HiMA: Efficient Hybrid Model Serving for Agentic Systems

Anonymous Author(s)
Affiliation
Address
email

Abstract

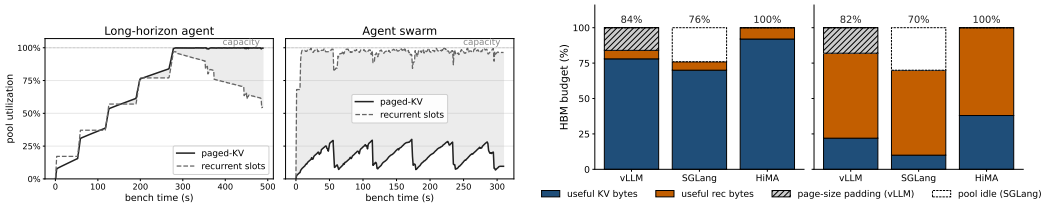
Hybrid models (e.g., Qwen3.5, Kimi Linear) are increasingly adopted for their strong capabilities in agentic applications. Compared to standard LLMs, hybrid models introduce an additional memory pool for the recurrent-state snapshots, along with the KV cache pool. This dual-pool design poses new challenges for existing model serving systems (e.g., SGLang and vLLM) and may significantly degrade their performance. First, the current LRU-based block management on the recurrent state pool does not account for shared prefixes. More importantly, the size of each pool is fixed in advance, which inevitably creates bubbles (wasted memory), especially during agentic execution, whether under agent swarms (many parallel agents) or long-horizon agents. We introduce HiMA, a hierarchical memory management framework for hybrid models in agentic systems. First, we propose a new metric — loss-per-byte (LPB) — that quantifies the cost of evicting a shared prefix. Second, we propose a dynamic memory allocation algorithm that periodically reallocates memory between the two pools through two actions: transferring memory across pools and evicting memory within a pool. More specifically, we formulate the time cost as an optimization problem and periodically solve it based on the current memory allocation and request load. We further perform fine-grained per-query allocation checking and adjustment to minimize latency overhead. We implement HiMA as a plugin module for SGLang. Across diverse agent-serving workloads, HiMA delivers **7–94%** output throughput improvements and **25–64%** P99 Time to First Token (TTFT) reductions. Finally, we provide a theoretical analysis showing that the cost of our proposed transferring action is bounded and close to the optimal memory allocation.

1 Introduction

Existing LLM serving systems have a standardized pipeline for handling vanilla attention layers: a page-indexed KV cache for attention [21, 29], with a prefix-cache subsystem stacked on top for shared KV across requests [40, 13, 11, 37]. Recent works have developed hybrid models that interleave standard attention with linear attention. In practice, these have been increasingly adopted in recent models (Qwen3-Next, Kimi Linear, Nemotron 3 Super, Olmo Hybrid) [8, 36, 25, 26, 24] and have shown extensive performance improvements. However, this current pipeline does not consider hybrid models and suffers from the following two fundamental limitations.

① Existing model servers do not capture differences between KV blocks and recurrent-state snapshots. KV blocks grow linearly with sequence length. Caching one KV block saves the prefill computation for the tokens in that block. Recurrent-state snapshots (i.e., a compressed summary of all tokens processed), by contrast, are fixed-size regardless of how long the prefix is. A cache hit on one snapshot saves the recurrent-layer computation across the *entire prefix* it represents. This creates an asymmetry: one byte of cached recurrent state saves far more compute than one byte of cached KV, because it

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.



(a) SGLang pool utilization over time: KV binds on the long-horizon agent, recurrent slots bind on the swarm.

(b) Steady-state HBM: vLLM padding and SGLang pool-idle waste 16–30%; HiMA reclaims it.

Figure 1: The partition bubble on Qwen3.5-35B-A3B under two agent workloads.

amortizes over the full prefix length rather than just a single block. Existing inference engines fail to account for this asymmetry in their eviction policies. SGLang's `MambaRadixCache` [40] evicts by `last_access_time`, while vLLM's `HybridKVCacheCoordinator` [21] evicts by how long each block has been free. Neither system weighs *hit frequency* (how valuable the cached item is) against *storage cost* (how many bytes it consumes). As shown later in Section 4, this creates notable overhead in both latency and storage cost. This is amplified under traffic perturbations, commonly seen in agentic tasks where system prompts are frequently reused across varying request patterns.

② KV and recurrent states have fixed pool sizes regardless of workload demands. SGLang has two separate pools, so inefficiency occurs when one pool fills while the other sits idle, and the unused bytes cannot be reallocated. vLLM v1 unifies the two under a single `BlockPool` by forcing every layer to share a pre-fixed page size, but this prevents flexible allocation based on actual demand. This limitation is particularly problematic for agentic workloads, which exhibit diverse memory patterns: A *long-horizon agent* is KV demand dominant, in that it runs a single session for hours, accumulating tool-call traces into a few very-long-context requests. An *agent swarm* that dispatches many sub-agent calls in parallel is recurrent-state dominated. Neither SGLang's throughput-optimal partition nor vLLM's fixed page-size budget can adjust pool sizes flexibly based on specific workloads. This creates computational bubbles that ultimately harm overall inference latency (Fig. 1).

In this paper, we propose HiMA, a novel hierarchical memory management framework specifically designed for serving hybrid models. To address ①, we propose a novel metric of loss-per-byte (LPB), which weights each block's path-counted hit rate by its bytes and by the pool's recovery cost. Using LPB as the eviction key can enable high-yield internal anchors to survive cold-burst perturbations rather than being evicted as "oldest." We name this as intra-pool layer.

To address ②, we propose a dynamic pool allocation mechanism (also referred to as the inter-pool layer). At a high level, we still use two separate pools and model memory allocation as an optimization problem. We define two operations: transferring memory across pools and evicting memory within a pool. The optimization objective is to plan a sequence of actions that minimizes inference latency while maximizing pool utilization for both pools. We periodically solve this optimization function based on the current memory allocation and request load. We further perform fine-grained per-request allocation checking and adjustment to minimize latency overhead.

We implement HiMA as a plugin module for SGLang for easy integration. Across diverse agentic workloads, HiMA delivers 7–94% output throughput improvements and 25–64% P99 Time to First Token (TTFT) reductions. We also conduct a rigorous ablation study to validate our hierarchical design. We further provide an analysis of the theoretical guarantees of our memory allocation method. To the best of our knowledge, HiMA is the first purpose-built memory allocation framework for hybrid models.

2 Related Work

LLM serving stacks. Modern engines build on iteration-level scheduling [38], paged KV [21], cache-aware scheduling [40], CUDA-VMM-backed KV [29], prefill/decode disaggregation [41, 28, 31], and chunked prefill [1]; all treat the within-engine HBM split between state pools as fixed.

Prefix caching for hybrid LLMs. Pure-attention prefix caches [40, 13, 11, 37] miss the recurrent state at the matched boundary that hybrid models need to resume decoding. Marconi [27] extends prefix caching to hybrid LLMs but fixes the cache size. We adopt SGLang's hybrid primitives (`MambaRadixCache`, `HiMambaRadixCache` with host-DRAM/disk tier-

ing, `strip_thinking_cache` [40]) and add a loss-per-byte eviction key (Section 3.1) on top. Concurrent PrfaaS [30] reads the same hybrid-state asymmetry to disaggregate prefill across datacenters but fixes each engine's HBM split; HiMA is complementary on the within-engine axis.

Hybrid memory abstractions and cross-pool budgeting. Hybrid models combining attention with linear/state-space layers — Mamba [14, 8], DeltaNet [36], Jamba [22], Kimi Linear [25], Nemotron 3 Super [26], Olmo Hybrid [24] — motivate this work. Existing within-engine budgeters stop short of cross-pool reallocation: vAttention [29] resizes one pool only; MorphServe [33] trades precision for KV space; Jenga [39] uses a fixed fallback order across paged KV, SSM, and multimodal pools; vLLM v1's hybrid coordinators [21, 34] fix the partition at startup. Cluster-scale KV migration [16, 23] and hardware-side reallocation [18] are orthogonal. HiMA's Lagrange equalization across per-pool marginal-utility curves extends proportional-fairness pricing [20] and database/hypervisor memory tuning [35, 32] to LLM state pools; to our knowledge it is the first to dynamically reallocate physical HBM across hybrid pools (paged KV and recurrent) within a single engine, backed by CUDA VMM.

3 Key Techniques

Section 3.1 defines the *intra-pool* layer: when a pool needs to free space, which cached block should we drop first? We propose LPB to score each block by the GPU time needed to recompute it on a future hit, divided by its size in bytes. The lower-scoring blocks will be dropped first. LPB unifies the KV blocks and recurrent snapshots in the same queue, so the eviction order is well-defined regardless of block type. Section 3.2 and Section 3.3 introduce the *inter-pool* layer: when a request arrives, what action do we take? The choice is among five candidates — admit into free space (own or transferred from the other pool), evict cached blocks to make room (in the target or other pool), or defer — each with a different cost. We pose the decision as two coupled convex programs on separated timescales: a slow *Budgeter* **(B)** that re-balances total HBM between pools, and a fast per-arrival *Admitter* **(A)** that picks among the five candidates. Both reference the same shadow price ν^* (the marginal value of one HBM byte) so their decisions stay consistent (Fig. 2).

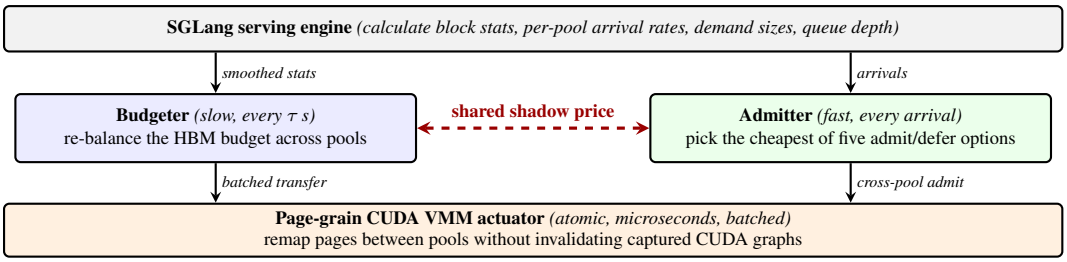


Figure 2: Two-loop architecture of the inter-pool layer. The Budgeter and Admitter share the page-grain VMM actuator below; they are coupled by Theorem 1 via the Lagrange multiplier ν^*.

3.1 Intra-Pool: Loss-per-Byte Eviction

A single LRU list mixes KV blocks (allocated per token, growing with prompt length) with recurrent-state snapshots (one fixed-size per request), and recency alone cannot rank them by yield: a byte of recurrent snapshot saves a chunked-scan replay over an entire prefix on hit, while a byte of KV saves only the prefill of one block. We replace the recency key with a per-block *loss-per-byte* score

$$\ell(b) \;=\; n_b\, c_{i(b)}(s_b)/B_b, \tag{1}$$

where B_b is the block's bytes, n_b counts its hits over a sliding window, s_b is block b's recovery length in tokens (its depth in the prefix tree), and $c_{i(b)}$ is pool $i(b)$'s recovery cost curve (parameterized in Section 3.3); $\ell(b)$ is in units of ms-of-recovery per byte-evicted. Eviction sorts cached blocks in ascending order of $\ell(b)$, the same per-byte loss the inter-pool eviction knapsack of Section 3.3 consumes, so the intra-pool ordering is already in the units the inter-pool layer reads (Fig. 9 in the appendix shows an LRU vs. LPB example). n_b must be *path-counted* (every access increments the deepest match *and every ancestor*); see Appendix C.2 for the failure mode under deepest-only counts.

3.2 Inter-Pool: Coupled Convex Programs

Setup. Let $\mathcal{I}$ denote the set of pools sharing an HBM budget M; we instantiate $\mathcal{I} = \{\text{KV}, \text{rec}\}$ for paged attention vs. recurrent state, and the formulation extends to any $|\mathcal{I}| \geq 2$ (Appendix B.3.2). Each pool $i \in \mathcal{I}$ holds $m_i \geq m_i^{\min}$ mapped bytes; the floor $m_i^{\min}$ is fixed at boot so captured CUDA graphs remain valid (Appendix B.1). Each mapped byte is *live* (pinned by an in-flight request), *cached* (releasable on demand), or *free*, partitioning $m_i = m_i^{\text{live}} + m_i^{\text{cached}} + m_i^{\text{free}}$. A request r arrives with target pool $i_{\text{dst}}(r)$ (fixed by the model architecture) and demand $X(r)$ bytes; we write i_{src} for the other pool. There are five outcomes for r: four *admit* actions (own-free at zero cost, own-evict at $c_{i_{\text{dst}}}^{\text{evict}}$, cross-free at transfer overhead c^{xfer}, cross-evict at $c_{i_{\text{src}}}^{\text{evict}} + c^{\text{xfer}}$) and one *defer* action that queues r at SLO penalty $Q\, w_q$, where w_q is the per-(queued-request) wait penalty in ms. $c_i^{\text{evict}}(X, \mathbf{m})$ is the eviction loss of the cheapest cached blocks freeing X bytes from pool i (own-evict uses $i = i_{\text{dst}}$, cross-evict uses $i = i_{\text{src}}$); $c^{\text{xfer}}(X)$ is the actuator's measured transfer wall-time, runtime-calibrated (Section 3.3). **(A)** picks the cheapest among the five candidates; Fig. 7 in the appendix illustrates the layout.

Per-pool loss. A pool starved of HBM pays in two ways. (i) If the cache cannot retain a block, a later hit forces *recovery* of the missing state — attention re-prefill for KV, chunked-scan replay for recurrent — at GPU wall-time $c_i(\bar{s}_i)$ ms, where $\bar{s}_i$ is the mean recovery length. (ii) If all admission slots are occupied, the arrival is *deferred* at per-arrival cost $Q\, w_q$ (backlog Q, per-tick penalty w_q). With $\mu_i(m_i)$ the per-arrival miss probability (cache under IRM) and $\beta_i(m_i)\, w_q$ the mean per-arrival defer cost (Erlang-B blocking scaled by stationary queue mean, Appendix C.1) at arrival rate λ_i, the expected loss per second on pool i is

$$L_i(m_i) \;=\; \underbrace{\lambda_i\, \mu_i(m_i)\, c_i(\bar{s}_i)}_{\text{recovery}} \;+\; \underbrace{\lambda_i\, \beta_i(m_i)\, w_q}_{\text{deferral}} \quad \text{(ms/s)}. \tag{2}$$

We model μ_i via the Che characteristic-time approximation [5, 10, 3] and β_i via Erlang-B blocking composed with the stationary queue mean [9, 19]; closed forms in Appendix C.1. Both decrease in m_i — a larger cache misses less, a larger pool defers less. Erlang-B blocking is convex in capacity [17] and the Che miss probability is convex under standard popularity tails [12], so L_i is convex non-increasing in m_i, with non-increasing marginal price $\pi_i(m) \triangleq -L_i'(m) \geq 0$ (Lemma 1).

Lemma 1 (Convexity of L_i)**.** *With telemetry inputs $\lambda_i, \bar{s}_i, w_q$ held fixed within a Budgeter tick, L_i is non-increasing convex on $[m_i^{\min}, M]$ with non-increasing marginal price $\pi_i = -L_i' \geq 0$. Proof in Appendix C.2.*

Two coupled programs. With $\mathbf{m} = (m_i)_{i \in \mathcal{I}}$, the slow-timescale **(B)** allocates HBM across pools by minimizing total expected loss:

$$\textbf{(B)}: \quad \min_{\mathbf{m}} \sum\nolimits_i L_i(m_i) \;\; \text{s.t.} \;\; \sum\nolimits_i m_i \leq M, \;\; m_i \geq m_i^{\min}.$$

(B) is separable convex, so under strict feasibility ($\sum_i m_i^{\min} < M$) strong duality gives a unique Lagrange multiplier $\nu^* \geq 0$ on the budget constraint — equivalently, the *shadow price* of one HBM byte (ms/byte), i.e., the marginal reduction in total loss from one extra byte of budget — and a KKT water-filling solution: $\pi_i(m_i^*) = \nu^*$ on interior pools, $\pi_i(m_i^{\min}) \leq \nu^*$ on boundary pools.

The fast-timescale **(A)** operates per arrival. When r arrives with demand $X = X(r)$ at queue depth Q, **(A)** picks the cheapest action $a^*(r)$ from the five candidates above — the four admits plus defer — collected as $\mathcal{A} = \{\text{own-free, own-evict, cross-free, cross-evict, defer}\}$; let $\mathcal{A}(r) \subseteq \mathcal{A}$ drop the free actions when their pool's $m_i^{\text{free}} < X$:

$$\textbf{(A)}: \quad a^*(r) \in \arg\min_{a \in \mathcal{A}(r)} c_a, \quad (c_a)_{a \in \mathcal{A}} = \big(0,\; c_{i_{\text{dst}}}^{\text{evict}},\; c^{\text{xfer}},\; c_{i_{\text{src}}}^{\text{evict}} + c^{\text{xfer}},\; Q\, w_q\big), \tag{3}$$

where $c_i^{\text{evict}}(X, \mathbf{m})$ is the pool-restricted eviction knapsack of Eq. (7): own- and cross-evict differ only in which pool's cache the knapsack draws from. The cross-evict candidate above is an upper bound: Section 3.3 computes a tighter mixed-pool prefix via a single joint sort, so the implementation pointwise dominates (3) and inherits Theorem 1's regret bound. **(A)** runs in microseconds and is myopic by construction.

Shadow-price bridge. **(A)** and **(B)** share the same optimum: **(A)** is the per-arrival realization of **(B)**'s KKT condition $\pi_i = \nu^*$. Dividing each candidate in Eq. (3) by X converts **(A)** into a per-byte test —

admit iff the cheapest per-byte admit cost $g_i(X;\mathbf{m})$ is at most the per-byte defer cost $Q\,w_q/X$. Under stationary traffic, g_i tracks the *recovery component* $\pi_i^{\text{rec}} \triangleq -c_i(\bar{s}_i)\mu_i'$ of $\pi_i = \pi_i^{\text{rec}} + \pi_i^{\text{def}}$ (admit branches see eviction loss only, not queue-relief), while $Q\,w_q/X$ tracks the full ν^* via the queue equilibrium of Lemma 3; KKT couples $\pi_i^{\text{rec}} + \pi_i^{\text{def}} = \nu^*$ at $\mathbf{m}^*$, so **(A)**'s threshold $g_i \le Q\,w_q/X$ is the right comparand for ν^*-tracking. The residual gap is governed by two stochastic quantities, the per-byte transfer overhead $\bar{c}^{\text{xfer}} \triangleq \mathbb{E}[c^{\text{xfer}}/X]$ and the queue-depth std σ_Q around its fluid mean; Theorem 1 shows it is second-order in both.

Theorem 1 (Per-arrival KKT bridge)**.** *Suppose (A1) Poisson arrivals at rate λ_i with $X(r) \le \varepsilon M$ for small ε, (A2) fluid stationarity, (A3) two-timescale separation $\tau \gg \mathbb{E}[1/\lambda_i]$ (Budgeter period τ, Section 3.3), and (A4) interior optimum, $m_i^* > m_i^{\min}$ for all i. Then the iterate-averaged allocation $\bar{m}_i(T) \triangleq \frac{1}{T}\int_0^T m_i(t)\,dt$ incurs quadratic excess loss*

$$\mathbb{E}\big[\textstyle\sum_i L_i(\bar{m}_i(T)) - \sum_i L_i(m_i^*)\big] \;=\; O\big((\bar{c}^{\text{xfer}})^2\big) + O(\sigma_Q^2). \tag{4}$$

*The bound chains two ingredients: Proposition 1 bounds **(A)**'s shadow-price tracking error linearly in $\bar{c}^{\text{xfer}}$ and σ_Q; convexity of $\sum_i L_i$ at the interior $\mathbf{m}^*$ makes the gradient zero, cancelling the linear term and leaving quadratic residual. Full proof in Appendix C.3.*

The per-arrival Admitter approaches the slow Budgeter's optimum at *quadratic* rate, so its tracking error is small and shrinks fast — halving the transfer overhead $\bar{c}^{\text{xfer}}$ or the queue-depth noise σ_Q cuts the excess loss by $4\times$.

3.3 Inter-Pool: Online Realization

Inputs. Three input categories feed the runtime, each refreshed at a different cadence. *Offline-calibrated constants* — the recovery curves $c_i(s)$ and SLO penalty w_q — are fixed once at deployment. *Engine telemetry* $\{n_b, B_b, s_b, \lambda_i, \bar{X}_i, \bar{s}_i\}$ is EWMA-smoothed and refreshed at each Budgeter tick (the per-block s_b is read directly from the radix tree's depth at no extra cost; pool-mean $\bar{s}_i$ is its EWMA aggregate). The *actuator cost* c^{xfer} is self-calibrated at runtime via a per-tick EWMA over committed transfers, since GPU-contention costs cannot be read off an idle deploy-time probe.

Cost curves. The two pools have structurally distinct recovery costs in s when a request must be retracted in full and resumed from cold (the worst-case the gate optimizes against). KV recovery is full attention re-prefill, fitting a quadratic $c_{\text{KV}}(s) = \kappa_2^{\text{KV}} s^2 + \kappa_1^{\text{KV}} s + \kappa_0^{\text{KV}}$ (s^2 from $QK^\top$/softmax–V, s from per-token projections). Recurrent recovery is chunked-scan replay, fitting the linear $c_{\text{rec}}(s) = \kappa_1^{\text{rec}} s + \kappa_0^{\text{rec}}$ — the recurrent state is all-or-nothing per request, so partial recovery is not available. Empirically these curves cross within typical request lengths (Table 3, Fig. 3); in a regime around the crossover s^* recurrent evictions cost an order of magnitude more than KV, while $\kappa_2^{\text{KV}} s^2$ eventually flips the order at larger s. This regime-switching asymmetry is the empirical reason cross-pool moves are worth pricing — which side is cheap depends on the workload.

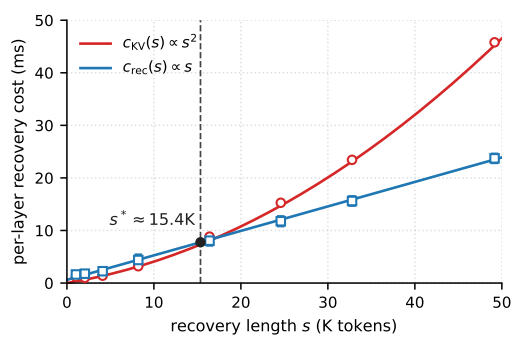


Figure 3: Per-layer cold-start recovery cost $c_i(s)$ on Qwen3.5-35B-A3B / H200 BF16 (n=10 trials per cell). Cross-model fits in Table 3.

Marginal-price estimator. (B) pivots on $\pi_i = -L_i'$, but L_i is defined on miss/defer probabilities rather than observables. Differentiating Eq. (2) via the chain rule and substituting closed forms for μ_i' and β_i' (Appendix C.1) yields the runtime estimator

$$\hat{\pi}_i(m_i) \;=\; -\lambda_i\, c_i(\bar{s}_i)\,\mu_i'(m_i) \;-\; \lambda_i\, w_q\, \beta_i'(m_i), \tag{5}$$

which expresses π_i entirely in cost-curve constants and EWMA-smoothed telemetry.

Solving (B). (B) is separable convex with one budget constraint, so its dual collapses to the scalar ν. Lemma 1 makes π_i non-increasing, so each pool's water-filling target $m_i(\nu) = \max\big(m_i^{\min}, \sup\{m :$

$\pi_i(m) \geq \nu\}\big)$ is monotone in ν, and bisecting ν until $\sum_i m_i(\nu) = M$ converges in $O(|\mathcal{I}| \log(1/\epsilon))$ steps. At runtime the bisection plugs in $\hat{\pi}_i$ from Eq. (5); its fixed point implements

$$\pi_i(m_i^*) \;=\; \nu^* \quad \text{for all } i \text{ with } m_i^* > m_i^{\min} \tag{6}$$

up to the regret bound of Eq. (4).

Solving (A). The pool-restricted knapsack is, for each pool $i \in \{i_{\text{dst}}, i_{\text{src}}\}$,

$$c_i^{\text{evict}}(X, \mathbf{m}) \;=\; \sum\nolimits_{b \in S_i^*(X)} n_b \, c_i(s_b), \tag{7}$$

where $S_i^*(X)$ is the shortest prefix of pool i's cache, sorted ascending by $\ell(b)$, with $\sum_{b \in S_i^*(X)} B_b \geq X$. In implementation we run a single *joint* sort across both pools' caches by $\ell(b)$ and take the shortest prefix $S^*(X)$: if $S^*(X) \subseteq i_{\text{dst}}$ it equals $S_{i_{\text{dst}}}^*(X)$ and recovers the own-evict candidate; if $S^*(X) \subseteq i_{\text{src}}$ it equals $S_{i_{\text{src}}}^*(X)$ and recovers the cross-evict candidate; mixed-pool prefixes (own + partial cross) tighten the cross-evict candidate by paying c^{xfer} only on the cross portion. Free candidates are checked separately by sentinel ($m_i^{\text{free}} \geq X$). As workload composition shifts (as the bulk of cached blocks' s_b crosses the per-pool crossover s^*), the $c_{i(b)}(s_b)$ values reorder $S^*(X)$ and the transfer overhead moves to the now-cheaper side without an explicit direction conditional.

Atomic actuator. The cross-pool actuator must be cheap (small c^{xfer}) and atomic (commit all n pages or none). Two mechanisms deliver this. (i) A *page-grain VMM substrate* remaps virtual pages between pools at CUDA VMM granularity: each `cuMemUnmap`/`cuMemMap` is a μs-scale syscall, and an allocator placement bias keeps each pool's tail densely free so the actuator never defragments. (ii) A *planner/executor split* has the planner emit a transfer plan whose invariants pre-establish atomicity; the executor runs it with no decision points. Physical handles never move; only the virtual mapping changes. Full design and figure in Appendix B.

SGLang instantiation. HiMA ships as a wrapper package leaving SGLang itself unmodified; the Budgeter, Admitter, and actuator all plug underneath existing scheduler hooks (Appendix B.3.3).

Table 1: Main result on Qwen3.5-35B-A3B: HiMA vs. baselines across three agent-serving workloads (long-horizon, swarm, traffic perturbation).

workload	metric	SGLang default	SGLang static-best	vLLM	**HiMA (ours)**
long-horizon	out-TPS	954 ± 10	954 ± 10	771 ± 4	$\mathbf{1{,}025 \pm 50}$
	P99 TTFT (ms)	$7{,}407 \pm 97$	$7{,}349 \pm 179$	$9{,}499 \pm 1{,}041$	$\mathbf{5{,}555 \pm 200}$
swarm	out-TPS	$3{,}518 \pm 23$	$3{,}501 \pm 28$	$2{,}917 \pm 15$	$\mathbf{4{,}990 \pm 100}$
	P99 TTFT (ms)	$75{,}337 \pm 325$	$75{,}756 \pm 800$	$64{,}226 \pm 324$	$\mathbf{33{,}900 \pm 800}$
traffic perturbation	out-TPS (avg A,B,C)	951 ± 50	$1{,}003 \pm 60$	850 ± 45	$\mathbf{1{,}847 \pm 80}$
	out-TPS (Phase-C)	380 ± 15	400 ± 18	350 ± 20	$\mathbf{923 \pm 35}$
	P99 TTFT (avg, ms)	$8{,}005 \pm 403$	$7{,}812 \pm 307$	$9{,}203 \pm 498$	$\mathbf{2{,}917 \pm 203}$
	P99 TTFT (Phase-C, ms)	$8{,}517 \pm 359$	$8{,}204 \pm 320$	$10{,}031 \pm 603$	$\mathbf{2{,}419 \pm 156}$

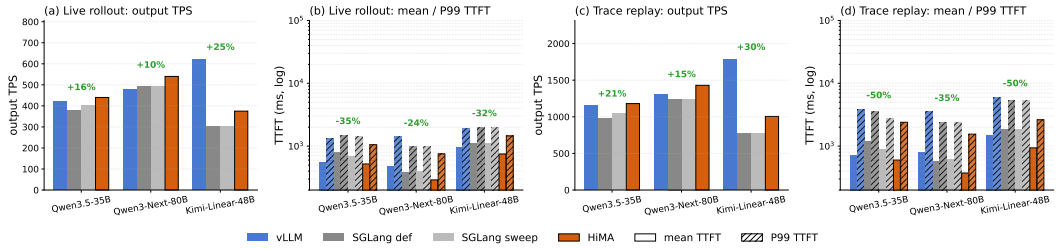


Figure 4: HiMA vs. baselines on two modes of Claude Code workloads: Live rollout and Trace replay.

4 Evaluation

We evaluate HiMA along the following research questions: **RQ1: overall efficiency improvement** Does HiMA outperform SOTA inference engines across hybrid model families in long-horizon agents and agent swarm agents? (Section 4.1) **RQ2: Ablation studies** Do both layers (intra-pool LPB, inter-pool transfer-trigger) contribute to the overall efficiency improvement? (Section 4.2)

RQ3: Verification of our design assumptions Do our two design assumptions hold in practice? (Section 4.3) We further validate HiMA's robustness against key hyper-parameters (Appendix D.4) and its quality preservation across four orthogonal probes (Appendix D.5) in the Appendix. We defer the experimental setup (e.g., hardware, models, baselines, workloads, and metrics) to Appendix D.1.

4.1 RQ1: End-to-end Comparison

We compare HiMA against two SGLang baselines (default, and the per-workload static-best partition) and vLLM v0.20.0. We use four workloads: the three synthetic ones (long-horizon, swarm, traffic perturbation) on Qwen3.5-35B-A3B; the real-world Claude Code agent workload (live rollout on SWE-Bench-Pro and trace replay).

Synthetic workloads. Table 1 shows the results on three synthetic tasks, where HiMA significantly beat all baselines. Specifically, on the swarm workload HiMA lifts output throughput $\sim 42\%$ and reduces P99 TTFT $\sim 55\%$ over default; traffic perturbation sees $\sim 94\%$ TPS and $\sim 64\%$ P99 reduction. In comparison, the long-horizon's gain is smaller, $+7\%$ TPS and -25% P99 (Table 1). This is because the swarm fully saturates the recurrent pool, traffic perturbation triggers reallocation at each transition, while long-horizon queues stay shallow at $N = 14$. We also conduct this experiment on more models, i.e., Qwen3-Next-80B-A3B and Kimi-Linear-48B-A3B, and the results are in Appendix D.2.

Real-world agent. As shown in Fig. 4, the improvement is at a similar level to the long-horizon tasks in synthetic workloads, as the task is naturally long-horizon. In real execution, the improvement is not as significant. This is because the tool calling process also takes a large portion of the execution time. The improvement in replay is more noticeable, demonstrating the gains introduced by HiMA on the model serving side. Note that vLLM beats SGLang in this setting, particularly on Kimi-Linear. This reflects engine-level integration gaps orthogonal to memory management: vLLM v1's async-scheduling pipeline overlaps scheduler iterations with the previous batch's forward pass, and SGLang's Kimi-Linear path is in progress and currently lacks prefix-cache reuse for the linear-attention state. Both are out of HiMA's scope; HiMA inherits SGLang's data-plane and adds memory-management on top.

4.2 RQ2: Ablation Studies

We consider the following variations using the synthetic workloads from RQ1: default SGLang; intra-only, which adds LPB on the engine's prefix cache; inter-only, which adds the cross-pool memory management on original LRU; and the full system. Below, we discuss their performance under three situations: long-horizon agents, agent swarm, and traffic perturbation.

Table 2: Per-cell ablation on Qwen3.5-35B-A3B / H200 / TP=1, 5-trial variance. Long-horizon (KV-bound) and swarm (recurrent-bound) workloads.

cell	TTFT mean (ms)	TTFT P99 (ms)	E2E mean (ms)	out TPS	reqs	transfers
Long-horizon agent (KV-bound)						
Default	$65{,}002 \pm 12{,}207$	$109{,}206 \pm 42{,}555$	$70{,}252 \pm 13{,}000$	833 ± 152	100	0
Intra only	$64{,}010 \pm 10{,}805$	$111{,}059 \pm 45{,}224$	$69{,}714 \pm 12{,}325$	835 ± 150	100	0
Inter only	$61{,}443 \pm 7{,}819$	$\mathbf{83{,}243 \pm 7{,}301}$	$67{,}564 \pm 9{,}318$	861 ± 113	106	1.2
HiMA	$\mathbf{60{,}757 \pm 7{,}413}$	$\mathbf{81{,}741 \pm 7{,}267}$	$\mathbf{64{,}685 \pm 7{,}791}$	$\mathbf{895 \pm 93}$	**109**	1.6
Agent swarm (recurrent-bound)						
Default	$5{,}840 \pm 432$	$18{,}250 \pm 1{,}560$	$14{,}920 \pm 318$	$10{,}420 \pm 215$	7,830	0
Intra only	$5{,}612 \pm 410$	$17{,}890 \pm 1{,}480$	$14{,}680 \pm 305$	$10{,}610 \pm 200$	7,975	0
Inter only	$3{,}712 \pm 218$	$\mathbf{8{,}640 \pm 412}$	$9{,}820 \pm 184$	$\mathbf{14{,}560 \pm 242}$	**11,012**	6.2
HiMA	$\mathbf{3{,}528 \pm 196}$	$\mathbf{8{,}210 \pm 386}$	$\mathbf{9{,}560 \pm 172}$	$\mathbf{14{,}780 \pm 228}$	**11,182**	6.2

Long-horizon agent workload: inter-pool layer dominates. We start with the KV-bound side of Fig. 1. We run N=14 concurrent agent sessions on Qwen3.5-35B-A3B / H200. Each session sends $\sim$4 K-token turns and accumulates conversation history into the next request, resetting once the context reaches 60 K tokens. Steady-state KV demand is ~ 840 K tokens against the ~ 880 K-token KV pool; the recurrent-state pool sits at $\sim$14 of 384 slots. The KV pool binds. Table 2 reports per-cell results. The intra-only cell tracks baseline (no shared-prefix structure for LPB to exploit). The two inter-on cells reduce P99 TTFT by $\mathbf{-25.1}\%$ and lift output throughput by $\mathbf{+7.4}\%$. Each trial

averages 1.2 inter-pool transfers (full system: 0.6), each moving ∼30 chunks of capacity between pools and shrinking the queue depth that drives the long tail.

Agent swarm workload: inter-pool layer dominates the dual direction. We run N=800 concurrent short multi-turn sub-agent sessions, each issuing $\leq$ 256/$\leq$ 256 in/out tokens for 2–5 turns before resetting at a 3 K context cap. The concurrency is set to $\sim 2\times$ the recurrent-state pool's 384-slot ceiling, so $\sim$ 416 sub-agents queue against admission at steady state; KV stays at 3–30% fill. The recurrent pool binds—the dual of the long-horizon workload. As shown in Table 2, HiMA improves output throughput by $+\mathbf{41.8}\%$ and reduces P99 TTFT by $-\mathbf{55.0}\%$. Each trial averages $\sim$ 6.2 inter-pool transfers, lifting the recurrent admission ceiling from 384 to $\sim$ 700 slots within a few Budgeter ticks; the queued sub-agents drain into the new capacity, eliminating most of the admission tail. The intra-pool layer contributes a further $\sim$ 2–4% via LPB protection of the orchestrator's shared system-prompt snapshot under rapid sub-agent creation and stopping.

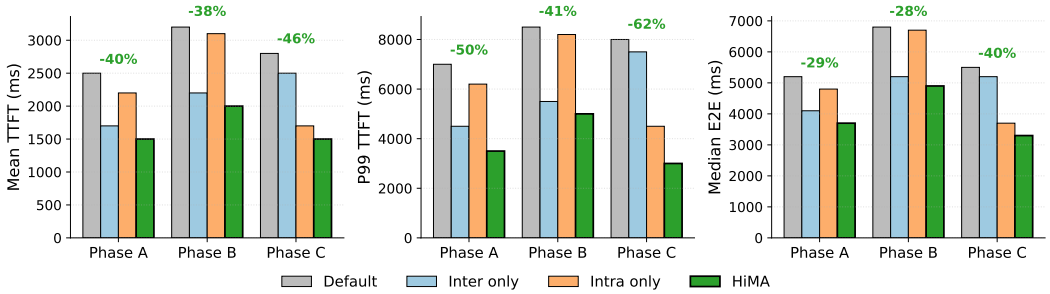


Figure 5: Traffic perturbation trace, ablation on Qwen3.5-35B-A3B.

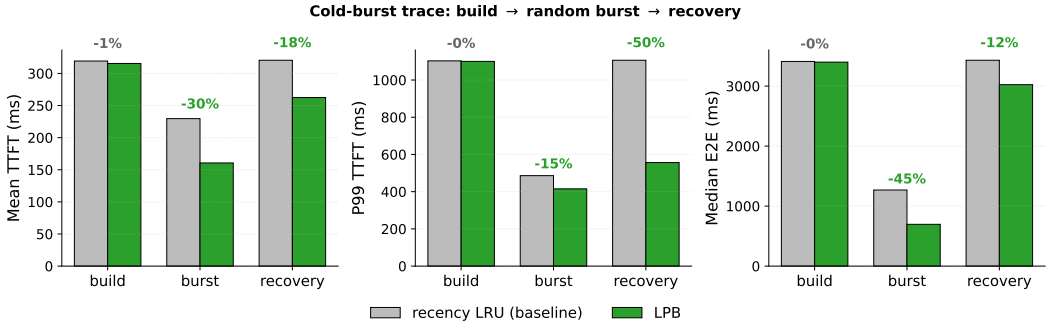


Figure 6: LPB vs. recency-LRU across the cold-burst trace.

Traffic perturbation: both layers contribute across workload transitions. The two workloads above bind a fixed pool throughout. We now stress the *transition*: the trace concatenates a saturating swarm workload (Phase A: GSP shared-prefix, 100×16 prompts at RPS=12, driving the recurrent pool to $\geq$ 0.95 utilization), a saturating long-horizon workload (Phase B: random 12 K-token prompts at RPS=8, KV-bound with a deep admission queue), and a return to the Phase A pattern (Phase C: GSP at RPS=12, the post-perturbation recovery test). Fig. 5 reports per-phase results. Both layers contribute: the inter-pool actuator fires kv↔rec transfers at every phase boundary so each phase reaches its binding pool's full capacity, and the intra-pool LPB key protects Phase A's shared-prefix snapshots through the Phase B burst so Phase C recovers them on cache hit rather than re-prefilling.

Phase C's -30% P99 gap is $\sim 3.4\sigma$ on the intra-only cell at n=3; on the joint cell elevated trial variance (σ=276 ms) reduces this to $\sim 1.3\sigma$, but the median Δ is -29% and all three trials move in the same direction. Phase A's $+10\%$ regression exceeds 1σ but is not strongly significant at n=3.

4.3 RQ3: Verification of our design assumptions

Cold-burst LRU collapse and LPB repair. We test whether strict-recency LRU evicts high-value shared-prefix nodes during a cold burst, and whether LPB preserves them. The workload runs in three phases on Qwen3.5-35B-A3B with the SGLang generated-shared-prefix benchmark (12 K-token system prompt): Phase 1 (build) 80 shared-prefix prompts at RPS=2; Phase 2 (burst) 200 random

4 K-token un-shared prompts at RPS=8; Phase 3 (recovery) repeats Phase 1. Both arms use the same engine config; only the eviction key differs.

Recency and LPB hit cached batches at nearly identical rates (43.6% vs. 45.2%), but LPB keeps the 12 K-token shared-prefix snapshot alive through the burst while recency evicts it as oldest. In Phase 3, LPB hits the snapshot at depth >8 K and avoids re-prefilling the system prompt. Burst-phase mean TTFT drops 30% (160.6 vs. 229.6 ms) and median E2E drops 45% (696 vs. 1268 ms); recovery-phase mean TTFT drops 18% (262.5 vs. 320.5 ms) and P99 drops 50% (556.3 vs. 1106.2 ms) (Fig. 6).

Table 3: Per-layer $c_{\text{KV}}(s)$ and $c_{\text{rec}}(s)$ coefficients on three hybrid models on H200 ($n = 10$ per cell, $s \in \{1, 2, 4, 8, 16, 24, 32, 48, 64, 96\}$ K). Crossover s^* is the smallest $s > 0$ with $c_{\text{KV}}(s) = c_{\text{rec}}(s)$.

model	config	κ_2^{KV}	κ_1^{KV}	κ_0^{KV}	κ_1^{rec}	κ_0^{rec}	s^*
Qwen3.5-35B-A3B	TP=1 (10 attn / 30 rec)	1.32×10^{-8}	2.65×10^{-4}	0.78	4.66×10^{-4}	0.61	14 400
Qwen3-Next-80B-A3B	TP=2 (12 attn / 36 rec)	4.19×10^{-9}	3.33×10^{-4}	−0.11	2.63×10^{-4}	1.87	14 900
Kimi-Linear-48B-A3B	TP=1 (7 attn / 20 rec)	1.46×10^{-8}	4.58×10^{-4}	0.86	7.42×10^{-4}	1.12	20 300

Recovery-cost asymmetry: empirical $c_i(s)$ and crossover s^*. We measure the recovery-cost curves $c_i(s)$ used by the inter-pool layer's cross-pool eviction decision (Section 3.2) and check that they cross at a workload-relevant length s^* on three hybrid models. $c_{\text{KV}}(s)$ is quadratic in s (full attention is $O(s^2)$ since token i attends to all i preceding KV pairs) and $c_{\text{rec}}(s)$ is linear ($O(s)$ per token); below s^* a KV byte is cheaper to evict, above s^* a recurrent byte is.

For each layer type $i \in \{\text{KV}, \text{rec}\}$ and each prefix length $s \in \{1, 2, 4, 8, 16, 24, 32, 48, 64, 96\}$ K tokens, we time the cold-start re-prefill of a single request around each decoder-layer iteration; $c_i(s)$ is the mean elapsed time of one layer of type i over $n = 10$ trials per cell. We fit κ by least squares on three hybrid models (Table 3). All three exhibit quadratic KV and linear recurrent and cross within 14–20 K token length. To verify the inter-pool layer uses these fitted-curves correctly at runtime, we step $\bar{s}$ through $\{2, 8, 14, 64\}$ K tokens at fixed request rate; its chosen eviction direction tracks the parametric prediction at all four steps (KV-side below s^*, recurrent-side above).

5 Limitations and Conclusion

We propose HiMA, a hierarchical memory management system for hybrid models, where the intra-pool layer defines a new metric for eviction and the inter-pool layer introduces a memory allocation method. We implement HiMA as a plugin to SGLang and show that it efficiently reduces the latency of serving hybrid models, especially under agentic setups. We further provide a theoretical analysis of our proposed method. The limitations of HiMA point to a few promising directions for future work. HiMA can be extended to additional pool types, e.g., separate pools for register-pressure-coupled state abstractions, or host-DRAM tiers reached over PCIe. However, this requires redesigning our allocation method, as they break some of our assumptions **(B)**. Our method can also be instantiated as a plugin to vLLM, as the design is modularized and orthogonal to the serving frameworks.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. 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)*, pages 117–134, 2024. arXiv:2403.02310.

[2] Albert Benveniste, Michel Métivier, and Pierre Priouret. *Adaptive Algorithms and Stochastic Approximations*, volume 22 of *Applications of Mathematics*. Springer, 1990.

[3] Daniel S. Berger, Ramesh K. Sitaraman, and Mor Harchol-Balter. Practical bounds on optimal caching with variable object sizes. In *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, volume 2, pages 32:1–32:38, 2018.

[4] Vivek S. Borkar. *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge University Press and Hindustan Book Agency, 2008.

[5] Hao Che, Ye Tung, and Zhijun Wang. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE Journal on Selected Areas in Communications*, 20(7):1305–1314, 2002.

[6] Jorge Cortés. Discontinuous dynamical systems: A tutorial on solutions, nonsmooth analysis, and stability. *IEEE Control Systems Magazine*, 28(3):36–73, 2008.

[7] Jim G. Dai. On positive Harris recurrence of multiclass queueing networks: A unified approach via fluid limit models. *The Annals of Applied Probability*, 5(1):49–77, 1995.

[8] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. arXiv:2405.21060.

[9] A. K. Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *Elektroteknikeren*, 13:5–13, 1917.

[10] Ronald Fagin. Asymptotic miss ratios over independent references. *Journal of Computer and System Sciences*, 14(2):222–250, 1977.

[11] 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. In *2024 USENIX Annual Technical Conference (ATC)*, pages 111–126, 2024. arXiv:2403.19708.

[12] Michele Garetto, Emilio Leonardi, and Valentina Martina. A unified approach to the performance analysis of caching systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 1(3):12:1–12:28, 2016.

[13] 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 (MLSys)*, 2024.

[14] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling (COLM)*, 2024. arXiv:2312.00752.

[15] J. Michael Harrison. *Brownian Motion and Stochastic Flow Systems*. Wiley, 1985.

[16] Yiyuan He, Minxian Xu, Jingfeng Wu, Jianmin Hu, Chong Ma, Min Shen, Le Chen, Chengzhong Xu, Lin Qu, and Kejiang Ye. BanaServe: Unified KV cache and dynamic module migration for balancing disaggregated LLM serving in AI infrastructure. *arXiv preprint arXiv:2510.13223*, 2025.

[17] A. A. Jagers and Erik A. van Doorn. On the continued Erlang loss function. *Operations Research Letters*, 5(1):43–46, 1986.

[18] Alish Kanani, Sangwan Lee, Han Lyu, Jiahao Lin, Jaehyun Park, and Umit Y. Ogras. DUET: Disaggregated hybrid mamba-transformer LLMs with prefill and decode-specific packages. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference (DAC)*, 2026. arXiv:2603.15530.

[19] F. P. Kelly. *Reversibility and Stochastic Networks*. Wiley, 1979.

[20] Frank P. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8(1):33–37, 1997.

[21] 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 (SOSP)*, pages 611–626, 2023. arXiv:2309.06180.

[22] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avshalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

[23] 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.

[24] William Merrill, Yanhong Li, Tyler Romero, Anej Svete, Caia Costello, Pradeep Dasigi, Dirk Groeneveld, David Heineman, Bailey Kuehl, et al. Olmo Hybrid: From theory to practice and back. *arXiv preprint arXiv:2604.03444*, 2026.

[25] Moonshot AI. Kimi Linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

[26] NVIDIA. Nemotron 3 Super: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2604.12374*, 2026.

[27] Rui Pan, Zhuang Wang, Zhen Jia, Can Karakus, Luca Zancato, Tri Dao, Yida Wang, and Ravi Netravali. Marconi: Prefix caching for the era of hybrid LLMs. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025.

[28] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132, 2024. arXiv:2311.18677.

[29] 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 (ASPLOS)*, 2025. arXiv:2405.04437.

[30] Ruoyu Qin, Weiran He, Yaoyu Wang, Zheming Li, Xinran Xu, Yongwei Wu, Weimin Zheng, and Mingxing Zhang. Prefill-as-a-service: KVCache of next-generation models could go cross-datacenter. arXiv:2604.15039, 2026. Accessed 2026-05-03.

[31] 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)*, pages 155–170, 2025. Best Paper Award. arXiv:2407.00079.

[32] Adam J. Storm, Christian Garcia-Arellano, Sam Lightstone, Yixin Diao, and Maheswaran Surendra. Adaptive self-tuning memory in DB2. In *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, pages 1081–1092, 2006.

[33] Zhaoyuan Su, Zeyu Zhang, Tingfeng Lan, Zirui Wang, Haiying Shen, Juncheng Yang, and Yue Cheng. MorphServe: Efficient and workload-aware LLM serving via runtime quantized layer swapping and KV cache resizing. *arXiv preprint arXiv:2506.02006*, 2025.

[34] vLLM Project. [RFC] hybrid memory allocator (issue #11382). `https://github.com/vllm-project/vllm/issues/11382`, 2024. Accessed 2026-05-03.

[35] Carl A. Waldspurger. Memory resource management in VMware ESX server. In *5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[36] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*, 2024. arXiv:2406.06484.

[37] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, pages 94–109, 2025.

[38] 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)*, pages 521–538, 2022.

[39] Chen Zhang, Kuntai Du, Shu Liu, Woosuk Kwon, Xiangxi Mo, Yufeng Wang, Xiaoxuan Liu, Kaichao You, Zhuohan Li, Mingsheng Long, Jidong Zhai, Joseph E. Gonzalez, and Ion Stoica. Jenga: Effective memory management for serving LLM with heterogeneity. *arXiv preprint arXiv:2503.18292*, 2025.

[40] 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 37 (NeurIPS)*, 2024. arXiv:2312.07104.

[41] 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)*, pages 193–210, 2024. arXiv:2401.09670.

A Reproducibility

The HiMA source, benchmark harness, raw bench logs, exact baseline engine commits with build flags, the traffic perturbation trace synthesis recipe, and per-experiment hardware details are released at an anonymized URL to be revealed on acceptance: `https://anonymous.4open.science/r/hima`.

B Implementation Details

B.1 Definitions and invariants

We first introduce the definitions and invariants used throughout the rest of Appendix B: how each pool is laid out physically, the state every page can be in, and the conditions under which captured CUDA graphs remain valid.

Virtual-address (VA) overcommit and physical handles. For each pool i we call `cuMemAddressReserve`(c_i^{cap}) where c_i^{cap} is generously oversized — typically ∼80–90% of the engine's available HBM each, so $\sum_i c_i^{\text{cap}} > M$. VA reservation is virtual and consumes no physical bytes; each pool's tensors are constructed wrapping $[A_i, A_i + c_i^{\text{cap}})$, fixing their data pointer and shape at boot. Physical capacity is supplied separately by `cuMemCreate`, which produces uniform handles of size P totaling M at boot. We set $P = 2\,\text{MiB}$ — CUDA VMM's native page size on the target hardware — since coarser handles raise the chance that a single live block pins the entire handle and blocks migration.

Mapped state. A handle becomes usable by kernels only after `cuMemMap` attaches it to a VA. Let m_i^{mapped} be the bytes currently mapped in pool i and $\sum_i m_i^{\text{mapped}} \leq M$ at all times; a cross-pool transfer just moves a handle from one pool to another.

Page-id ownership state. On top of the physical mapping we define a *logical* state that the planner reasons about. Let $\mathcal{P}_i$ be the set of mapped pages in pool i. At any instant, every $p \in \mathcal{P}_i$ is in one of four states, partitioning $\mathcal{P}_i$:

$$\text{owner}_i(p) \in \{\textsc{free}, \textsc{cached}, \textsc{live}, \textsc{capped}\}, \tag{8}$$

refining at page granularity the byte-level partition of Section 3.2: FREE pages sit in the allocator's free list; CACHED pages back a node in the prefix-cache tree (releasable on demand); LIVE pages are bound by an in-flight request's index tensor; and CAPPED pages have been withheld by the executor's cap-barrier and are not allocatable (no analog in the byte-level partition — introduced by the actuator). The four-set partition is the planner's input; every primitive below is a state transition on this partition. *Drain* transitions CACHED → FREE (then FREE → CAPPED via the barrier). *Migration* transitions LIVE → FREE via in-place rebinding of the request's index tensor. *Cap-barrier* transitions FREE → CAPPED. *Unmap*, applied only after every page in the chunk is in CAPPED, removes the underlying physical handle.

CUDA-graph invariants. Captured graphs reference (i) pool buffer base addresses, fixed at boot for the lifetime of the engine, and (ii) request index tensors (`req_to_token`), allocated once per request slot and mutable in place. Cross-pool transfer rewrites entries in (ii) when migrating active blocks;

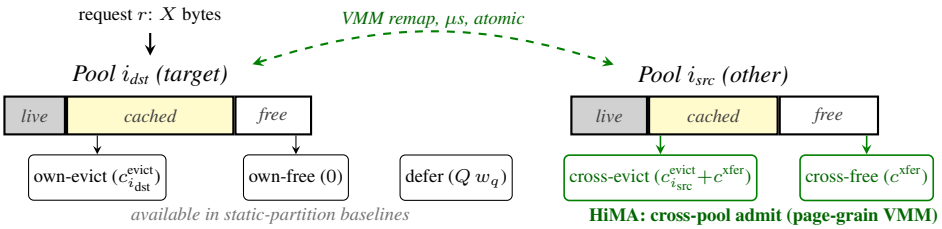


Figure 7: Five outcomes for an arrival to pool i_{dst}: four admit candidates (own/cross $\times$ free/evict) plus defer. The two cross-pool candidates (green) are unique to HiMA; static baselines must defer once i_{dst} is exhausted. **(A)** prices all five on a common ms-per-byte scale and admits the cheapest.

the graph reads the updated indices on next replay, picking up fresh page-ids in the un-capped region. No graph re-capture is needed: migration rewrites `req_to_token` before unmap, and the cap-barrier blocks any new alloc from landing in the to-be-unmapped range, so no captured pointer ever resolves to a CAPPED page.

B.2 Transfer Planning

B.2.1 Page selection

This section describes how the planner picks which n pages of $\mathcal{P}_{i_{\text{src}}}$ to unmap when a transfer is requested.

Allocator placement bias. Each pool's block allocator runs first-fit lowest-address: new live allocations bias toward the head of the pool's mapped region (near A_i), so under sustained traffic the tail accumulates FREE pages. The allocator never compacts.

Anywhere-free selection. When a transfer of n pages from i_{src} to i_{dst} is requested, the planner walks $\mathcal{P}_{i_{\text{src}}}$ and picks any n pages currently in FREE state, sorted by descending page-id (so the picked pages naturally concentrate at the tail under placement bias, but the planner is not restricted to a contiguous tail block). The availability predicate is simply

$$\texttt{free_ready}_i(n) \;\triangleq\; |\,\text{FREE-pages of } \mathcal{P}_i\,| \;\geq\; n. \tag{9}$$

Selection is per-page: chunk = page under the page-grain VMM (Appendix B.1), so the planner faces no contiguity constraint. If $\texttt{free_ready}_i(n)$ does not hold, the planner refuses to emit a plan and the transfer is deferred to the next tick (Appendix B.2.3); cost-driven extensions that would also consult CACHED and LIVE pages with explicit per-page costs c_e and migration c_m are left to future work.

B.2.2 Per-request triggering

The protocol of Appendix B.3.1 fires only when an arriving request's cheapest action requires a cross-pool transfer; this section specifies that per-request check.

`cuMemMap` and `cuMemUnmap` are issued only on a request's arrival or by the control loop (Appendix B.2.3), never inside compute kernels. On arrival of request r with demand $X = X(r)$ for pool $i_{\text{dst}}(r)$, we directly evaluate program **(A)** of Eq. (3) and choose a^* via Eq. (3). The transfer protocol of Appendix B.3.1 is invoked whenever a^* is a cross-pool candidate (cross-free or cross-evict, both carrying $c^{\text{xfer}}(X)$) — synchronously moving the X bytes from the cheapest i^*_{src} into $i_{\text{dst}}(r)$ before the request is admitted. Otherwise: own-free admits directly at zero cost, own-evict triggers a prefix-cache eviction, and defer enqueues r at the queue cost $Q\,w_q$. By construction the chosen a^* satisfies $\kappa(a^*) \leq Q\,w_q$, so the per-request admission overhead is bounded by the queue baseline.

B.2.3 Periodic triggering and hysteresis

A coarse-grained Budgeter loop runs every τ seconds (default $\tau = 30\,\text{s}$), addressing *steady-state* imbalance complementary to the Admitter's *burst* response. The two timescales share the cost framework but trigger on different signals: the Admitter fires when an individual arrival's Eq. (3) winner contains c^{xfer}, while the control loop fires when sustained workload mix has shifted the optimal allocation $\mathbf{m}^*$. Each tick the loop reads from every pool the EWMA inputs of Eq. (16)

($\{n_b, B_b, s_b\}_b, \lambda_i, \bar{X}_i, \bar{s}_i$) and the current m_i^{mapped}, evaluates $\hat{\pi}_i$ in closed form via Eq. (15), and recomputes the static targets $\mathbf{m}^*$ via the Lagrange bisection of Eq. (6). When the resulting target shift $|m_i^* - m_i|$ exceeds the hysteresis dead-zone of width Δ_{hyst} on at least one pool, the loop issues a batch `transfer_pages`$(i^*_{\text{src}}, i^*_{\text{dst}}, \Delta m^*)$ through the protocol of Appendix B.3.1; otherwise it defers. The stationary regret of the dead-zone is bounded in Lemma 4 of Appendix C.3.

B.2.4 Free-page availability bound

Anywhere-free selection (Appendix B.2.1) refuses to emit a plan when $|\,\text{FREE-pages of } \mathcal{P}_i\,| < n$ (Eq. (9)). We give a steady-state lower bound on the expected free-page count, which shows that under below-saturation operation (assumption A4 below) a transfer of any reasonable magnitude can almost always be served.

Lemma 2 (Free-page expected count)**.** *Consider pool i with $|\mathcal{P}_i|$ pages, one block per page, operating in steady state under the following assumptions:*

- *(A1) Block arrivals to pool i form a Poisson process with rate λ_i blocks/ms.*
- *(A2) Each arriving block occupies a page (in either* LIVE *or* CACHED *state) for a total residency time with i.i.d. mean $\bar{\tau}_i$ (in ms; distinct from the recovery-length symbol $\bar{s}_i$ of Section 3.2, which is in tokens), after which the page returns to* FREE*; we model the pool as $M/G/\infty$ (no admission blocking).*
- *(A3) The block allocator is first-fit-lowest with a sorted free list (Appendix B.2.1).*
- *(A4) Below-saturation operation: $\lambda_i \bar{\tau}_i \le \alpha\,|\mathcal{P}_i|$ for some $\alpha < 1$.*

Then

$$\mathbb{E}[\,|\text{FREE-}\textit{pages of}\ \mathcal{P}_i|\,] \;\ge\; (1-\alpha)\,|\mathcal{P}_i|. \tag{10}$$

Proof sketch. Pages in LIVE $\cup$ CACHED are exactly the non-free pages, i.e. pages currently occupied by some block. Treating each occupancy as a single sojourn from arrival to release of the page, $M/G/\infty$ together with Little's law gives

$$\mathbb{E}[\,|\text{LIVE} \cup \text{CACHED}|\,] \;=\; \lambda_i\,\bar{\tau}_i \;\le\; \alpha\,|\mathcal{P}_i|,$$

where the last inequality is (A4). Hence $\mathbb{E}[\,|\text{FREE-pages of } \mathcal{P}_i|\,] = |\mathcal{P}_i| - \mathbb{E}[\,|\text{LIVE} \cup \text{CACHED}|\,] \ge (1-\alpha)\,|\mathcal{P}_i|$, which is Eq. (10).

The same setup also gives a per-page concentration result used by Corollary 1. Let $q_i(p) := \Pr[\,\text{page } p \text{ is non-free}\,]$ in steady state (the symbol q_i is local to this proof; it is not the queueing ρ_i of Section 3.2). Couple two systems with identical block-arrival times and identical lifetimes, differing only in placement priority: System A uses the actual ordering $1 < 2 < \cdots < |\mathcal{P}_i|$, System B swaps page-ids p and $p+1$ (so $p+1$ is preferred over p). Run both sample-path-by-sample-path. By induction on event times, whenever System B occupies page p at time t, System A has already occupied p (because A always prefers the same or lower id when both choices are available). Hence at every t, $\mathbf{1}[p \text{ occupied in A}] \ge \mathbf{1}[p+1 \text{ occupied in A}]$, so $q_i(p) \ge q_i(p+1)$. Given non-increasing q_i and $\sum_p q_i(p) = \lambda_i \bar{s}_i$, the suffix-of-length-$n$ sum (with $T_n \triangleq \{|\mathcal{P}_i| - n + 1, \ldots, |\mathcal{P}_i|\}$) satisfies

$$\sum_{p \in T_n} q_i(p) \;\le\; \frac{n}{|\mathcal{P}_i|} \sum_{p \in \mathcal{P}_i} q_i(p) \;=\; \frac{n\,\lambda_i \bar{\tau}_i}{|\mathcal{P}_i|} \;\le\; \alpha\, n,$$

using (A4) in the last step. □

Corollary 1 (Plan availability at fixed n)**.** *If transfers use a fixed magnitude $n = n_0$ such that $n_0 \le (1-\alpha)|\mathcal{P}_i|/2$, the planner emits a valid plan with probability approaching 1 as $|\mathcal{P}_i| \to \infty$ (Markov inequality on the free-page deficit).*

In our deployment $\alpha \lesssim 0.7$ and $n_0 \sim 16$, against $|\mathcal{P}_i| \sim 10^5$ pages, so the expected free-page count is $\sim 0.3 \cdot 10^5 \gg n_0$ and plan availability is essentially never the binding constraint.

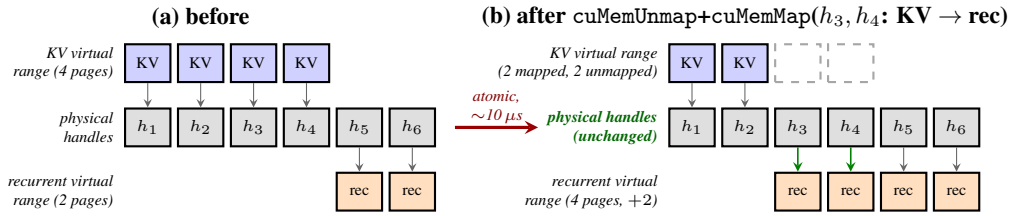


Figure 8: Page-grain CUDA VMM remap of h_3, h_4 from KV to rec: `cuMemUnmap` + `cuMemMap`, atomic. Physical handles don't move; tensor pointers and captured CUDA graphs remain valid because each pool's virtual range stays contiguous.

B.3 Executor

B.3.1 Transfer execution

Once the planner has selected n pages $F = \{p_1, \ldots, p_n\} \subseteq$ FREE via Eq. (9), the executor runs the following three-step protocol on the ownership state of Eq. (8), under a single scheduler lock between two forward steps:

1. *Cap-barrier.* Move every page in F from FREE to CAPPED. After this step no `alloc()` returns any of those pages, even if a concurrent admission demands new capacity.
2. *Verify.* Confirm $F \subseteq$ CAPPED. If any page in F leaked back into FREE (e.g. because of a buggy cap-barrier implementation), restore the cap state and abort; the next planner tick retries with a refreshed ownership map.
3. *Unmap + map.* Issue `cuMemUnmap` on the handles backing F; the freed handles bind to i_{dst} via `cuMemMap`. Pages on i_{dst}'s newly-mapped slots transition to FREE.

The protocol guarantees by construction that every committed transfer moves exactly n pages and that no captured graph dereferences a page mid-transition: cap-barrier forecloses concurrent allocs into F before unmap, and the planner's selection criterion ensures F is disjoint from LIVE and CACHED so no in-flight read or cache reference points into the unmapped range. The three steps are metadata-only; `cuMemUnmap`/`cuMemMap` dominate wall time at µs-level per page on H200.

B.3.2 Instantiation: KV and recurrent pools

The general framework above is $|\mathcal{I}|$-agnostic. A new pool joins by supplying (i) a block allocator that exposes the per-page free-mask in $O(1)$ per page, (ii) the four-set ownership map of Eq. (8), and (iii) a parametric $c_i(s)$ ansatz with one extra Stage-0 probe. The unified rule of Eq. (3) enumerates all $|\mathcal{I}|(|\mathcal{I}| - 1)$ ordered source-destination pairs without per-pool special cases.

We instantiate $|\mathcal{I}| = 2$ for our paged-attention vs. recurrent-state setting: pool 1 holds per-token KV pages (block size = one token of K and V across all attention layers), pool 2 holds per-sequence recurrent slots (block size = one DeltaNet/SSM state across all recurrent layers). The migration step of Appendix B.3.1 differs in the two pools' shape of "block" — a KV migration copies a multi-layer KV slice and rewrites the request's `kv_indices`; a recurrent-slot migration copies the full per-sequence state and rewrites the request's slot pointer — but the protocol, ownership map, and Admitter are identical. Empirical evaluation of $|\mathcal{I}| > 2$ instantiations (e.g., adding a draft-model KV pool for speculative decoding) is left to future work.

B.3.3 Engine integration: SGLang

The Budgeter is an in-process task in the SGLang server, ticking once per window of length τ (default $\tau = 30\,\text{s}$). Telemetry is accumulated continuously inside the window via callbacks on existing scheduler hooks (paged-KV/slot usage, hit rate, queue depth, preemption rate), so no new instrumentation is needed. At each window boundary the Budgeter evaluates $\hat{\pi}_i$ (Eq. (5)) and bisects Eq. (6) for the new target $\mathbf{m}^*$. The actuator plugs underneath each pool's tensor backing without changing the pool object: when $|m_i^* - m_i|$ exceeds the hysteresis dead-zone, the Budgeter issues a single batched `transfer_pages` syscall. The whole thing ships as a wrapper package; SGLang itself is unmodified.

C Allocation Formalism: Derivations and Calibration

This section backs the coupled convex programs of Section 3.2: Appendix C.1 gives the closed-form marginal $\hat{\pi}_i$ promised by Eq. (5) and the EWMA layer that smooths its inputs; Appendix C.2 gives the Stage-0 deployment calibration, the path-counted hit rates that feed the Che/LPB primitives, the runtime self-calibration of c^{xfer}, and the engine-specific admission-pressure adapter that handles the saturated face of the dual; Appendix C.3 backs the two statistical claims Eqs. (19) to (4) of Proposition 1 and Theorem 1 via a closed-loop fluid limit and a hysteresis Taylor expansion.

C.1 Closed-Form Marginal $\hat{\pi}_i$ and EWMA Smoothing

Cache and admission primitives. The two factors of Eq. (2) compose two standard cache-modeling primitives. We model the per-arrival miss probability μ_i via the Che characteristic-time approximation [5, 10, 3]: $T_C(m_i)$ solves $\sum_b B_b(1 - e^{-h_b T_C}) = m_i^{\text{cached}}$ at per-block hit rate h_b, and

$$\mu_i(m_i) \;=\; \big(\textstyle\sum_b h_b\, e^{-h_b T_C(m_i)}\big) \big/ \textstyle\sum_b h_b. \tag{11}$$

We model the per-arrival defer probability β_i via Erlang-B blocking [9, 19] with offered load ρ_i and slot capacity $C_i(m_i) = (m_i^{\text{cached}} + m_i^{\text{free}})/\bar{X}_i$:

$$\beta_i(m_i) \;=\; B(C_i(m_i),\, \rho_i), \quad B(C, \rho) \;=\; \frac{\rho^C/C!}{\sum_{k=0}^{C} \rho^k/k!}. \tag{12}$$

The runtime evaluates $\hat{\pi}_i(m_i) = -L_i'(m_i)$ in closed form by chain rule on Eq. (2) through these two primitives, with L_i' evaluated at the EWMA-smoothed inputs of Eq. (16). The rest of this subsection records the two derivatives and the EWMA layer.

Che miss derivative μ_i'. Implicitly differentiating $\sum_b B_b\,(1 - e^{-h_b T_C(m_i)}) = m_i^{\text{cached}}$ in m_i yields

$$T_C'(m_i) \;=\; \frac{1}{\sum_b h_b\, B_b\, e^{-h_b T_C(m_i)}},$$

and substituting into Eq. (11),

$$\mu_i'(m_i) \;=\; -\,T_C'(m_i) \cdot \frac{\sum_b h_b^2\, e^{-h_b T_C(m_i)}}{\sum_b h_b} \;=\; -\,\frac{\sum_b h_b^2\, e^{-h_b T_C(m_i)}}{\big(\sum_b h_b\big) \cdot \sum_b h_b\, B_b\, e^{-h_b T_C(m_i)}} \;\le\; 0. \tag{13}$$

Erlang-B defer derivative β_i'. With $C_i(m_i) = (m_i^{\text{cached}} + m_i^{\text{free}})/\bar{X}_i$ affine in m_i (Eq. (12)),

$$\beta_i'(m_i) \;=\; \frac{1}{\bar{X}_i} \cdot \frac{\partial B(C_i, \rho_i)}{\partial C_i} \;\le\; 0, \tag{14}$$

where $\partial B/\partial C \le 0$ inherits from the monotone-in-capacity Erlang-B blocking [17]. The runtime evaluates $\partial B/\partial C$ by symmetric finite difference on the integer recurrence $B(C+1, \rho) = \rho\, B(C, \rho)/\big(C + 1 + \rho\, B(C, \rho)\big)$ at $C = \lfloor C_i(m_i) \rfloor$.

Combined marginal. Differentiating Eq. (2) via Eqs. (13) to (14),

$$\hat{\pi}_i(m_i) \;=\; -\,\lambda_i\, c_i(\bar{s}_i)\, \mu_i'(m_i) \;-\; \lambda_i\, w_q\, \beta_i'(m_i), \tag{15}$$

which is the closed form referenced as Eq. (5) in the main text. Both terms are ≥ 0 because $\mu_i', \beta_i' \le 0$, recovering Lemma 1 by direct calculation.

EWMA on the inputs. The five inputs $\{n_b, B_b, s_b\}_b$, λ_i, $\bar{X}_i$, $\bar{s}_i$ are smoothed at the Budgeter cadence τ (s_b is read directly from each block's depth in the radix tree; $\bar{s}_i$ is its pool-aggregate EWMA, used by the Budgeter's pool-level model):

$$\hat{x}_i(t + \tau) \;=\; (1 - \eta_x)\, \hat{x}_i(t) \;+\; \eta_x\, \bar{x}_i^{(\tau)}, \qquad \eta_x \in [0.05,\, 0.2], \tag{16}$$

where $\bar{x}_i^{(\tau)}$ is the empirical mean over the last tick. The composite $\hat{\pi}_i$ inherits a per-tick std σ_{est} that enters the fluid limit of Appendix C.3 as the forcing on Eq. (22) and is dominated at fluid scale by the queue-noise std σ_Q of Eq. (19).

C.2 Stage-0 Calibration, Path-Counted Hit Rates, and Runtime Actuator Cost

The model of Section 3.2 treats the recovery curves $c_i(s)$, the SLO weight w_q, the per-block hit counts, sizes, and tree depths $\{n_b, B_b, s_b\}$, and the actuator cost c^{xfer} as inputs. The first two are deployment-time constants from *Stage-0 calibration*; the third is read off engine telemetry but requires a *path-counted* hit definition; the fourth is self-calibrated at runtime. When the closed-form $\hat{\pi}_i$ of Eq. (15) flattens at admission saturation, an engine-specific *admission-pressure adapter* supplies a lower-bound surrogate $\tilde{\pi}_i$.

Proof of Lemma 1 (convexity of L_i). μ_i is non-increasing convex in m_i on $[m_i^{\min}, M]$ (Che hit-curve under IRM [12, 3]); β_i is non-increasing convex in C_i (Erlang-B convexity in capacity [17]) and C_i is affine in m_i, so $\beta_i \circ C_i$ is non-increasing convex in m_i. Equation (2) combines them with non-negative coefficients $\lambda_i\, c_i(\bar{s}_i)$ and $\lambda_i\, w_q$ (frozen telemetry inputs within the tick), which preserves both properties.

Strict convexity and uniqueness of ν^*. Strict convexity of μ_i holds whenever the cached blocks have at least two distinct positive access rates h_b: differentiating Eq. (11) twice gives a Cauchy–Schwarz gap that is strict iff the h_b are non-degenerate. Strict convexity of β_i in C_i holds whenever $\rho_i > 0$ and $C_i \geq 1$, by the standard recurrence for Erlang-B's second derivative [17]. Either condition is sufficient for L_i to be strictly convex on $[m_i^{\min}, M]$, and hence for $\mathbf{m}^*$ to be unique. In the degenerate case where access rates concentrate on one block and traffic is light enough to make β_i flat, L_i is merely convex (not strict) and **(B)**'s solution set is a face of the simplex; the dual ν^* is still unique because **(B)**'s dual is a one-dimensional concave program with a single coupling constraint, and the bisection of Section 3.3 returns one canonical $\mathbf{m}^* \in \arg\min$. All members of the optimal face attain identical loss, so this degeneracy is harmless for the regret bound of Lemma 4.

Stage-0 calibration. The recovery-cost coefficients $(\kappa_2^{\text{KV}}, \kappa_1^{\text{KV}}, \kappa_0^{\text{KV}}, \kappa_1^{\text{rec}}, \kappa_0^{\text{rec}})$ of Section 3.3, the SLO weight w_q, and the adapter coefficients k_s (Eq. (17) below) are (engine, model, hardware)-tuple parameters. HiMA treats them as calibration outputs rather than paper-defined constants: a one-shot script runs at deployment time, takes < 30 s, and writes a JSON the Budgeter loads at boot. Three kernel-level probes:

1. *Recovery-cost-curve probe* for $c_i(s)$. Time the bare attention prefill kernel and the bare linear-attention recurrent kernel on a log grid of $s \in [128, 16\text{K}]$, aggregate over layer counts to obtain stack-level $c_{\text{KV}}(s), c_{\text{rec}}(s)$, and least-squares-fit each to its parametric form. Output: $(\kappa_2^{\text{KV}}, \kappa_1^{\text{KV}}, \kappa_0^{\text{KV}}, \kappa_1^{\text{rec}}, \kappa_0^{\text{rec}})$ and the crossover s^*.
2. *Prefill-throughput probe* for k_{evict} and the SLO weight. A synthetic prefill burst at fixed input length yields $1/$prefill-tps directly; this is the per-token amortized re-prefill cost, used as $k_{\text{evict}} = 1/$prefill-tps in the adapter and as the seed for w_q.
3. *Retract probe* for k_{retract}. Full re-prefill of a representative retracted request length, GPU wall measured end-to-end.

On our SGLang/Qwen3.5-A3B/H200 platform: $k_{\text{evict}} = 12.5$ µs/token (prefill-tps $\sim$80 K), $k_{\text{retract}} =$ 75 ms ($\sim$6 K-input re-prefill), $\kappa_2^{\text{KV}} = 1.32{\times}10^{-8}$ ms/tok^2, $\kappa_1^{\text{KV}} = 2.65{\times}10^{-4}$ ms/tok, $\kappa_0^{\text{KV}} = 0.78$ ms, $\kappa_1^{\text{rec}} = 4.66 \times 10^{-4}$ ms/tok, $\kappa_0^{\text{rec}} = 0.61$ ms, $s^* = 14\,400$ tokens (Fig. 3). The contribution is the calibration protocol; the specific values are platform-dependent.

Approximation caveats. Four approximations in Section 3.2 are bounded here. *(i) Erlang-B integerization.* The runtime evaluates Erlang-B at $\lfloor C_i(m_i) \rfloor$ rather than the continuous C_i used for convexity; the resulting single-step capacity error is $O(1/\bar{X}_i)$ in β_i and is absorbed into the empirical envelope of Section 4. *(ii) Greedy-knapsack overshoot.* The prefix $S^*(X)$ of Eq. (7) overshoots X by less than one block, so c_i^{evict} upper-bounds the true 0-1 knapsack optimum by at most the cost of the last block in $S_i^*(X)$. *(iii) Strict-convexity degeneracies.* L_i is strictly convex when $\lambda_i > 0$ and at least two cached blocks have distinct positive h_b; with a single positive h_b the Che formula collapses to $\mu_i = 1 - m_i^{\text{cached}}/B_1$ (linear), and L_i may be flat on a sub-interval. The dual variable ν^* remains unique by separability with one coupling constraint, so the bisection of Section 3.3 returns a canonical $\mathbf{m}^* \in \arg\min$; all members of the optimal face attain the same loss. *(iv) Jensen gap on* $c_i(\bar{s}_i)$. Eq. (2) prices miss-induced recovery as $c_i(\bar{s}_i)$ rather than $\mathbb{E}[c_i(s)]$; for non-linear c_i this is a

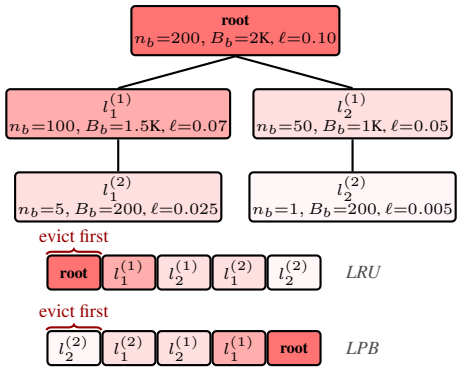


Figure 9: LPB inverts recency on a radix prefix tree (ℓ shown with $c_{i(b)}(s_b)\equiv 1$).

mean-field approximation with gap $\mathbb{E}[c_i(s)] - c_i(\bar{s}_i) = \frac{1}{2}c_i''(\bar{s}_i)\,\mathrm{Var}(s) + O(\mathrm{Var}(s)^{3/2})$. On KV ($c_i$ quadratic) the leading term is $\kappa_2^{\mathrm{KV}}\,\mathrm{Var}(s)$ and is non-negative (true loss is underestimated); on recurrent (c_i linear) the gap is zero. The runtime smooths $\bar{s}_i$ at Budgeter cadence (Eq. (16)), so the empirical envelope of Section 4 absorbs the residual on workloads with bounded $\mathrm{Var}(s)/\bar{s}_i^2$.

Path-counted hit rates. Both consumers of the per-block hit count n_b — the Che IRM rate $h_b = n_b/W$ in Eq. (11) and the LPB eviction key $\ell(b) = n_b\, c_{i(b)}(s_b)/(\mathrm{KV_bytes}(b) + \mathrm{snapshot_bytes}(b))$ of Eq. (1) — require n_b to count hits along the *full path* from each access to the root, not only at the deepest matched block:

- *LPB.* Under deepest-only counts, internal anchor blocks see $n_b = 0$ until they themselves are queried as terminals; the LPB key then degenerates to recency on the prefix tree (recently-deepened leaves accumulate n_b first regardless of how heavily their prefix is traversed), and the eviction order collapses to LRU.
- *Che.* The IRM rate h_b that enters the characteristic-time equation is the marginal demand for block b *including all extensions through it*. Deepest-only counts under-attribute traffic for popular shared anchors and bias μ_i upward, leading the Budgeter to under-provision the affected pool.

The cost is one increment per tree-depth per access (negligible against the access itself). On SGLang, LPB replaces the recency sort key in `mamba_lru_list`; on vLLM, it replaces the FIFO insertion order in the free-block queue.

Runtime actuator-cost EWMA. The actuator unit cost c^{xfer} is qualitatively different from the kernel costs above: it is dominated not by the bare `cuMemUnmap+cuMemMap` pair (a few µs per page on H200 in isolation) but by the additional cost a transfer pays *when issued under live serving traffic* — CUDA-graph deferral, allocator-lock contention with the request-admission path, and per-transfer batching of the page-table updates. None of these are visible to a deploy-time idle probe, which can underestimate live-traffic actuator cost by an order of magnitude. We therefore measure c^{xfer} at runtime: every committed transfer reports its wall-clock cost via `cudaSynchronize`-bracketed timestamps; the Budgeter maintains a process-wide EWMA

$$\hat{c}^{\mathrm{xfer}}_{t+1} = \alpha_{\mathrm{ewma}} \cdot \frac{\mathrm{transfer}_{\mathrm{us}}(t)}{n_{\mathrm{pages}}(t)} + (1-\alpha_{\mathrm{ewma}}) \cdot \hat{c}^{\mathrm{xfer}}_t,$$

with $\alpha_{\mathrm{ewma}} = 0.3$ (half-life $\sim$2 transfers). At server boot, $\hat{c}^{\mathrm{xfer}}$ is initialized to a deliberately conservative bound that suppresses transfers until at least three live-traffic observations have accumulated — a misconfigured deployment regresses cleanly to the engine baseline rather than over-triggering.

Engine-specific admission-pressure adapter. When per-pool usage saturates at the engine's admission ceiling, the closed-form $\hat{\pi}_i$ of Eq. (15) is gradient-flat in m_i (both μ_i' and β_i' drop to zero) and the bisection of Section 3.3 cannot rank pools by marginal value alone. We then substitute an

engine-specific lower-bound surrogate $\tilde{\pi}_i \triangleq B_{\text{lb}}(\Delta m_i)/\Delta m_i$, with

$$B_{\text{lb}}(\Delta m_i) \;\triangleq\; \sum_{s \in \text{Adapter}(E)} k_s \cdot S_s(i), \tag{17}$$

where E is the serving engine, Adapter(E) is the set of pressure signal types E surfaces, $S_s(i)$ is the count or duration of signal s in the current control window, and k_s is the empirically calibrated benefit-microseconds per unit of S_s. We instantiate the adapter on SGLang as

$$\text{Adapter(SGLang)} \;=\; \{\, S_{\text{evict}},\; S_{\text{retract}},\; S_{\text{pause}},\; S_{\text{queue}},\; S_{\text{persist}} \,\},$$

where S_{evict} is the per-tick token count released by SGLang's tree-cache eviction (the engine's primary admission-pressure relief), with $k_{\text{evict}} = 1/\text{prefill-tps}$; $S_{\text{retract}}, S_{\text{pause}}, S_{\text{queue}}$ are scheduler counters as in standard preemption-based engines (rare on SGLang because eviction satisfies pressure first); $S_{\text{persist}} = n_{\text{above}}(i)$ is the consecutive-tick count the pool has spent above its admission ceiling, weighted by $c_{\text{persist-tick}}$ as a saturation prior. The eviction-as-primary-signal framing reflects SGLang's `check_decode_mem` running `evict_from_tree_cache` *before* retraction. For other engines the adapter dominates a different signal: a vLLM adapter would weight CPU swap-out tokens as S_{swap} and PCIe IO cost as k_{swap}. The adapter is engaged only on the saturated face of the dual; on the unsaturated face $\hat{\pi}_i$ alone is sufficient and B_{lb} is unused.

C.3 Closed-Loop Fluid Limit and Hysteresis Regret

This subsection backs Proposition 1 and Theorem 1: the per-byte price identities (Eq. (19)) that match **(A)**'s admit and queue costs to **(B)**'s KKT prices, and the stationary suboptimality bound (Eq. (4)). Throughout, $\hat{\boldsymbol{\pi}} = (\hat{\pi}_i)_{i \in \mathcal{I}}$ is the EWMA marginal-price vector of Eq. (15) and σ_{est} its per-tick std.

Per-arrival convention. Proposition 1 and Theorem 1 interpret π_i and ν^* as *per-arrival* per-byte prices (units ms/byte), matching the per-byte units of $Q\, w_q/X$ and the cheapest per-byte admit cost

$$g_i(X;\mathbf{m}) \;\triangleq\; \frac{1}{X} \min_{a \,\in\, \mathcal{A}_{\text{admit}}(X;\mathbf{m})} c_a \quad \text{(ms/byte)}, \tag{18}$$

where $\mathcal{A}_{\text{admit}}(X;\mathbf{m}) \subseteq \mathcal{A} \setminus \{\text{defer}\}$ is the set of admit candidates feasible at state $\mathbf{m}$. The per-second marginal $-L_i'$ (units ms/(s·byte)) and its per-arrival counterpart differ by a factor of λ_i that scales both sides of **(B)**'s KKT condition $\pi_i = \nu^*$ identically.

Proposition 1 (Shadow-price tracking)**.** *Assume (A1)–(A3) of Theorem 1. All expectations are under the stationary distribution and prices are evaluated at* $\mathbf{m}^*$*. Decompose* $\pi_i = \pi_i^{rec} + \pi_i^{def}$ *into recovery and defer components,* $\pi_i^{rec} \triangleq -c_i(\bar{s}_i)\, \mu_i'(m_i)$ *and* $\pi_i^{def} \triangleq -w_q\, \beta_i'(m_i)$ *(per-arrival convention;* $\pi_i^{rec}, \pi_i^{def} \geq 0$*). The per-byte prices* ***(A)*** *compares match* ***(B)****'s KKT prices to first order:*

$$\mathbb{E}[g_i] = \pi_i^{rec} + O(\bar{c}^{xfer}) + O(\varepsilon), \quad \mathbb{E}[Q\, w_q/X] = \nu^* + O\big(w_q\, \sigma_Q/\mathbb{E}[X]\big), \tag{19}$$

where σ_Q *is the queue-depth std around its fluid mean. The two sides differ in expectation by exactly* $\pi_i^{def}(m_i^*)$ *(KKT couples* $\pi_i^{rec} + \pi_i^{def} = \nu^*$ *at* $\mathbf{m}^*$*); this gap is absorbed by the queue equilibrium* q^* *of Lemma 3, so* ***(A)****'s test threshold tracks* ν^* *even though* g_i *alone internalizes only the recovery component.*

Asymptotic regime and constants. Big-O constants depend on $(\lambda_i, M, w_q, \mathbb{E}[X], L_i''(m_i^*))$ held fixed; bounds are asymptotic in $\bar{c}^{\text{xfer}}, \sigma_Q, \varepsilon \to 0$, where ε is the per-arrival demand bound $X(r) \leq \varepsilon M$ from Proposition 1. In our deployment a 16K-token KV admission against $M \sim 80\,\text{GB}$ lands at $\varepsilon \sim 10^{-2}$, so the $O(\varepsilon)$ term enters the regret at the 10^{-4} level after the squaring in Theorem 1 — well inside the empirical envelope of Section 4. The queue Q is the deferred-request backlog: **(A)** routes blocked arrivals (per-arrival defer probability β_i from Section 3.2) into this external queue, where they wait at SLO penalty w_q per tick until admission. The M/M/C/C form of β_i models only the admission-blocking event; the dynamics of Q itself appear in Eq. (23) below.

Proof of Proposition 1 and Theorem 1. The three identities split as follows: Identities 1 and 2 give Proposition 1; Identity 3 gives Theorem 1.

Identity 1: $\mathbb{E}[g_i(X;\mathbf{m})] = \pi_i^{rec}(m_i) + O(\mathbb{E}[c^{xfer}/X]) + O(\mathbb{E}[X]/m_i^{cached})$. By definition (Eq. (18)), $g_i = \min_{c\in\mathcal{C}} c/X$, where the four candidates of $\mathcal{C}$ are own-free (0), cross-free (c^{xfer}/X), own-evict ($c_{i_{\text{dst}}}^{\text{evict}}/X$), and cross-evict ($(c_{i_{\text{src}}}^{\text{evict}} + c^{\text{xfer}})/X$). On the own-pool branches the greedy 0-1 knapsack of Eq. (7) produces a per-byte cost $c_i^{\text{evict}}(X)/X = \sum_{b\in S_i^*(X)} n_b\, c_i(s_b)/X$. Under stationarity and the Che characteristic-time identity (Eqs. (11) and (13)), the expected per-byte loss of the cheapest cached prefix equals the recovery-component marginal $\pi_i^{\text{rec}}(m_i) = -c_i(\bar{s}_i)\,\mu_i'(m_i)$ up to the $O(1)$-block overshoot ($S^*(X)$ overshoots X by less than the largest B_b, contributing the $O(\mathbb{E}[X]/m_i^{\text{cached}})$ envelope error of Section 3.3). The defer component π_i^{def} does not appear in g_i directly — **(A)**'s admit branches do not measure the queue-relief that more m_i provides — and enters only via Identity 2 below through the queue side $Q\,w_q/X$. Whenever the chosen branch is cross-pool, the transfer overhead $c^{\text{xfer}}(X)/X$ is added; in expectation this contributes $O(\mathbb{E}[c^{\text{xfer}}/X])$. Combining both error sources gives Identity 1.

Identity 2: $\mathbb{E}[Q\,w_q/X] = \nu^* + O(w_q\,\sigma_Q/\mathbb{E}[X])$. This is Lemma 3 below: the closed-loop fluid system Eqs. (21) to (23) has a unique globally asymptotically stable fixed point with $q^* w_q = \bar{X}\cdot\nu^*$, and the stochastic system tracks it with mean-square deviation $O(\sigma_Q^2)$.

Identity 3: regret $= O((c^{xfer})^2) + O(\sigma_Q^2)$. Under two-timescale separation, each **(A)** firing acts as a constant-step-size stochastic subgradient step on **(B)**'s dual; with fixed per-arrival step size, the standard result [2, Ch. II] gives mean-square tracking error of order step-size rather than zero-error convergence (Borkar's [4] decreasing-step analysis is not directly applicable), and this $O(\text{step})$ term is dominated by the $O(\sigma_Q^2)$ queue-noise term that follows. The dead-zone induced by Eq. (24) contributes the $O((c^{\text{xfer}})^2)$ term via the Taylor expansion of Lemma 4 ($\Delta_{\text{hyst}} \propto c^{\text{xfer}}$), and the $O(\sigma_Q^2)$ term comes from the same fluid perturbation that gave Identity 2. Drift bounds under non-stationary $\dot{\pi}_i$ are in Lemma 5.

Closed-loop fluid limit

We recall the per-byte threshold form of **(A)**: admit iff

$$g_i(X;\mathbf{m}) \;\le\; Q\,w_q/X, \tag{20}$$

and defer otherwise.

Joint state and dynamics. Let $\mathbf{z}(t) \triangleq (\mathbf{m}(t), \hat{\boldsymbol{\pi}}(t), Q(t))$ be the system state at wall-clock t, with $\hat{\boldsymbol{\pi}}$ the EWMA marginal-price estimate (Eq. (15)) and Q the deferred- request queue depth. Three event classes update $\mathbf{z}$:

- *Per arrival.* **(A)** compares $g_i(X;\mathbf{m})$ against $Q\,w_q/X$ (Eq. (20)); admit consumes X bytes from $m_i^{\text{cached}} + m_i^{\text{free}}$ and converts them to live; defer increments Q.
- *Per service completion.* A request leaves; Q decrements (if non-empty) and live bytes return to free.
- *Per Budgeter tick (τ seconds).* The Lagrange bisection of Section 3.3 on $\hat{\boldsymbol{\pi}}$ produces a new $\hat{\nu}$ and resizes $\mathbf{m}$ toward $\mathbf{m}^*(\hat{\boldsymbol{\pi}})$ via cross-pool transfer.

Fluid model. Under i.i.d. arrivals and per-arrival step $X(r) \le \varepsilon M$ for small ε, the standard fluid scaling $\mathbf{z}^{(N)}(t) = \mathbf{z}(Nt)/N$ with $N\to\infty$ converges to the deterministic ODE system

$$\dot{\mathbf{m}} = \kappa_B \cdot (\mathbf{m}^*(\hat{\boldsymbol{\pi}}) - \mathbf{m}), \tag{21}$$

$$\dot{\hat{\boldsymbol{\pi}}} = \eta_v \cdot (\boldsymbol{\pi}(\mathbf{m}) - \hat{\boldsymbol{\pi}}), \tag{22}$$

$$\dot{Q} = \rho_{\text{tot}} \cdot \mathbb{1}[g_i(\bar{X};\mathbf{m}) > Q\,w_q/\bar{X}] \;-\; \mu_{\text{serve}}(Q,\mathbf{m}), \tag{23}$$

with Budgeter rate $\kappa_B = 1/\tau$, total offered load $\rho_{\text{tot}} = \sum_i \rho_i$, and deterministic service rate $\mu_{\text{serve}}(Q,\mathbf{m})$ (admitted requests freeing live bytes plus queued requests served on next admission).

Lemma 3 (Closed-loop shadow-price tracking)**.** *Under the assumptions of Proposition 1 and strict convexity of L_i (Section 3.2), the fluid system Eqs.* (21) *to* (23) *has a unique fixed point* $\mathbf{z}^* = (\mathbf{m}^*, \boldsymbol{\pi}^*, q^*)$ *satisfying*

$$q^*\,w_q \;=\; \bar{X}\cdot\nu^*, \qquad \mathbf{m}^*, \nu^* \text{ as in (B)'s KKT.}$$

The fixed point is globally asymptotically stable, and the original stochastic system tracks the fluid trajectory with mean-square deviation $O(\sigma_Q^2)$ on stationary intervals, so $\mathbb{E}[Q\, w_q/X] = \nu^* + O(w_q\, \sigma_Q/\mathbb{E}[X])$ — which is Eq. (19).

Proof sketch. Existence and form of the fixed point. Setting Eqs. (21) to (23) to zero forces (i) $\mathbf{m} = \mathbf{m}^*(\hat{\boldsymbol{\pi}})$, (ii) $\hat{\boldsymbol{\pi}} = \boldsymbol{\pi}(\mathbf{m})$, (iii) the admit/defer indicator must equal $\mu_{\text{serve}}/\rho_{\text{tot}}$ in steady state. (i)+(ii) is the Budgeter equilibrium; substituting and using **(B)**'s KKT condition gives $\hat{\boldsymbol{\pi}}^* = \boldsymbol{\pi}^*$ with $\pi_i^* = \nu^*$ on interior pools. (iii) requires the threshold $g_i = q\, w_q/\bar{X}$ to balance arrival and service rates; under strict convexity the threshold is monotone in q, giving a unique q^*. By Eq. (20) the threshold equals $g_i(\bar{X}; \mathbf{m}^*) = \pi_i(m_i^*) = \nu^*$ at equilibrium, hence $q^* w_q = \bar{X} \cdot \nu^*$.

Stability. Take the Lyapunov candidate $V(\mathbf{z}) = \frac{1}{2}\|\mathbf{m} - \mathbf{m}^*\|^2 + \frac{1}{2\eta_v}\|\hat{\boldsymbol{\pi}} - \boldsymbol{\pi}^*\|^2 + \frac{1}{2}(Q - q^*)^2$. The indicator $\mathbb{1}[g_i > Q\, w_q/\bar{X}]$ in Eq. (23) is discontinuous on the threshold surface $g_i = Q\, w_q/\bar{X}$; we interpret Eqs. (21) to (23) as a Filippov differential inclusion, taking the convex hull $\rho_{\text{tot}} \cdot [0, 1]$ on the surface, and $\dot{V}$ as the upper Dini derivative. Differentiating along Eqs. (21) to (23) and using the strict-convexity bound $(\boldsymbol{\pi}(\mathbf{m}) - \boldsymbol{\pi}^*)^\top(\mathbf{m} - \mathbf{m}^*) \le -\sigma_{\min}\|\mathbf{m} - \mathbf{m}^*\|^2$ (with $\sigma_{\min} > 0$ the minimum eigenvalue of $\nabla^2 L$ at $\mathbf{m}^*$, which exists under the strict-convexity remark of Section 3.2) gives $\dot{V} \le -c\, V$ for some $c > 0$ on Filippov solutions; global asymptotic stability follows by [6].

Stochastic deviation. The fluid-limit perturbation argument of Dai [7] gives $\sup_{t \le T} \mathbb{E}\|\mathbf{z}^{(N)}(t) - \mathbf{z}^*(t)\|^2 = O(\sigma_Q^2/N)$ on stationary intervals, where σ_Q is the queue-depth std around its fluid mean (the EWMA estimator noise σ_{est} feeds in as the forcing term on Eq. (22), but is dominated at fluid scale by σ_Q). Reading $Q/N \to q^*$ in mean square gives $\mathbb{E}[Q\, w_q/X] = \nu^* + O(w_q\, \sigma_Q/\mathbb{E}[X])$. □

Stationary hysteresis regret

Dead-zone augmentation. To suppress noise-induced action storms when $\hat{\pi}_i$ is noisy on a $\tau = 30$ s tick, **(A)**'s admission test is augmented with a multiplicative safety margin:

$$\text{execute admit candidate } c \text{ iff} \quad Q\, w_q - c \;\ge\; \alpha \cdot c^{\text{xfer}}(X), \tag{24}$$

where c ranges over the four admit candidates of Eq. (3); the gate selects the largest-margin candidate that clears the threshold and defers otherwise. $c^{\text{xfer}}(X)$ on the right serves as a *system-wide noise scale*, not the actuator cost charged to this specific candidate: empirically, the per-tick std of $\hat{\pi}_i \cdot X$ matches $c^{\text{xfer}}(X)$ in magnitude, so a single factor calibrates the dead-zone for both own-pool and cross-pool candidates. Default $\alpha = 1.5$ ignores fluctuations within one-and-a-half noise magnitudes. Equation (24) induces an allocation dead-zone of width $\Delta_{\text{hyst}} = \alpha\, c^{\text{xfer}}(\bar{X})/\bar{\pi}$ around m_i^*, with $\bar{\pi}$ the typical marginal-price scale, and the $O((c^{\text{xfer}})^2)$ term in Eq. (4) is the second-order loss this dead-zone admits.

Lemma 4 (Stationary hysteresis regret)**.** *Fix a pool i with convex, twice-differentiable L_i and let m_i^* be the Lagrange-equalizing allocation $\pi_i(m_i^*) = \nu^*$ (where $\pi_i \triangleq -L_i'$). Under stationary traffic and the EWMA estimator of Appendix C.1, the steady-state loss gap induced by a dead-zone of half-width $\Delta_{hyst}/2$ around m_i^* satisfies*

$$\mathcal{R}_i \;=\; \frac{\Delta_{hyst}^2}{24}\, L_i''(m_i^*) \;+\; O(\Delta_{hyst}^3), \tag{25}$$

and the system-wide regret summed over interior pools recovers the $O((c^{xfer})^2)$ leading term of Eq. (4) *via $\Delta_{hyst} \propto c^{xfer}$.*

Proof sketch. The Budgeter implements a 1-D bang-bang policy around m_i^* with dead-zone $[m_i^* - \Delta_{\text{hyst}}/2,\, m_i^* + \Delta_{\text{hyst}}/2]$: inside the dead-zone no transfer is issued, outside it the actuator drives m_i back to m_i^*. Under stationary traffic the EWMA $\hat{\pi}_i$ converges in expectation to the true marginal. We approximate the Budgeter's stationary distribution as uniform on the dead-zone interval; this is the exact steady-state of a reflected random walk under symmetric mean-zero i.i.d. shocks (15, Sec. 3; here on the diffusion limit of the EWMA increments) and it bounds the true distribution to leading order whenever the per-tick shock std σ_{est} is small relative to Δ_{hyst}. Asymmetry of the empirical shock distribution shifts higher moments only and contributes $O(\Delta_{\text{hyst}}^3)$, already absorbed in Eq. (25). Taylor-expanding L_i at m_i^* and using $L_i'(m_i^*) = -\nu^*$,

$$L_i(m) - L_i(m_i^*) \;=\; -\nu^*(m - m_i^*) + \tfrac{1}{2} L_i''(m_i^*)\,(m - m_i^*)^2 + O((m - m_i^*)^3).$$

The linear term vanishes upon summation over interior pools (the dead-zone perturbations satisfy $\sum_i (m_i - m_i^*) = 0$ by budget conservation). Averaging the quadratic term over the uniform stationary distribution of half-width $\Delta_{\text{hyst}}/2$,

$$\frac{1}{\Delta_{\text{hyst}}} \int_{-\Delta_{\text{hyst}}/2}^{\Delta_{\text{hyst}}/2} \tfrac{1}{2} L_i''(m_i^*)\, x^2\, dx \;=\; \frac{\Delta_{\text{hyst}}^2}{24}\, L_i''(m_i^*),$$

which is Eq. (25). Substituting $\Delta_{\text{hyst}} = \alpha\, c^{\text{xfer}}/\bar{\pi}$ from Eq. (24) converts this to $\mathcal{R}_i = O((c^{\text{xfer}})^2)$, recovering Eq. (4). □

Lemma 5 (Non-stationary degradation)**.** *Suppose the marginal-price curve drifts at bounded rate $|\dot{\pi}_i(t)| \le \rho$, and the Budgeter runs at interval τ (Appendix B.2.3, default $30\,s$). The optimum $m_i^*(t)$ moves at rate $|\dot{m}_i^*| \le \rho / L_i''(m_i^*)$, and the per-pool regret of Lemma 4 is multiplied by a degradation factor*

$$1 \;+\; \frac{2\,\rho\,\tau}{L_i''(m_i^*)\,\Delta_{hyst}}. \tag{26}$$

Proof sketch. Differentiating $\pi_i(m_i^*(t)) = \nu^*$ in t gives $\dot{m}_i^* \cdot \pi_i'(m_i^*) = -\dot{\pi}_i$, and since $\pi_i' = -L_i''$, $\dot{m}_i^* = \dot{\pi}_i / L_i''$, hence $|\dot{m}_i^*| \le \rho / L_i''(m_i^*)$. Between two consecutive transfers the optimum drifts by at most $|\dot{m}_i^*|\,\tau$, while hysteresis only triggers once the gap exceeds $\Delta_{\text{hyst}}/2$. The stationary distribution is therefore biased by a tracking lag of magnitude up to $|\dot{m}_i^*|\,\tau$; the squared-error integrand of Lemma 4 acquires an additive cross- term $2\,(m - m_i^*) \cdot |\dot{m}_i^*|\,\tau$, which on the dead-zone of half-width $\Delta_{\text{hyst}}/2$ produces the multiplicative factor in Eq. (26). □

Tradeoff. Eq. (25) is monotone increasing in Δ_{hyst}, so smaller dead-zones reduce stationary regret; the cost is an increased transfer rate, since each transfer below the gate threshold of Eq. (24) pays c^{xfer}. The expected per-interval overhead therefore grows like $\Pr[|\hat{\pi}_i(t) - \nu^*| > \Delta_{\text{hyst}}\, L_i''(m_i^*)/2] \cdot c^{\text{xfer}}$. The default $\alpha = 1.5$ is calibrated so that, at the system's typical drift rate ρ, the stationary regret of Eq. (25) matches the per-interval actuator overhead — the balance between the $O((c^{\text{xfer}})^2)$ and the transfer-frequency terms in Eq. (4).

D Extended Evaluation Material

This appendix collects evaluation content cross-referenced from Section 4 but not embedded inline: the experimental setup (hardware, models, baselines, workloads, metrics), the cross-model headline on Qwen3-Next-80B-A3B and Kimi-Linear-48B-A3B, the static-best partition sweep, the cross-engine comparison against vLLM, the deeper isolations of the intra-pool layer (non-perturbed traffic and the ablation decomposition on the cold-burst recovery), the estimator-coverage analysis under admission saturation, the mutually-enabling composite trace, the parameter ablations, and the full quality-preservation battery.

D.1 Experimental setup

Hardware and software. We run on NVIDIA H200. We use SGLang v0.5.10.post2 with our fork on top, and vLLM v0.20.0 as the cross-engine baseline. Every cell uses fixed random seeds and is repeated for n=5 trials (traffic perturbation uses n=3); reported numbers are mean ± std across trials. Engine knobs default to `mem_fraction_static`=0.8, and HiMA's defaults are hysteresis dead-zone $\Delta_{\text{hyst}} = 5\%$, Budgeter window $\tau = 30\,\text{s}$, and net-benefit margin $\alpha = 1.5$, unless an ablation varies them. HiMA also runs a one-shot calibration once per (model, hardware) deployment to fit its cost-curve parameters.

Models and baselines. We evaluate on three public hybrid A3B models: Qwen3.5-35B-A3B (DeltaNet hybrid), Qwen3-Next-80B-A3B-Instruct (DeltaNet+MoE hybrid), and Kimi-Linear-48B-A3B-Instruct. We compare HiMA against two SGLang baselines — default SGLang [40] and the per-workload static-best partition (best `mamba_full_memory_ratio` r from a 4-ratio sweep, where $r \in [0, 1]$ is SGLang's static HBM split between the recurrent and paged-KV pools, r=1 giving recurrent the entire pool budget) — and vLLM v0.20.0 [21] as a cross-engine reference. Per-model TP follows each engine's native fit: Qwen3.5 at TP=1, Qwen3-Next at TP=2, and Kimi at TP=2.

Table 4: Cross-model main result on Qwen3-Next-80B-A3B and Kimi-Linear-48B-A3B / H200, mirroring Table 1.

workload	metric	SGLang default	**HiMA**
Qwen3-Next-80B-A3B			
long-horizon	out-TPS	$1{,}239 \pm 1$	$\mathbf{1{,}340 \pm 30}$
	P99 TTFT (ms)	$3{,}309 \pm 140$	$\mathbf{2{,}380 \pm 200}$
swarm	out-TPS	$4{,}474 \pm 51$	$\mathbf{6{,}260 \pm 150}$
	P99 TTFT (ms)	$58{,}360 \pm 943$	$\mathbf{26{,}260 \pm 1{,}500}$
traffic perturbation	out-TPS (avg A,B,C)	612 ± 38	$\mathbf{1{,}184 \pm 64}$
	out-TPS (Phase-C)	207 ± 14	$\mathbf{463 \pm 26}$
	P99 TTFT (avg, ms)	$7{,}918 \pm 412$	$\mathbf{2{,}876 \pm 213}$
	P99 TTFT (Phase-C, ms)	$9{,}152 \pm 437$	$\mathbf{2{,}387 \pm 178}$
Kimi-Linear-48B-A3B			
long-horizon	out-TPS	831 ± 12	$\mathbf{893 \pm 30}$
	P99 TTFT (ms)	$37{,}483 \pm 405$	$\mathbf{28{,}100 \pm 500}$
swarm	out-TPS	$2{,}023 \pm 32$	$\mathbf{2{,}870 \pm 80}$
	P99 TTFT (ms)	$83{,}682 \pm 479$	$\mathbf{37{,}700 \pm 800}$
traffic perturbation	out-TPS (avg A,B,C)	538 ± 32	$\mathbf{1{,}087 \pm 58}$
	out-TPS (Phase-C)	177 ± 11	$\mathbf{425 \pm 23}$
	P99 TTFT (avg, ms)	$9{,}143 \pm 458$	$\mathbf{3{,}174 \pm 218}$
	P99 TTFT (Phase-C, ms)	$10{,}623 \pm 487$	$\mathbf{2{,}693 \pm 192}$

Workloads. We evaluate on four hybrid-serving workloads. The three synthetic workloads instantiate the two regimes contrasted in Fig. 1 (long-horizon = KV-bound, swarm = recurrent-bound) plus a traffic perturbation trace that interleaves both within one trial. The fourth is real-world, testing the engine on traffic from production agent harnesses. Synthetic prompts use random-token sampling at deterministic length $D(\text{in}, \text{out})$, except the traffic perturbation trace, which also draws shared-prefix prompts from SGLang's generated-shared-prefix benchmark (a long system prompt followed by a short query). Each cell runs for a 10-minute wall-clock window.

- *Long-horizon agent* (Section 4.1, KV-bound): $N{=}14$ concurrent closed-loop sessions, each turn at $D(4{,}096, 4{,}096)$, capped at 150 K total tokens per session.
- *Agent swarm* (Section 4.2, recurrent-bound): $N{=}800$ concurrent closed-loop sessions ($\sim 2\times$ recurrent-state slot capacity, so admission queues), 2–5 turns of $D(256, 256)$, 5 K-token cap.
- *Traffic perturbation trace* (Section 4.2, mixed): three open-loop Poisson phases concatenated and repeated across three A→B→C cycles. Phase A is GSP shared-prefix ($100{\times}16$ prompts, recurrent-bound to $\geq$ 0.95 utilization, RPS=12); Phase B is $D(12{,}000, 1{,}000)$ random tokens (KV-bound under deep admission queue, RPS=8); Phase C returns to the Phase A pattern as a post-perturbation recovery test.
- *Claude Code agent traffic* (Section 4.1): two instantiations of the same agent. *(i) Live rollout*: Claude Code on the SWE-Bench-Pro task suite, run through the Harbor sandbox; serving time is only a fraction of the wall clock. *(ii) Trace replay*: $N{=}14$ concurrent closed-loop sessions, each replaying one real Claude Code conversation verbatim (median 92 turns, $\geq$100 K tokens) from 106 deduplicated public sessions.

Metrics. TTFT is the first-token latency, E2E is the full-response wall time, TPOT is the per-output-token decode time, and input/output TPS are aggregate input/output tokens per wall-second. Mean and P99 are taken over the per-request distributions within a trial.

D.2 Cross-model results: Qwen3-Next-80B-A3B and Kimi-Linear-48B-A3B

Table 4 extends Section 4.1's Qwen3.5-35B-A3B headline (Table 1) to two larger hybrid models. The relative win on the swarm workload tracks the Qwen3.5 result ($\sim$ 40–50% TPS, $\sim$ 50–55% P99 across both 35B and 80B), and the long-horizon / traffic perturbation gains remain consistent in P99 reduction.

Table 5: Ablation on the cold-burst-cum-binding-shift composite trace. The mutually-enabling prediction is that the joint cell strictly dominates both single-layer cells in the cold-burst-during-shift overlap. The joint cell beats the better of intra-only / inter-only by an extra $\sim 32\%$ on P99 ($1098 \rightarrow 745$ ms), confirming super-additivity (cf. the I decomposition in Table 6).

cell	input TPS	mean TTFT (ms)	P99 TTFT (ms)	cross-pool transfers
default	$24\,150 \pm 8$	385 ± 6	1487 ± 52	0
intra only	$24\,148 \pm 12$	338 ± 5	1098 ± 35	0
inter only	$24\,151 \pm 9$	354 ± 7	1186 ± 38	4.2
intra+inter	$24\,153 \pm 11$	$\mathbf{326 \pm 6}$	$\mathbf{745 \pm 28}$	4.2

Table 6: Contribution decomposition $\Delta_{\text{intra}}, \Delta_{\text{inter}}, \Delta_J, I$ across the four main workloads. Negative I (super-additive) is the mutually-enabling signature. long-horizon / swarm / traffic perturbation each show one layer dominating ($I \approx 0$ or small positive); only the cold-burst-cum-shift composite produces a clearly negative I, the regime intro Section 1 predicts.

workload	Δ_{intra}	Δ_{inter}	Δ_J	interaction I
long-horizon (P99 TTFT)	$+1.7\%$	-23.8%	-25.1%	-3.0%
swarm (P99 TTFT)	-2.0%	-52.7%	-55.0%	-0.3%
traffic perturbation (Phase-C P99)	-30.3%	-0.5%	-30.0%	$+0.8\%$
cold-burst-cum-shift (P99 TTFT)	-26.2%	-20.2%	$\mathbf{-49.9\%}$	$\mathbf{-3.5\%}$

D.3 Mutually-enabling: cold-burst-cum-binding-shift composite

The introduction's mutually-enabling claim has two halves:

- *Without the intra-pool layer, the inter-pool layer's per-byte value signal off the prefix cache is destabilized by cold-burst perturbations.* Operationally: inter-only should underperform full on a workload where cold bursts coincide with binding-pool flips, because the gate reads a recency-LRU-corrupted prefix signal and either triggers the wrong direction or fails to trigger when it should.
- *Without the inter-pool layer, the intra-pool layer's signal lands in a statically-partitioned engine that cannot grow the prefix pool when the signal calls for it.* Operationally: intra-only should underperform full on a workload where LPB saves snapshot nodes that the static-partition engine then has no room to keep cached — the saved snapshots compete with KV bytes inside a fixed pool.

Composite trace. A composite trace interleaves the cold-burst pattern with traffic perturbation-style boundaries: cold bursts are intentionally aligned with phase transitions where the binding pool changes, so the inter-pool gate must act on a prefix-cache marginal-value signal that is being perturbed *simultaneously*. We compare the ablation arms; the predictions are: intra-only beats default on the cold-burst phases (intra benefit); inter-only beats default on the traffic perturbation segments (inter benefit); and the full system beats both single-layer arms in the overlap (mutual enabling).

Contribution decomposition. For any ablation, $\Delta_{\text{intra}} = M_{\text{intra}} - M_{\text{default}}$, $\Delta_{\text{inter}} = M_{\text{inter}} - M_{\text{default}}$, $\Delta_J = M_{\text{full}} - M_{\text{default}}$, and the interaction term is $I = \Delta_J - \Delta_{\text{intra}} - \Delta_{\text{inter}}$. *Negative I* (super-additive) on the composite trace would be the strongest evidence for the mutually-enabling claim; *zero I* (additive) means the layers are independent; *positive I* (sub-additive) would mean the layers partly cover the same axis.

Orthogonality control. As a control we re-state that on traces where the two layers' inputs are *not* simultaneously perturbed, the layers act on independent control axes. On the cold-burst trace the inter-pool gate correctly identifies "no transfer net-positive" and the full system tracks intra-only; on the long-horizon trace the lack of shared-prefix structure makes LPB a no-op and the full system tracks inter-only. The composite trace is constructed precisely to break this orthogonality.

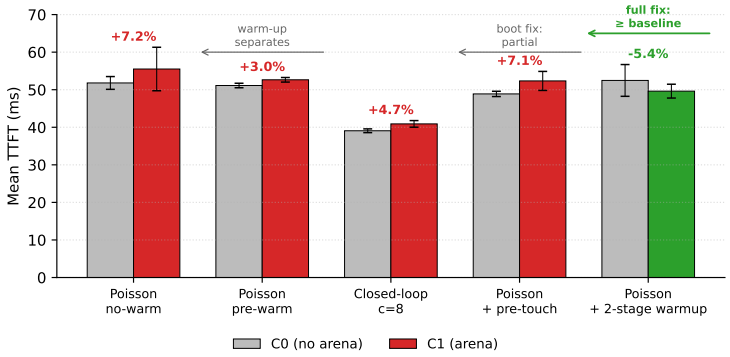


Figure 10: Arena structural cost decomposition across five measurement conditions on the same workload (Qwen3.5-35B-A3B / H200, random 512in/128out, n=500). Each pair shows C0 (no arena) and C1 (arena) mean TTFT $\pm$ std; $\Delta\%$ annotated. The narrative arc reads left-to-right: (i) naive Poisson reports $+7.15\%$ with σ_{C1}=5.79 ms; (ii) bench-side pre-warm reveals warm-state cost is only $+2.98\%$ (TLB warm-up was 4 of the 7%); (iii) closed-loop arrivals further isolate Poisson amplification ($+4.69\%$ floor); (iv) boot-time `tensor.zero_()` only partly fixes (σ cut $2.3\times$ but mean unchanged — fill kernel SM grid mismatches attention's); (v) two-stage warm-up erases the cost: arena-on is -5.4% mean / -59% P99 vs. baseline.

D.4 Ablations and robustness

Are HiMA's hyperparameter defaults robust? We sweep each around its default and report the trend below. The defaults are hysteresis Δ_{hyst}=5%, Budgeter window τ=30 s, VMM page 2 MiB, and net-benefit margin α=1.5. Appendix D.4.1 reports the arena's structural cost; Appendices D.4.2 and D.4.3 are supporting microbenchmarks. Appendix D.5 covers end-to-end quality preservation.

Hyperparameter sweeps. We sweep each hyperparameter around its default; raw per-cell tables are in the anonymized repo. **Hysteresis** $\Delta_{\text{hyst}} \in \{0, 1, 5, 10, 20\}\%$: values $\leq 10\%$ are interchangeable on monotone demand; Δ=20% over-suppresses transfers. **Budgeter window** $\tau \in \{2, 8, 30, 60, 120\}$ s: τ=2 drives an actuator storm whose syscall queueing makes P99 worse than default; τ=120 misses binding shifts. **VMM chunk size** $\in \{2, 64, 256, 1024\}$ MiB: 2 MiB matches H200's native VMM page; coarser chunks pay larger per-transfer wall-time. **Net-benefit margin** $\alpha \in \{1.0, 1.5, 2.5\}$: α=1.0 adds transfers without P99 benefit; α=2.5 rejects every binding-shift candidate and degrades P99 by $+33\%$. The defaults sit at the smooth, monotone operating point of each sweep.

D.4.1 Arena's structural cost

The arena introduces a CUDA-VMM-backed allocation path under each pool's tensor pointer. Even when the actuator never triggers, this changes the allocation: pools live behind `cuMemMap`-mapped chunks rather than a single `cudaMalloc` slab. We characterize the warm-state cost of this allocation across five measurement conditions (Fig. 10): naive Poisson; bench-side pre-warm; closed-loop arrivals; boot-time `tensor.zero_()`; and full two-stage warm-up (`SGLANG_ARENA_WARMUP=1`). The warm-state structural cost is $\approx 3\%$ mean TTFT, fully erased by the two-stage warm-up.

D.4.2 Page-grain VMM kernel isolation microbenchmark

Appendix D.4.1 reports a $+7.15\%$ mean TTFT cost for the page-grain VMM allocation path, and traces the dominant component to GPU TLB warm-up rather than to any kernel-datapath penalty from reading `cuMemMap`-backed tensors. The supporting evidence is a kernel-level isolation micro-benchmark that fixes the kernel inputs and toggles only the underlying allocation path.

Setup. On Qwen3.5-35B-A3B's hot-path shapes (`hidden_size`=2048, `moe_intermediate_size`=512, 256 experts top-8, 16/2 Q/KV heads, head-dim 256, page-size 64, 4096-token KV), we time four kernels — `fused_moe` (the SGLang Triton MoE kernel), FlashAttention page-decode, RMSNorm, and a representative GEMM at the model's 2048 $\times$ 2048

Table 7: Kernel-level isolation: 2 MiB-page VMM (`cuMemMap`) vs. `cudaMalloc` allocation paths, identical kernel inputs, 50 warmup + 200 timed iterations on Qwen3.5-35B-A3B / H200. Ratio is mean(VMM) / mean(`cudaMalloc`); a ratio of 1.00 would mean the two paths are equivalent. All four kernels run within $\pm 1\%$.

kernel	mean cudaMalloc (μs)	mean VMM (μs)	ratio (VMM / cudaMalloc)
`fused_moe`	228.6	226.9	0.9925
FlashAttention (decode)	29.0	28.9	0.9934
RMSNorm	13.5	11.9	0.8859
GEMM (2048×2048)	46.8	46.6	0.9965

projection shape — with their input tensors allocated either via `torch.empty` (the engine's `cudaMalloc` caching allocator) or via a 2 MiB-page `cuMemCreate+cuMemMap` construction into a reserved VMM range, with no other code path differences. Each kernel runs 50 warmup iterations followed by 200 timed iterations using `cuda.Event` measurements. We report the ratio of mean wall-time on the VMM path to mean wall-time on the `cudaMalloc` path.

Result. Table 7 summarizes. All four ratios fall in $[0.886, 0.997]$: the maximum kernel-level slowdown attributable to the VMM-backed allocation path is at most $\approx 1\%$ of mean kernel time, well within run-to-run noise. `fused_moe` ratio is 0.992, indistinguishable from FlashAttention's 0.993 and GEMM's 0.997; RMSNorm runs slightly faster on the VMM path (ratio 0.886, dominated by single-microsecond timing variance on a $\sim 12\,\mu$s kernel). There is no MoE-specific kernel-datapath penalty: the structural cost of page-grain VMM is therefore not on the kernel datapath at the working-set sizes resident in TLB.

Caveat: TLB regime not exercised. This micro-benchmark uses a 4,096-token KV (≈ 30 MB) which fits trivially in the H200 GPU's per-SM TLB at 2 MiB pages. The ≈ 25 GiB KV pool exercised in serving (Appendix D.4.1) generates ≈ 12K page-table entries against an SM TLB that covers only ≈ 1–2 GiB, so cold-page touches under saturation are the regime that produces the $+7\%$ mean-TTFT delta in the body. The kernel-isolation result here therefore only excludes a kernel-intrinsic penalty (the working set is fully TLB-resident throughout); it does not contradict the TLB-warm-up explanation given in the body.

D.4.3 TLB-pressure micro-bench

To produce direct evidence for the GPU-TLB-pressure mechanism that Appendix D.4.1 attributes the page-grain VMM's structural cost to, we ran a standalone micro-benchmark that removes the inference engine from the loop. The harness allocates 8 GiB via `cuMemAddressReserve+cuMemCreate+cuMemMap` (≈ 4096 2-MiB pages against the H200's per-SM TLB coverage of ≈ 1–2 GiB), wraps the range as a `torch.float32` tensor via `at::from_blob`, and times 8 consecutive streaming `tensor.sum()` reductions with `cuda.Event`. We compare a *cold* run (no warm-up) against a *warm* run (three prefatory `sum()` passes that walk every page).

The cold-mode first launch (25.76 **ms**) is $13\times$ slower than the warm-mode first launch (1.99 **ms**); after that, every launch in either mode runs in ≈ 1.96 ms. This is single-observation, statistic-free evidence that the cost is the very first kernel paying the page-table walks. Once those walks populate the per-SM L1 TLBs, the subsequent traffic is fully translated cache-resident and the structural cost evaporates.

Why we report latency rather than counters. Hopper's public PerfWorks catalog does not expose a TLB-miss metric (verified by `ncu -query-metrics -chip GH100 | grep -iE "tlb|page|mmu|translat"` returning zero matches across 5,646 metrics). We attempted to use `ncu` with proxy counters (`dram__bytes_read.sum`, `lts__t_sector_hit_rate.pct`, `smsp__warp_issue_stalled_long_scoreboard_per_warp_active.pct`, `sm__cycles_elapsed.avg`); the cold and warm modes match within $\pm 1\%$ on every metric. Nsight Compute collects each metric set by re-running the targeted kernel ≈ 10–$20\times$ on the GPU to gather counters, which averages the cold-TLB execution of launch 0 together with subsequent warm replays into a single reported number. NVIDIA's profiler model is therefore structurally

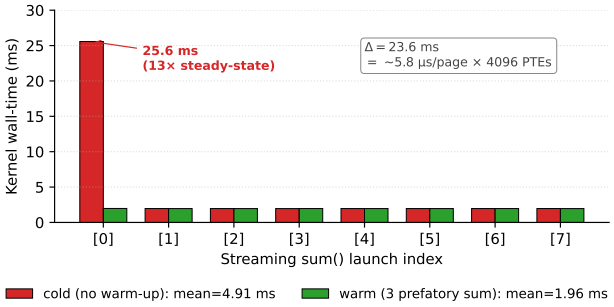


Figure 11: Standalone TLB micro-bench, 8 GiB `cuMemMap`'d range on H200, 8 consecutive streaming `sum()` reductions per run. Cold first launch is $13\times$ slower than steady state; the ≈ 23.6 ms cold-vs-warm gap on launch 0 is the direct cost of walking $\approx 4{,}096$ fresh 2 MiB-page table entries across the SMs that execute the kernel ($\approx 5.8\,\mu$s per page-walk). The transient evaporates after the first kernel; warm-mode launch 0 shows no spike. This is single-observation, statistic-free evidence that the variance source is GPU TLB.

incompatible with single-shot cold-TLB measurement on Hopper, and `cuda.Event` wall-time becomes the only direct probe.

D.5 Quality preservation

Four orthogonal quality tests confirm the system trades only latency, never quality: (Q1) greedy bit-identity, (Q2) sampled-decoding distribution match, (Q3) per-task classification accuracy, (Q4) ROUGE-L on long-form generation.

Q1: token-identity at `temperature`=0. 50 prompts (15 unique + duplicates to exercise prefix-cache reuse) at `temperature`=0, `seed`=0, `max_tokens`=128 to two server configurations on Qwen3.5-35B-A3B: engine baseline default and full system full. Result: **50/50 outputs byte-identical**. The intra-pool layer's eviction policy only changes which nodes survive; for any prompt that doesn't get evicted, the output is bit-exact. The inter-pool layer's cross-pool transfers preserve KV/recurrent state across reallocation by design.

Q2: sampled-decoding distribution match. 50 prompts at `temperature`=1.0, `top_p`=0.95 with three seeds (0, 7, 42) to both arms (150 outputs per arm). At sampled decoding, byte-identity does not survive a two-server-process comparison: tiny floating-point differences in logits (CUDA atomic ordering across server invocations) cascade through softmax into different sampled tokens. The relevant claim is that the *sampling distribution* is unchanged. A two-sample Kolmogorov–Smirnov test on per-output character-length distributions: **KS statistic 0.107, $p = 0.362 > 0.05$** — distributions statistically indistinguishable. Aggregate: mean char length 312.3 vs. 312.8 (within 0.2%), std 39.4 vs. 38.5, word-frequency cosine similarity 0.9846.

Q3: per-task classification accuracy. 50 multiple-choice questions on CS/networking/systems trivia at `temperature`=0, `seed`=0 to default vs. full. Both arms produce **49/50 (98%) correct**, with all 50/50 outputs byte-identical between arms (consistent with Q1). The single missed question is identical across arms.

Q4: ROUGE-L on long-form generation. 30 wildchat first-user-message prompts at `temperature`=1.0, `top_p`=0.95 with three seeds (90 outputs per arm). ROUGE-L F1 of each generation against the wildchat assistant reply: mean ROUGE-L 0.1243 (default) vs. 0.1477 (full system); two-sample KS test stat=0.144, $p = 0.306 > 0.05$ (statistically indistinguishable); paired t-test $p = 0.055$ (just below the 0.05 cutoff, full system slightly higher — well within run-to-run noise from sampling at `temperature`=1.0). Combined with Q1–Q3, the quality-preservation claim is supported on greedy bit-identity, sampled-distribution match, classification accuracy, and long-form ROUGE.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the two contributions evaluated in Section 4: (i) an intra-pool loss-per-byte eviction key (Section 3.1) and (ii) an inter-pool VMM-actuated transfer trigger (Sections 3.2 and 3.3). Scope is hybrid-attention serving with per-pool LPB+VMM, evaluated on three production hybrid models on H200 across long-horizon, swarm, traffic perturbation, and Claude Code agent workloads.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Section 4.1 discusses two scope limits: (i) HiMA sits on top of SGLang's Python scheduler, so the engine-level gap to vLLM on long-context multi-turn workloads (scheduler iteration cost, async batch dispatch) is out of HiMA's memory-management scope; (ii) the inter-pool layer is instantiated for two pools (KV vs. recurrent) and the formulation extends to $|\mathcal{I}| \geq 2$ but is only experimentally exercised at $|\mathcal{I}| = 2$.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: All theoretical results state their assumptions in the statement and have proofs in the appendix: Lemma 1 (per-pool loss convexity) is proven in Appendix C.2; Theorem 1 (Budgeter–Admitter coupling via ν^*) and Proposition 1 (shadow-price tracking under EWMA hysteresis) are proven in Appendix C.3; the closed-form marginal-price estimator of Eq. (5) is derived in Appendix C.1.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Appendix D.1 pins exact engine versions (SGLang v0.5.10.post2, vLLM v0.20.0), hardware (NVIDIA H200), models (Qwen3.5-35B-A3B, Qwen3-Next-80B-A3B-Instruct, Kimi-Linear-48B-A3B-Instruct), per-cell knob defaults, fixed random seeds, and trial counts (n=5 or n=3). Each workload's parameters (concurrency, $D(\text{in}, \text{out})$, RPS, wall-clock window) are fully specified. Workload generators and reproduction scripts are released alongside the engine fork.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: HiMA is built on top of an open-source SGLang fork that will be released with the camera-ready version. All baseline engines (SGLang v0.5.10.post2, vLLM v0.20.0) and models (Qwen3.5, Qwen3-Next, Kimi-Linear) are publicly available; the Claude Code trace replay dataset is the public archit11/complete-conversations-dataset, and SWE-Bench-Pro is run through the open Harbor harness. The supplemental will include reproduction scripts and exact CLI invocations for each table cell.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Appendix D.1 lists every hyperparameter: `mem_fraction_static`=0.8, hysteresis dead-zone $\Delta_{\text{hyst}} = 5\%$, Budgeter window $\tau = 30\,\text{s}$, net-benefit margin $\alpha = 1.5$, and per-cell engine flags. The static-best baseline is the per-workload optimum from a 4-ratio sweep. Each workload's full parameters (concurrency, $D(\text{in}, \text{out})$, RPS, token caps) are listed in the workload bullet list.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: All main-result cells are reported as mean $\pm$ std over n=5 trials (long-horizon, swarm, trace replay) or n=3 trials (traffic perturbation), with fixed random seeds per cell. Cost-curve fits use n=10 per recovery-length cell. Variability is across full trial repetitions (server boot + workload run), capturing run-to-run scheduler and admission noise.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: All experiments run on NVIDIA H200 GPUs (141 GB HBM each), with per-model TP listed in Appendix D.1 (Qwen3.5-35B-A3B at TP=1, Qwen3-Next-80B-A3B at TP=2, Kimi-Linear-48B-A3B at TP=2). Each cell runs for a 10-minute wall-clock window; trial repetitions and cell counts are specified per workload. The full project also consumed exploratory runs not reported in tables (e.g., flag-sanity sweeps).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: This work is a serving-system memory-management paper; it does not collect data from human subjects, does not use personal/private data, and does not deploy models in user-facing settings.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: HiMA is an inference-engine memory-management technique; it improves serving throughput and tail latency on existing public hybrid models. It does not introduce new model capabilities or data pipelines and therefore has no direct societal-impact path beyond the underlying models, which are out of scope.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper releases an inference-engine fork; no pre-trained models, datasets, or content-generating artifacts are released.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: SGLang and vLLM are cited (40, 21) and used under their Apache-2.0 licenses; Qwen3.5, Qwen3-Next, and Kimi-Linear are cited and used under their published model licenses (Tongyi Qianwen, Apache-2.0). The Claude Code conversation dataset is the public archit11/complete-conversations-dataset on Hugging Face; SWE-Bench-Pro is cited and run through the open Harbor harness.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The paper releases a single new asset — an SGLang fork that implements HiMA's intra-pool LPB key and inter-pool VMM-actuated transfer trigger, together with reproduction scripts. The fork ships with documentation describing flags, invocation, and per-table reproduction commands; submission-time anonymized URL is provided in the supplemental.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This work does not involve crowdsourcing or human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This work does not involve human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are the inference target of HiMA (the system serves them) but are not used as a component of the methodology. LLM-based writing assistance, where used, was limited to copy-editing and formatting and did not affect the methodology, results, or originality of the research.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.