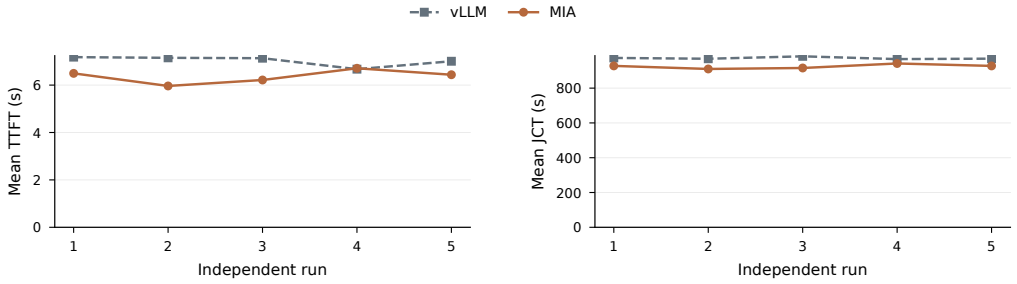


Figure 8: Mean TTFT and JCT across five independent service runs, A100 9B / Terminal40 / JPS 0.030.

Table 4: MIA on A100 9B / Terminal40 / JPS 0.030. Counts have different denominators and do not establish hint-only benefit.

Metric	Terminal40
Hint submissions	231
Hint returns	197
Valid hints	197
Handles used by decisions	170
Full-policy applications	588
Demote-order events	588

B.2 Method Effectiveness

Table 4 distinguishes hint production, decision participation, and physical cache operations on Terminal40 with A100 9B / JPS 0.030. Of 197 valid Hints, 170 are used by decisions, but the 588 full-policy applications are not 588 hint-only gains.

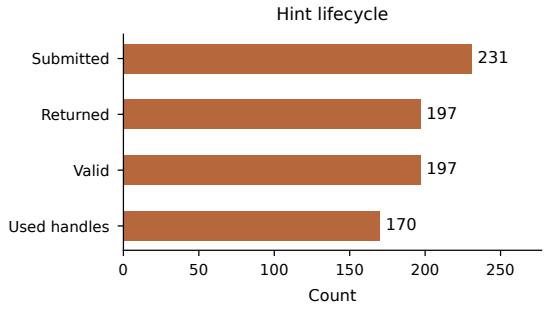


Figure 9: MIA Hint lifecycle counts, A100 9B / Terminal40 / JPS 0.030.

B.3 Finite-trace Throughput

The throughput view uses the same A100 9B / Terminal40 / JPS 0.030 scene. Output tokens per second divides business output by the measurement interval through final completion, including arrivals and drain.

B.4 KV-capacity Sensitivity

We vary HBM KV capacity on A100 at Terminal40 / JPS 0.030: 512, 640, 768, 1024, and 1536 blocks for 9B, and 224, 256, 320, 384, and 448 blocks for 35B-A3B. Each model uses its own block layout, so equal block counts need not represent equal memory sizes.

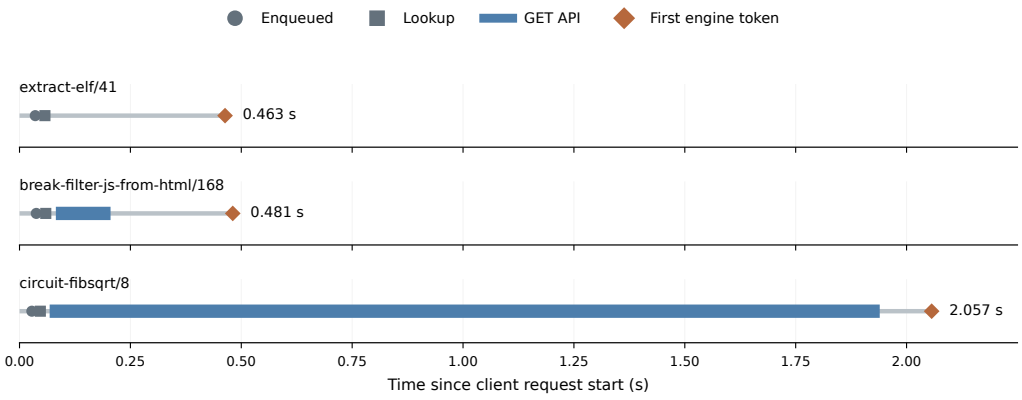


Figure 10: Three observed MIA business-request event traces.

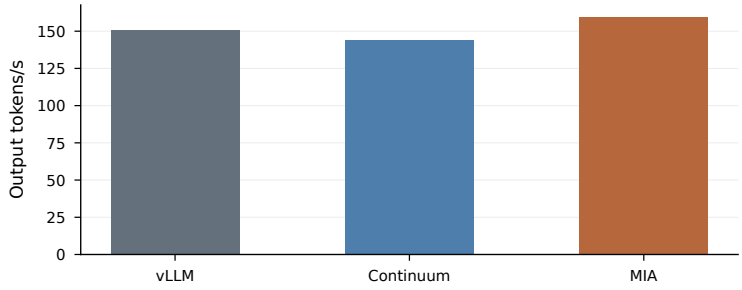


Figure 11: Finite-trace business output rate at A100 9B / Terminal40 / JPS 0.030.

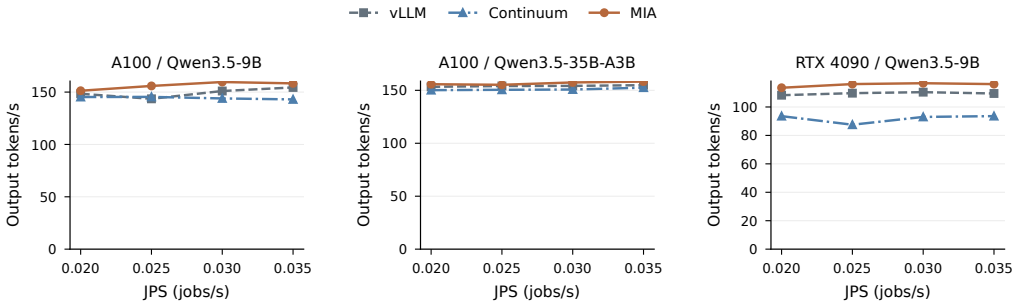


Figure 12: Terminal40 business output tokens per second over the finite-trace measurement window, including arrivals and drain.

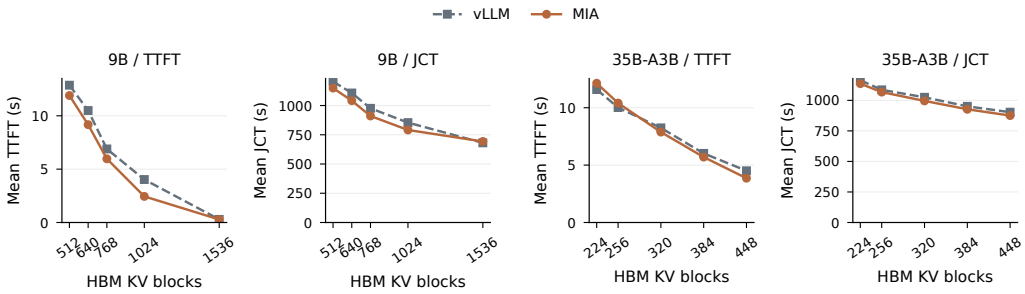


Figure 13: KV-capacity sensitivity on A100 / Terminal40 / JPS 0.030: 9B (left) and 35B-A3B (right). Each model retains five measured capacities.

The curves retain regressions, including the low-capacity 35B TTFT results and the 9B JCT result at 1536 blocks.