UMAO: COUPLING CACHE LIFECYCLE AND ADMISSION FOR AGENTS ON UNIFIED MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-round agents continually append context, making cross-round reuse of computed prefixes essential for avoiding redundant prefill. On devices with a unified memory architecture (UMA), such as the NVIDIA DGX Spark, device and host share one physical memory pool. Expanding the host cache therefore does not increase total capacity; instead, it reduces the allocatable device KV, while retained histories, active KV, and cross-tier transfers compete for limited capacity. The system must maintain history usable in later rounds under fixed shared capacity and coordinate recovery and computation for competing requests. We propose unified memory agent offloading (UMAO), a framework that connects cache management and request admission through a shared recoverable-prefix state. Unified cache arena offloads long-term history to storage outside the shared pool, restricts host to bounded transfer staging, and commits and recovers history in continuous-prefix units. Cache-aware agent admission reads the prefix summary generated at the same scheduling boundary, jointly estimates direct-hit, cross-tier recovery, and recomputation work, and uses an age fence to limit tail latency. We implement UMAO as an SGLang HiCache plugin and evaluate it on DGX Spark with multiple real agent workloads. UMAO raises prefix reuse to 87.5%–90.3%, approaching the Ideal prefix-reuse upper bound across all workloads. Compared with existing HiCache write policies and longest-prefix match (LPM) admission, UMAO achieves up to $2.1\times$/$6.4\times$ speedups in mean/P99 time to first token (TTFT) and $1.6\times$/$2.9\times$ speedups in mean/P99 session completion time (JCT), respectively.

1 INTRODUCTION

As open-source large language models (LLMs) continue to improve, deploying complex agents locally on personal devices is becoming increasingly feasible, although such deployments depend heavily on single-device memory capacity (Sheng et al., 2023; Song et al., 2024; Alizadeh et al., 2024). Compact AI systems such as the NVIDIA DGX Spark adopt a unified memory architecture (UMA), providing a large single-node memory space for local model deployment and agent inference. Agent inference nevertheless creates substantial memory pressure: the attention key-value cache (KV cache) for computed prefixes must be retained for reuse during decode and subsequent rounds (Kwon et al., 2023), while the KV cache grows continuously with the context (Kwon et al., 2023; Zhang et al., 2023). Multi-round agents continually append context across inference, tool calls, and environment feedback (Yao et al., 2023; Yang et al., 2024). When historical context cannot be reused, every round repeats prefill (Abhyankar et al., 2024; Yu et al., 2025), causing redundant multi-round context computation.

Existing systems avoid redundant prefill through prefix reuse and hierarchical caching (Jin et al., 2025). PagedAttention organizes KV on device, the GPU-side KV cache tier, into a shareable paged index (Kwon et al., 2023). SGLang's RadixAttention indexes common prefixes with a radix tree and admits requests using longest-prefix match (LPM) based on the current matched length (Zheng et al., 2024). HiCache, LMCache, and Mooncake further extend the same prefix from device to host, the host-side KV cache tier, and storage (Zheng et al., 2024; Xie et al., 2026b; Cheng et al., 2025; Qin et al., 2025). These designs share an implicit assumption: device and host have independent capacities, so host provides an additional KV capacity tier and a larger host quota can retain more history when capacity is insufficient (Gao et al., 2024; Sheng et al., 2023).

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

This assumption holds when device and host are physically separate, but UMA breaks it. Device and host share one physical pool, so expanding host occupancy does not increase total capacity; it only reduces the device KV that can be allocated. Although storage lies outside the pool, its reads and writes still require host buffers and consume shared memory and bandwidth. The key problem is therefore no longer how to divide capacity between device and host. It is how to publish growing histories to storage while retaining active KV under fixed shared capacity, and how to control the staging and execution costs of subsequent recovery. This problem has two related aspects: how to maintain recoverable prefixes of multi-round agent histories under fixed shared capacity, and how to admit competing agent requests that share this capacity.

The first aspect concerns the cache. Existing hierarchical caches record write results at page granularity (Xie et al., 2026b; Cheng et al., 2025; Qin et al., 2025), whereas an agent history is an ordered, growing state chain. A page receiving a storage acknowledgment only indicates that it can be read independently. To restore the history and continue execution, every KV page from the beginning of the history to the target position must be available, and the recurrent state required by a hybrid architecture (Gu & Dao, 2023; Yang et al., 2025) must be aligned to the same prefix boundary. If any preceding page is missing, later pages that have already been persisted cannot independently form an executable prefix. Multi-round agent inference under fixed shared capacity therefore requires active management of the lifecycle of recoverable historical prefixes.

The second aspect concerns admission. Competing agent requests share a fixed capacity, so admission decisions must consider both the current prefix-match length and the subsequent service cost of each request and its impact on other sessions (Srivatsa et al., 2025). Existing admission orders requests by the current matched length, but the same matched length can correspond to very different remaining service costs: a direct device hit, cross-tier recovery from storage, or an uncached suffix that still requires prefill. Ordering only by length defers recovery and recomputation costs to execution and can leave some requests waiting for a long time. Request admission therefore needs to estimate remaining service work from the currently recoverable historical prefix and account for waiting time when ordering execution, balancing mean latency against tail waiting.

Cache lifecycle and request admission meet at the recoverable prefix: the former maintains history usable by later requests, and the latter uses that history to arrange recovery and computation. We therefore propose unified memory agent offloading (UMAO), which connects both to the same recoverable-prefix state. UMAO consists of two components. Unified cache arena (Arena) implements the data plane: device stores active KV, host provides bounded transfer staging, and storage carries long-term history; histories are committed and recovered as continuous prefixes. Cache-aware agent admission (CAA) implements the control plane: it reads the prefix summary generated by Arena at the current scheduling boundary, jointly estimates the work of direct hits, cross-tier recovery, and recomputation, and uses an age fence to limit tail latency. History maintenance, recovery, and admission thus operate on a consistent prefix boundary.

Our contributions are as follows: (1) We propose a coordinated design for cache lifecycle and request admission in multi-round agent serving on UMA, using the recoverable prefix as their shared state boundary. Under fixed shared capacity, the cache actively maintains history usable in later rounds, while admission estimates cross-tier recovery and remaining prefill work from the same prefix state, connecting history maintenance, recovery, and admission ordering. (2) We design and implement UMAO. Unified cache arena maintains cross-round histories through bounded host staging and compound commit, so recovery consumes only committed prefixes. CAA reads the prefix summary from the same scheduling boundary, estimates cross-tier recovery and recomputation work, and uses an age fence to suppress tail latency. (3) We implement UMAO as an SGLang HiCache plugin and evaluate it on multiple real agent workloads. UMAO raises prefix reuse to 87.5%–90.3%, approaching the Ideal prefix-reuse upper bound across all workloads. Compared with existing HiCache write policies and LPM admission, it achieves up to $2.1\times$/$6.4\times$ speedups in mean/P99 TTFT and $1.6\times$/$2.9\times$ speedups in mean/P99 JCT, respectively.

2 Preliminaries and Related Work

2.1 Hierarchical KV Caching under Unified Memory

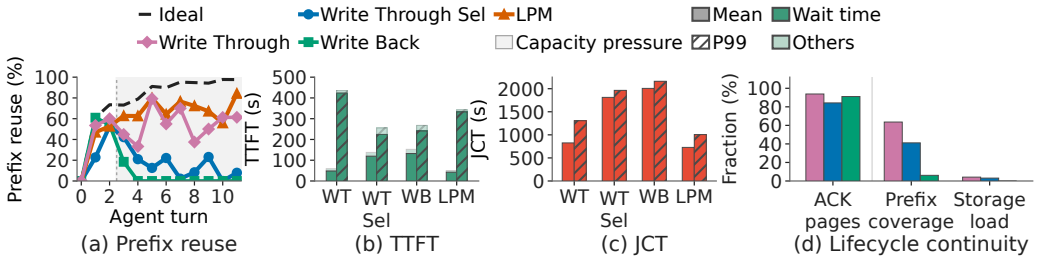

Figure 2: Motivation under fixed shared Device–Host capacity: (a) prefix reuse and the ideal bound, (b) TTFT, (c) session JCT, and (d) lifecycle continuity. WT, WT Sel, and WB denote Write Through, Write Through Sel, and Write Back; Others = TTFT - Wait time.

Hierarchical KV caching in existing systems is organized into the logical tiers device, host, storage, and remote. PagedAttention manages device KV in pages (Kwon et al., 2023), and RadixAttention indexes shared prefixes with a radix tree (Zheng et al., 2024). SGLang's HiCache and Strata use *Write Through*, *Write Through Sel*, and *Write Back* to control page writes to lower tiers (Zheng et al., 2024; Xie et al., 2026b). LMCache's `skip_l1` uses host as staging for lower-tier caching (Cheng et al., 2025), while Mooncake and CachedAttention retain session KV in storage (Qin et al., 2025; Gao et al., 2024).

In UMA, as shown in Figure 1, the CPU and GPU share a single finite, lower-bandwidth, non-expandable memory pool such as LPDDR. Model weights, activations, and the logical device/host caches all occupy this pool. Storage lies outside main memory, but its I/O typically still requires host buffers and consumes shared memory resources.

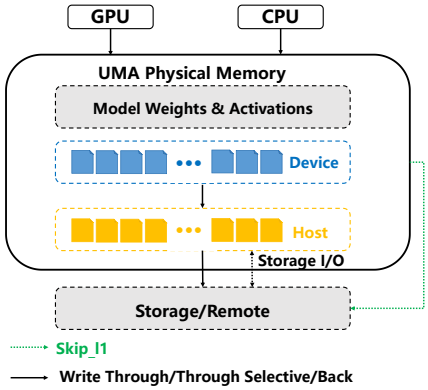

Figure 1: The cache hierarchy of UMA.

2.2 Cross-round Agent Prefix Reuse and Request Admission

For attention-only models, cross-round prefix reuse in multi-round agents primarily relies on the KV cache. Hybrid models (Gu & Dao, 2023; Yang et al., 2025) additionally require recurrent state and KV cache to reach the same prefix boundary. Once a historical prefix is evicted under capacity pressure, reuse requires pages to remain independently available in storage or another lower tier and the pages of the prefix to be recoverable in their original order as a continuous prefix. Prefix recoverability therefore determines whether cross-round reuse succeeds. Recovered pages still consume shared capacity and compete with active requests in UMA, constraining the number of requests that can execute concurrently.

Request admission commonly follows first-come, first-served (*FCFS*) or longest-prefix match (*LPM*) based on the prefix currently matched in device and host (Zheng et al., 2024). FastServe organizes queues by input length (Wu et al., 2024), while PEEK uses prefix and queue state for predictive scheduling (Xie et al., 2026a). Designed primarily for conventional systems with separate device and host memory, these policies do not jointly account for device/host hits, storage recovery, and remaining prefill under UMA. Appendix A.5 discusses related work in more detail.

3 Motivation

3.1 Reuse, Mean Latency, and Tail Tradeoffs under Fixed Shared Capacity

Under the hierarchical cache configuration in Section 5.1, we replay the Codex traces from Section 5 for 12 rounds, with 8 sessions competing in parallel for 2 execution slots. We define each round's input working set as the sum of the input lengths of all sessions in that round. The working set

grows from 113K tokens to 656K tokens, while the shared device–host cache capacity is 310K tokens; the peak working set reaches $2.12\times$ this capacity. The gray region in Figures 2(a) marks the rounds whose working set exceeds the shared cache capacity. We compare HiCache under *Write Through*, *Write Through Sel*, and *Write Back*, together with a variant of *Write Through Sel* that changes admission from FCFS to LPM. Figures 2(a)–(c) show that the Ideal upper bound defined in Appendix A.1.1, which is weighted by input tokens, is 87.9%, while the highest reuse among the executed configurations is 66.8% under LPM. LPM also yields the lowest mean TTFT, mean JCT, and P99 JCT, at 50.5 s, 727.9 s, and 1005.5 s, respectively, yet its P99 TTFT is 342.3 s, well above the 255.3 s of FCFS. These results show that, under capacity pressure, changing only the page write policy, or admitting requests solely by the current matched length, cannot simultaneously achieve a high reuse rate, low average latency, and low tail latency.

3.2 Recoverable Continuous Prefixes and Admission Tail Latency

Observation 1: High Page-ACK Rates Do Not Imply Continuous-Prefix Recoverability. To isolate cache write policies, we fix request order to *FCFS* and compare the three write policies. Figure 2(d) reports three lifecycle metrics at different granularities: the fraction of pages whose writes have been acknowledged by storage, labeled page-ACK coverage; the fraction of all input tokens covered by continuously recoverable historical prefixes, labeled continuous-prefix coverage; and the fraction of requests that hit a storage prefix, labeled storage load. Across the three policies, page-ACK coverage ranges from 84.2% to 93.8%, whereas continuous-prefix coverage is only 6.1%–63.6%, and the fraction of requests that hit storage does not exceed 4.2%. This gap is the key evidence for Observation 1: a page ACK indicates only that an individual page has been written to storage, whereas a continuous prefix requires every preceding page from the start of the history through the target position to be available in order. Cache lifecycle management must therefore maintain continuously recoverable prefix state beyond whether individual pages have received ACKs. Figure 2(d) characterizes recovery potential over the cache lifecycle, while the reuse rate in Figure 2(a) measures actual online reuse.

Observation 2: Prefix-Match Admission Exposes a Mean–Tail Trade-off. To isolate request admission order, we fix the *Write Through Sel* cache policy and compare FCFS with LPM. LPM reduces mean waiting time from 119.5 s to 42.4 s, but increases P99 waiting time from 224.2 s to 332.9 s. TTFT follows the same trend: its mean decreases from 137.6 s to 50.5 s, whereas its P99 increases from 255.3 s to 342.3 s. The mean waiting time drops substantially while the tail waiting time grows, showing that attending only to the current matched length cannot distinguish a direct hit, cross-tier recovery, or repeated prefill, and does not account for the waiting time of agent requests.

These observations lead to two connected design goals: maintaining recoverable prefixes under fixed shared-memory capacity and exposing the current prefix state to request admission. Arena maintains continuously recoverable history through bounded staging, incremental publication, and a recovery pipeline; CAA estimates service work from the same recoverable-prefix summary and constrains tail waiting.

4 UMAO Design

UMAO connects cross-round history maintenance and request admission through the same recoverable-prefix state. Figure 3 shows UMAO's overall architecture: shared cache capacity is repartitioned into device KV and host staging. Device stores the KV required for current execution, storage stores long-term history as sessions grow, and host staging performs only bounded transfers between device and storage.

When a request arrives, the native cache index first matches the prefix already present in device and host (Figure 3(1)). Arena queries storage for the farthest `committed` prefix anchor that extends the matched prefix and constructs the recoverable-prefix summary (2). CAA reorders waiting candidates using this summary (3). After the native engine completes resource checks, Arena recovers the selected prefix window by window through host staging (4); requests without a usable anchor follow normal prefill. After prefill and decode (5), the new suffix uses the same staging windows for publication (6), and its acknowledgments update the state visible at later scheduling boundaries.

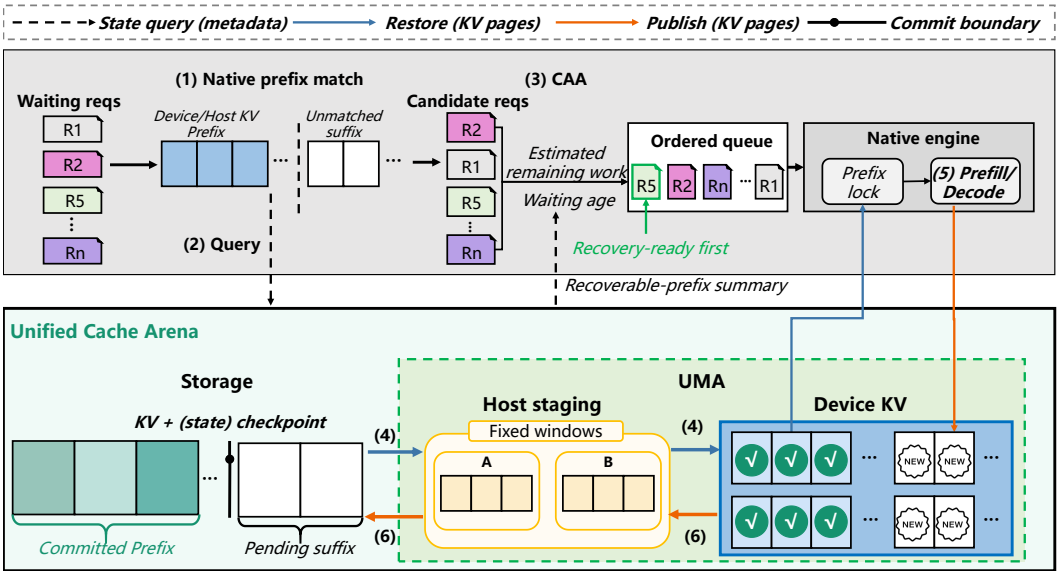

Figure 3: The overview of UMAO.

4.1 Unified Cache Arena

4.1.1 Shared-Capacity Repartitioning and Bounded Transfer Staging

Arena allocates shared capacity according to execution roles: device stores active KV, storage stores long-term history, and host provides transfer staging between device and storage. Under fixed shared capacity, let each page contain p tokens and each window contain w pages. Arena reserves two fixed pipeline windows, so host staging has a capacity of $2wp$ tokens and the remaining KV budget is assigned to device. State checkpoints of hybrid models are managed by the corresponding cache pool. The two windows are reused for publication and recovery, and the staging pool serves only one publication or recovery path at a time. Windows are returned after transfer, so staging occupancy does not grow with session count or history length. Window reuse and batched copies are detailed in Appendix A.2.2.

4.1.2 Round-Aligned Incremental Publication and Compound Commit

After each agent prefill, Arena compares the current page chain with the last observed prefix, identifies the new suffix, and re-enqueues previously incomplete pages for publication. The new suffix is transferred in windows at page granularity, with cache paths and token snapshots validated before transfer. A page becomes `durable` only after a complete storage ACK. A failed window can be published again in a later round as long as its pages remain available in device. Publication retries and state maintenance are detailed in Appendix A.2.1.

Arena maintains the commit boundary from persisted data. It scans storage ACKs in the order of the page prefix chain; when the continuous KV pages through an anchor and the state checkpoint required by a hybrid model have all been persisted, Arena advances the `committed` boundary for subsequent recovery and admission in Sec-

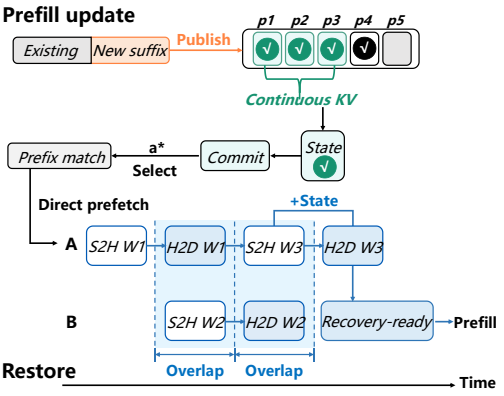

Figure 4: Arena lifecycle: compound commit enables continuous-prefix recovery through a two-window S2H/H2D pipeline.

tions 4.1.3 and 4.2. A gap holds the boundary at an earlier valid anchor regardless of ACKs for later pages. Figure 4 shows this workflow.

4.1.3 HIERARCHY-AWARE CONTINUOUS-PREFIX RECOVERY PIPELINE

Arena first matches the prefix already present in device and host through the native cache index, then queries storage for the farthest `committed` prefix anchor and uses that anchor as the recovery endpoint. Pages that have received storage ACKs but lie beyond the anchor are not included in recovery. If no usable anchor is available, the request follows normal prefill.

As shown in Figure 4, the prefix on the recovery path is partitioned into windows of w pages, as defined in §4.1.1, and transferred by overlapped pipelining: two fixed windows alternately perform storage-to-host (S2H) and host-to-device (H2D), so the current window is written to device while the next window's prefix is read from storage. For hybrid models, the required state checkpoint is recovered with the final window. The request then enters `recovery-ready` and begins preparing prefill. Per-tier hits, the remaining prefill, the recovery-window count, and the queue state are then summarized for CAA. Staging arbitration, the recovery control path, and lock handoff are described in Appendix A.2.2.

To reduce control-plane overhead, the two windows share a pinned host staging pool created at service initialization. For S2H, Arena obtains host-page indices directly from this pool and constructs prefetch operations before submitting storage reads, reducing the control overhead of allocating host pages after hit matching on the default path. For H2D, Arena uses batched `cudaMemcpyBatchAsync` operations to accelerate writes to device. The effect of the pipeline, S2H, and H2D optimizations on the recovery critical path is evaluated with real workloads in Section 5.4, Figure 7(b).

4.2 CACHE-AWARE AGENT ADMISSION

4.2.1 FROM CACHE SUMMARY TO SERVICE WORK

After native prefix matching, Arena creates a recoverable-prefix summary for each waiting agent request. The summary records device/host hits, the recoverable committed storage prefix, uncached tokens, recovery-window count, and recovery-queue depth, including active recovery, queued recovery, and in-progress publication requests. CAA uses one snapshot for the entire reordering operation so that cache semantics do not change during sorting.

For request r, CAA estimates prefill work $P(r)$ for the uncached suffix, host-hit cost $H_r p_H$, storage recovery time $R(r)$, and recovery-queue waiting $D(r)$. $P(r)$ accumulates fixed, per-token, and prefix-position costs over chunks; this chunked-prefill decomposition follows the serving granularity used in prior work (Agrawal et al., 2024). $R(r)$ uses the S2H/H2D single-window EWMA to estimate the two-window pipeline, and $D(r)$ is a first-order estimate from the recovery-queue depth. Before online statistics are available, $R(r)$ falls back to the product of the storage-token count S_r and static cost p_S, and $D(r) = 0$. The model-level coefficients, independent calibration protocol, EWMA update, and snapshot timing are given in Appendix A.3.1. The total admission work is

$$W(r) = P(r) + H_r p_H + R(r) + D(r).$$

These formulas map the path costs of waiting requests into a common numerical ordering space. CAA uses them to determine admission order, and the native engine performs resource checks and final execution. The coefficients of $P(r)$ are calibrated by independent prefill position probes, $R(r)$ corresponds to the measured stages of the two-window pipeline, and $D(r)$ is computed from the recovery-window count and queue depth before ordering. Complete component formulas, the admission algorithm, and conditional analysis are given in Appendices A.3.1–A.3.3.

4.2.2 JOINT ORDERING BY CACHE WORK AND AGING

CAA includes waiting-queue requests in the candidate set, sorts them by nondecreasing $W(r)$, and breaks ties by arrival time and request identifier. Let r_{old} be the earliest-arriving request and r_{head} be the head after work-based sorting. When the waiting time of r_{old} reaches $\rho \max\{W(r_{\text{head}}), f\}$, the age fence promotes r_{old} to the head, where f is the fixed overhead of one prefill chunk and ρ controls the aging trigger speed. If a `recovery-ready` request exists, CAA selects the highest-ranked such request and prioritizes its handoff, avoiding waste from completed recovery work waiting for a long

time. The final priority order is recovery handoff, age fence, and $W(r)$ ordering. The conditional property of the age fence is given in Appendix A.3.3, and the complete pseudocode for Algorithm 1 is given in Appendix A.3.4.

CAA's role ends at reordering waiting agent candidates. Resource checks, prefill, and decode remain with the native execution engine. At a single scheduling boundary, if the candidates contain K_r uncached chunks, the computation cost is $O(\sum_r K_r)$, the sorting cost is $O(N \log N)$, and the extra request metadata is $O(N)$. With maximum context and chunk size fixed, the complexity by candidate count is $O(N \log N)$.

5 Evaluation

5.1 Methodology

Testbed. We implement UMAO as a HiCache plugin for SGLang v0.5.16 (Zheng et al., 2024). All experiments run on an NVIDIA DGX Spark with a Blackwell GB10 GPU and a 20-core Arm CPU sharing 128 GB of LPDDR5x unified memory at 273 GB/s. Storage uses the NIXL backend. The main experiments use an approximately 310K-token device+host KV budget; the exact cache partition and window configuration are given in Appendix A.1, and the runtime memory headroom is characterized in Appendix A.4.3.

Models and workloads. We replay complete sessions from the InferAct `codex_swebenchpro_traces` (Codex sessions on SWE-bench Pro) (Liu & Inferact, 2026; Deng et al., 2026) and Lambda `hermes-agent-reasoning-traces` (Mueller & Lambda, 2026) on Qwen3.5-35B-A3B (Qwen Team, 2026) and GPT-OSS-20B (OpenAI, 2025). We use C_K to denote the number of simultaneously active agent sessions. Thus, C4, C6, and C8 denote 4, 6, and 8 simultaneously active sessions, respectively. Trace sources and replay statistics are summarized in Appendix A.1.

Baselines. We compare HiCache's *Write Through*, *Write Through Sel*, and *Write Back* configurations under *FCFS*, a configuration that uses native longest-prefix match (*LPM*) admission with *Write Through Sel* (Zheng et al., 2024; Xie et al., 2026b), and the complete system, *UMAO*. All configurations use the same model, traces, concurrency, execution slots, and NIXL backend. They vary in write and recovery paths, admission order, device/host capacity allocation, and host's role. Section 5.3 and Appendix A.4.1 examine Arena's overall benefit against matched-capacity controls.

Metrics. Prefix reuse rate is the fraction of input tokens covered by reused cached prefixes, weighted by request input length. TTFT and JCT both report mean and P99, and session JCT measures the duration from session activation to completion of the final round, including inter-round delays. The P99 sample units and counts are specified in Appendix A.1, and the analytical Ideal prefix-reuse upper bound is defined in Appendix A.1.1.

5.2 End-to-End Performance

Figure 5 summarizes the overall performance of the four workloads formed by two models and two traces at C4, C6, and C8. *UMAO* achieves the highest prefix reuse rate in all 12 settings, ranging from 87.5% to 90.3% and remaining close to the Ideal prefix-reuse upper bound as concurrency increases from C4 to C8. In contrast, the three *FCFS* HiCache write policies achieve only 2.6%–56.2% at C8. Although *LPM* raises reuse to 79.3%–85.2%, it still falls below *UMAO*. These results show that UMAO maintains cross-round prefix reuse under the tested shared-capacity and concurrency settings.

We next compare request response latency. Against the best baseline for each metric, *UMAO* accelerates mean TTFT over *LPM* by $1.3\times$–$2.1\times$ and P99 TTFT over *Write Through Sel* or *Write Through* by $2.7\times$–$6.4\times$. In the highest-pressure C8 workloads, its P99 TTFT speedup over the best baseline remains $2.7\times$–$5.8\times$; over *LPM*, the speedup is $4.1\times$–$17.7\times$. This agrees with Observation 2 in Section 3.2: ordering by matched length can reduce mean waiting while concentrating cost in the tail.

Request-level gains further accumulate into complete agent sessions. Relative to the best baseline, *LPM*, *UMAO* accelerates mean JCT by $1.2\times$–$1.6\times$ and P99 JCT by $1.4\times$–$2.9\times$. UMAO thus improves individual response times and shortens the overall completion time of multi-round agents.

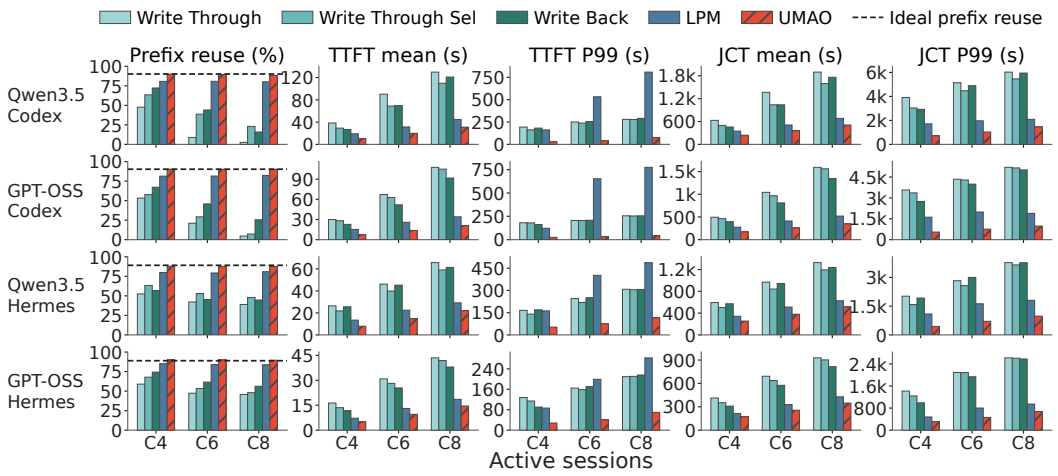

Figure 5: Performance: prefix reuse, TTFT, and session JCT across models, policies, and C4–C8. C4, C6, and C8 denote 4, 6, and 8 simultaneously active agent sessions, respectively.

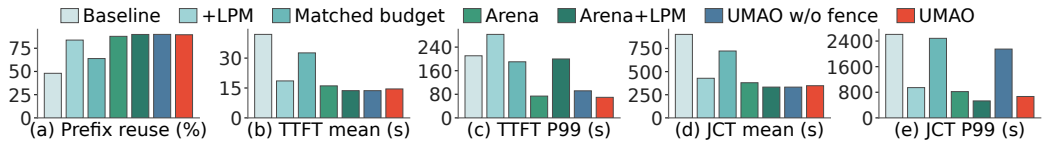

Figure 6: Ablation: seven GPT-OSS-20B & Hermes configurations at C8 under a 310K-token budget.

5.3 Ablation Study

To separate the contributions of cache lifecycle and request admission, we compare seven configurations on the GPT-OSS-20B, Hermes C8 setting in §5.2, with *Write Through Sel*, the total device+host capacity, and the execution slots held fixed. *Baseline* uses the native Radix cache index and *FCFS*. *+LPM* and *Matched budget* change only the admission order and host/device allocation, respectively. *Arena* enables incremental publication, compound commit, and continuous-prefix recovery on the matched capacity allocation with *FCFS*. *Arena+LPM* uses *LPM*, *UMAO w/o fence* retains recovery-ready handoff and sorts by $W(r)$, and complete *UMAO* additionally enables the age fence.

Arena maintains continuously recoverable historical prefixes. As shown in Figure 6, *Matched budget* increases reuse by 33.1% relative to *Baseline*, but reduces P99 JCT by only 4.7%. With the same capacity allocation, enabling Arena further increases reuse by 37.4% and reduces mean TTFT, P99 TTFT, and P99 JCT by 50.7%, 61.1%, and 67.0%, respectively. *Matched budget* does not achieve the same gains as *Arena*, indicating that capacity reallocation explains only part of the improvement; the remaining gains are consistent with Arena's lifecycle management through incremental publication, compound commit, and continuous-prefix recovery. Appendix A.4.1 further compares eager page persistence under matched capacity.

CAA reduces mean latency and mitigates LPM's long TTFT tail. Figure 6 compares three admission policies under the same Arena configuration. Compared with FCFS (*Arena*), the full CAA configuration (*UMAO*) reduces mean TTFT and mean JCT by 9.6% and 8.5%, respectively, and P99 TTFT and P99 JCT by 5.7% and 18.6%. Compared with LPM (*Arena+LPM*), CAA increases mean TTFT by only 6.3% while reducing P99 TTFT by 65.1%. Removing the age fence (*UMAO w/o fence*) reduces mean TTFT by only 6.3%, but increases P99 TTFT and P99 JCT by 31.7% and 221.7%, respectively. These results show that CAA improves mean latency over FCFS and mitigates LPM's long TTFT tail, while the age fence limits prolonged waiting caused by work-based ordering.

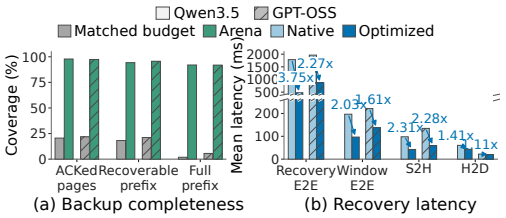

Figure 7: Arena backup completeness and recovery-path performance on Hermes at C8. Full prefix denotes complete-prefix recovery across rounds; Recovery E2E is request-level, whereas Window E2E is window-level.

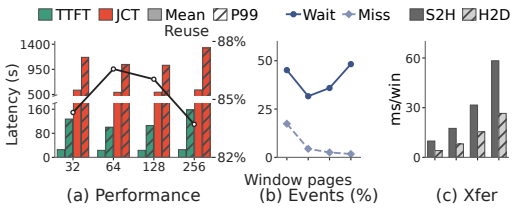

Figure 8: Arena window-size sensitivity on Qwen3.5-35B-A3B & Hermes at C8. Wait and Miss denote host-staging wait and checkpoint-miss events, respectively, normalized per 100 restore requests, S2H and H2D are window-level.

5.4 Mechanism and Sensitivity Analysis

Arena increases continuous-prefix coverage and reduces recovery-path latency. Figure 7 verifies both effects in the complete Hermes C8 replay from Section 5.2. Figure 7(a) uses the *Matched budget* and *Arena* configurations from Section 5.3. On all non-initial rounds, relative to *Matched budget*, *Arena* raises page-ACK coverage, continuous-prefix coverage, and the complete-prefix ratio by at least 75.6, 73.2, and 86.3 percentage points, respectively. Arena's incremental publication and compound commit thus improve backup completeness and prefix recoverability. Figure 7(b) compares *Arena* with and without the recovery-path optimizations in Section 4.1.3, pairing requests with identical recovery-token counts. Mean Recovery E2E accelerates by $3.75\times$ for Qwen3.5-35B-A3B and $2.27\times$ for GPT-OSS-20B; mean per-window recovery latency decreases by $2.03\times$ and $1.61\times$, respectively. S2H accelerates by approximately $2.3\times$ for both models, while H2D accelerates by $1.41\times$ and $1.11\times$, indicating that the main gains come from S2H optimization and the recovery pipeline.

Sensitivity to window size. Figure 8 fixes Qwen3.5-35B-A3B, Hermes C8, and the remaining settings while varying the recovery-window size from 32 to 256 pages. Prefix reuse varies between 83.7% and 86.6%, and mean TTFT and mean JCT remain close. The 64- and 128-page settings nearly overlap, whereas 256 pages produces higher P99 TTFT and P99 JCT. Smaller windows incur more state-checkpoint misses; larger windows increase per-window transfer time, host-staging wait, and device KV consumption. The main experiments use 128 pages, which lies in this flat region.

Sensitivity to age fence. Figure 9 fixes the Qwen3.5-35B-A3B & Hermes C8 replay and compares a disabled age fence with $\rho \in \{0.25, 1, 4, 16\}$. We additionally report P99/P50 to characterize tail amplification. Without the age fence, the P99/P50 JCT ratio is 13.08; with it enabled, the ratio falls to approximately 2. The four enabled settings have similar results, and $\rho = 1$ gives the lowest P99/P50 ratio. Relative to *off*, $\rho = 1$ reduces P99 JCT by 68.8% and P99 TTFT by 27.0%, while increasing mean TTFT and mean JCT by only 5.4% and 4.0%; prefix reuse is nearly unchanged. The main experiments therefore use $\rho = 1$. For extended evaluation, see Appendix A.4.

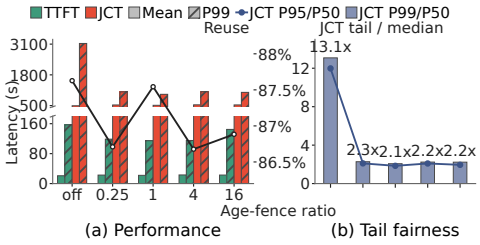

Figure 9: Age-fence sensitivity on Qwen3.5-35B-A3B & Hermes at C8.

6 Conclusion

We propose UMAO for multi-round agents serving under a fixed unified memory budget: Unified cache arena maintains recoverable prefixes through bounded host staging, incremental publication, compound commit, and continuous-prefix recovery. Cache-aware agent admission estimates remaining service work from the same recoverable-prefix summary and uses an age fence to mitigate prolonged waiting. On real agent traces, UMAO raises prefix reuse to 87.5%–90.3%. Compared with existing HiCache write policies and LPM admission, it achieves up to $2.1\times/6.4\times$ speedups in mean/P99 TTFT and $1.6\times/2.9\times$ speedups in mean/P99 JCT.

AI USE STATEMENT

Large language models were used only to assist the language expression of this paper, including grammar correction, sentence rewriting, and style polishing. The research questions, hypotheses, methods, implementation, experiments, analysis, and conclusions were completed by the authors without generative AI assistance. All AI-assisted text was carefully reviewed and verified by the authors, who take full responsibility for the final content of this work.

REFERENCES

Reyna Abhyankar, Zijian He, Vikranth Srivatsa, Hao Zhang, and Yiying Zhang. Infercept: Efficient intercept support for augmented large language model inference. *arXiv preprint arXiv:2402.01869*, 2024.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 117–134, 2024.

Keivan Alizadeh, Seyed Iman Mirzadeh, Dmitry Belenko, S Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. Llm in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12562–12584, 2024.

Yihua Cheng, Yuhan Liu, Jiayi Yao, Yuwei An, Xiaokun Chen, Shaoting Feng, Yuyang Huang, Samuel Shen, Kuntai Du, and Junchen Jiang. Lmcache: An efficient kv cache layer for enterprise-scale llm inference. *arXiv preprint arXiv:2510.09665*, 2025.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa R Kundurthy, Sean M. Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-bench pro: Can AI agents solve long-horizon software engineering tasks? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=uEVTdoAbnK.

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

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Shipeng Hu, Guangyan Zhang, Yuqi Zhou, Yaya Wei, Ziyan Zhong, and Jike Chen. Bidaw: Enhancing Key-Value caching for interactive LLM serving via bidirectional Computation–Storage awareness. In *24th USENIX Conference on File and Storage Technologies (FAST 26)*, pp. 101–116, Santa Clara, CA, February 2026. USENIX Association. ISBN 978-1-939133-53-3. URL https://www.usenix.org/conference/fast26/presentation/hu-shipeng.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 44(1):1–27, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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*, pp. 611–626, 2023.

Hanchen Li, Qiuyang Mang, Runyuan He, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Alvin Cheung, Joseph E. Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. In *ICLR 2026 Workshop on Lifelong Agents: Learning, Aligning, Evolving*, 2026. URL https://openreview.net/forum?id=sVzK0LC9pn.

Zijing Liu and Inferact. Codex SWE-bench pro traces, 2026. URL https://huggingface.co/datasets/Inferact/codex_swebenchpro_traces. MIT. Revision 0d52ae8. Accessed 2026-09-21.

Zachary Mueller and Lambda. Hermes agent reasoning traces, 2026. URL https://huggingface.co/datasets/lambda/hermes-agent-reasoning-traces. Apache-2.0. Revision b92885e. Accessed 2026-09-21.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Zaifeng Pan, AJJKUMAR DAHYALAL PATEL, Yipeng Shen, Zhengding Hu, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows. *Advances in Neural Information Processing Systems*, 38:126246–126265, 2026.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pp. 155–170, Santa Clara, CA, feb 2025. USENIX Association. ISBN 978-1-939133-45-8. URL https://www.usenix.org/conference/fast25/presentation/qin.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

Yakov Pyotr Shkolnikov. Agent memory below the prompt: Persistent q4 kv cache for multi-agent llm inference on edge devices. *arXiv preprint arXiv:2603.04428*, 2026.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. Powerinfer: Fast large language model serving with a consumer-grade gpu. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 590–606, 2024.

Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for llm serving. In *International conference on learning representations*, volume 2025, pp. 37057–37082, 2025.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models, 2024. URL https://arxiv.org/abs/2305.05920.

Bing Xie, Zhipeng Wang, Masahiro Tanaka, and Zheng Zhen. Peek: Predictive queue-informed kv cache management for llm serving. *arXiv preprint arXiv:2607.02525*, 2026a.

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

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 144–158. ACM, March 2025. doi: 10.1145/3689031.3696086. URL http://dx.doi.org/10.1145/3689031.3696086.

Rui Zhang, Chaeeun Kim, Shaoting Feng, Kuntai Du, Yuhan Liu, Yi Zhong, Cheng-Wei Ching, Junchen Jiang, and Liting Hu. Learning agent execution for kv-cache management in agentic serving, 2026. URL https://arxiv.org/abs/2608.14624.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in neural information processing systems*, 36:34661–34710, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Appendix

A.1 Experimental Setup

This section supplements the implementation scale, replay protocol, cache budget, and trace statistics omitted from Section 5.1. The settings below apply to the 12 model, trace, and concurrency combinations in Figure 5.

Implementation. UMAO is integrated as an SGLang v0.5.16 HiCache plugin and hooks both cache lifecycle management and request admission. The source code and tests comprise approximately 9.6K lines of Python, including approximately 3.2K lines for the core unified cache arena and CAA implementations.

Testbed. The platform is an NVIDIA DGX Spark with a Blackwell GB10 GPU, a 20-core Arm CPU, 128 GB of LPDDR5x unified system memory at 273 GB/s, and local NVMe storage. Device KV and host cache consume the same physical capacity, while storage lies outside shared main memory. All configurations enable hierarchical caching, use NIXL as the storage backend, and use `wait_complete` as the prefetch policy.

Capacity configuration. The main experiments use a unified device+host KV budget of 309,696 tokens with a page size of 64 tokens. Native HiCache assigns 147,456/162,240 tokens to device/host; UMAO reserves 16,384 tokens for two 128-page host staging windows and assigns the remaining 293,312 tokens to device. For Qwen3.5-35B-A3B, the state-checkpoint budget is 101 slots. Native HiCache and UMAO assign 48/53 and 95/6 slots to device/host, respectively. The *Matched budget* configuration in Section 5.3 uses UMAO's capacity partition; the window-sensitivity experiment fixes total capacity and adjusts host staging and device allocations with the window size.

Models. Qwen3.5-35B-A3B contains state-space components, so a recoverable prefix requires KV pages and state checkpoints to reach the same boundary. GPT-OSS-20B is a FULL/SWA hybrid-attention MoE model. Together, the models test whether compound commit depends on a particular cache-state type.

Workloads. We evaluate two real multi-round agent traces: InferAct's `codex_swebenchpro_traces`, recording Codex sessions on SWE-bench Pro, and Lambda's `hermes-agent-reasoning-traces`, recording multi-round Hermes reasoning sessions. Figures use the names Codex SWE-bench Pro and Hermes Reasoning. Each workload contains 50 sessions, replayed from round 0 through all assistant turns; Qwen3.5 and GPT-OSS use the same session cohort and turn counts. Figure 10 uses the Qwen3.5 tokenizer for both traces so that their

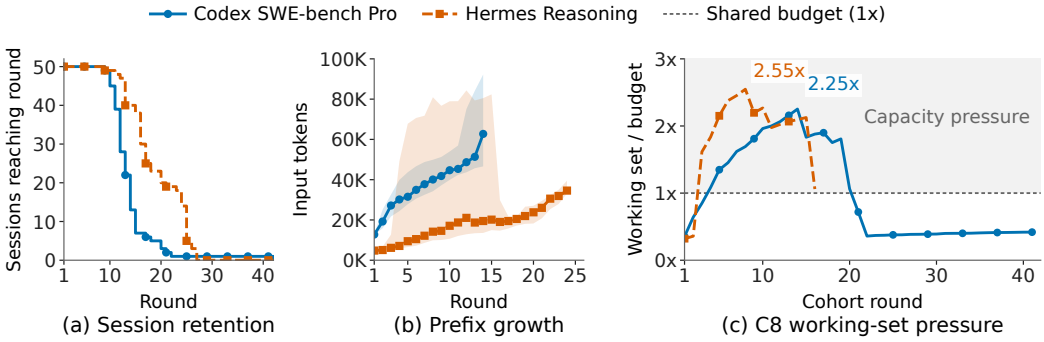

Figure 10: Evaluation traces. (a) Number of selected sessions reaching each assistant round. (b) Median input tokens by round; bands show the cross-session interquartile range, and only rounds reached by at least 10 sessions are included. (c) Complete aligned input-prefix sum for the first eight concurrent sessions relative to the 309,696-token shared budget; the shaded region marks demand above capacity. Codex SWE-bench Pro and Hermes Reasoning each contain 50 sessions. Token counts use the Qwen3.5 tokenizer; Qwen3.5 and GPT-OSS share the same sessions and turn counts.

logical demand is compared under the same token definition. Codex has 651 turns in total, with 13.0 turns on average and 12.0 at the median. Hermes has 910 turns in total, with 18.2 turns on average and 16.5 at the median.

Trace	Sessions	Turns	Avg. turns	Median turns	Input tokens
Codex SWE-bench Pro	50	651	13.0	12.0	12.3K–129.6K
Hermes Reasoning	50	910	18.2	16.5	4.7K–124.5K

Figure 10 shows that both traces enter capacity pressure early, but Hermes sessions are longer and its logical working set grows faster: Codex exceeds the shared budget at round 4 and peaks at 697.5K tokens ($2.25\times$), while Hermes exceeds it at round 3 and reaches 789.0K tokens ($2.55\times$) at round 8. These statistics describe aligned logical demand rather than physical cache occupancy and explain the C8 pressure in Section 5.

Reproducibility and replay protocol. All workloads use fixed session traces and the same selected session cohort for both models. Each workload contains 50 sessions, replayed from round 0 through all assistant turns. C4, C6, and C8 denote the number of simultaneously active sessions; a new session enters when one finishes. The server uses two fixed execution slots, an inter-round interval of 1 s, and a 128-token completion-length limit. All configurations use the NIXL storage backend with `wait_complete` prefetching. These controls fix the trace, cohort, concurrency, execution capacity, inter-round timing, output-length limit, and storage path across comparisons.

Baselines. *Write Through*, *Write Through Sel*, and *Write Back* retain *FCFS* to test whether changing only page-write timing can preserve cross-round reuse of continuously recoverable prefixes. *LPM* retains the *Write Through Sel* publication path and changes only server admission to sort by the currently matched prefix length, testing whether this order can reduce both average and tail waiting. *UMAO* enables Arena's incremental publication and continuous-prefix recovery together with CAA's work-and-aging admission. Except for write policy, admission order, and host role allocation, all configurations use the same model, trace, concurrency, execution slots, and NIXL backend.

Metric aggregation. Prefix reuse rate is weighted by request input length and measures history actually reused online, rather than the fraction of storage pages that have received an ACK. Mean TTFT measures average response latency and P99 TTFT captures the tail under capacity pressure. For each model–trace–concurrency–configuration tuple, P99 TTFT is computed over 651 request-level observations for Codex or 910 for Hermes, whereas P99 session JCT is computed over 50 session-level observations. Session JCT starts at session activation and ends at final-round completion, including inter-round delays. Recovery E2E in Figure 7(b) is paired per request, while Window

E2E, S2H, and H2D are measured per recovery window. Publication-side D2H/H2S in Figure 12 is measured per publication, S2H as KV I/O, and H2D as a CUDA copy.

A.1.1 IDEAL PREFIX-REUSE UPPER BOUND

Let $L_{s,t}$ denote the input-token length of turn t in session s, with $t = 0$ denoting the first turn, and let T_s be the number of turns in that session. The Ideal prefix-reuse upper bound assumes that every later prompt can reuse the complete prompt from the preceding turn in the same session. Its input-token-weighted value is

$$R_{\text{ideal}} = 100 \times \frac{\sum_{s} \sum_{t=0}^{T_s-2} L_{s,t}}{\sum_{s} \sum_{t=0}^{T_s-1} L_{s,t}}.$$

The numerator counts the preceding prompt available for reuse at each later turn, while the denominator counts all input tokens. This is an analytical upper bound derived from the selected traces, rather than a runnable serving configuration; it does not assume or measure storage bandwidth, persistence timing, failed writes, shared-memory recovery pressure, or queueing. Because the selected session cohort is shared across C4, C6, and C8, the bound is unchanged across concurrency levels for a given model–trace workload.

A.2 ARENA IMPLEMENTATION DETAILS

This section supplements the page-state maintenance, staging-window reuse, and recovery handoff omitted from Section 4.1, and explains how page states form a `committed` anchor.

A.2.1 INCREMENTAL PUBLICATION AND PAGE-STATE MAINTENANCE

After each agent prefill update, Arena compares the current ordered page chain with the last observed prefix, identifies the new suffix, and adds incomplete pages that remain available through the current cache path to the publication queue. Candidate pages are divided into fixed-size windows, with cache paths and token snapshots validated before transfer. A page moves from `pending` to `in-flight` and becomes `durable` after a complete storage ACK. A failed window remains incomplete and is published again in a later round if its pages are still available.

Arena maintains the prefix commit boundary from object persistence states. Let the current observed ordered history be $h = (p_1, \ldots, p_n)$, and let S_a be the set of endpoint states required by anchor a; for KV-only models, $S_a = \emptyset$. Writing $\text{durable}(x)$ for a complete storage ACK for object x, the compound commit condition is

$$\text{committed}(a) \iff \left(\bigwedge_{i \le a} \text{durable}(p_i) \right) \wedge \left(\bigwedge_{s \in S_a} \text{durable}(s) \right).$$

Arena checks page completion in history order and combines it with the endpoint states required at each anchor to find the farthest committed anchor. Thus, `committed` is a derived state determined jointly by continuous KV pages and endpoint state. Publication and retries update object persistence states, while the commit check summarizes them as the prefix boundary used by recovery and request summaries. If an incomplete page is no longer available through the current cache path or its token snapshot no longer matches, the current recovery range remains at an earlier committed anchor and the remaining suffix follows normal prefill.

A.2.2 TWO-WINDOW STAGING AND RECOVERY HANDOFF

Arena allocates a pinned host staging pool during service initialization and keeps two alternating windows. Publication and recovery share these windows and are allocated through the same arbitration path; a window is returned immediately after transfer, so host staging capacity remains $2wp$ tokens. The cross-tier KV in each window is written to device using batched H2D copies, and H2D for the current window can overlap storage reads for the next.

Table A.1: Model-level work coefficients used by CAA. Units are μs/Chunk for f, μs/token for p_u, p_H, and p_S, and μs per full-Chunk-position token for p_s. p_S is a static fallback for online recovery statistics; after S2H and H2D observations become available, $R(r)$ uses a window-level EWMA.

Model	f	p_u	p_s	p_H	p_S
Qwen3.5-35B-A3B	170,880.053	121.421	5.023241	0.0	20.0
GPT-OSS-20B	45,651.038	218.033043	8.988349	0.7	7.2

Once a candidate passes the native resource check, Arena starts recovery of its committed prefix. After the final window completes, the request enters `recovery-ready`; Arena retains a reference to the recovered prefix until the native execution path acquires the prefix lock, then releases its protection. Requests without a committed anchor do not start recovery and follow normal prefill. CAA consumes this snapshot without reserving extra device KV or batch capacity.

A.3 CAA Cost Model and Scheduling Properties

This section supplements Section 4.2 with work decomposition, the complete admission algorithm, ordering properties, and aging. The analysis considers one scheduling boundary, during which Arena's recoverable-prefix summary and cost estimates remain fixed.

A.3.1 Prefill and Recovery Work

The main experiments use chunk size $m = 2{,}048$ tokens, and all work estimates are in microseconds. Table A.1 lists the model-level coefficients used by the complete UMAO system; $\rho = 1$ is used as the main age-fence ratio.

The coefficients f, p_u, and p_s come from independent chunk-prefill position probes on the same DGX Spark, rather than from Codex or Hermes replay latency or ordering results. The probes vary the matched-prefix position and uncached-suffix length and fit the server-side forward time in Eq. (A.1). Qwen3.5 uses 15 valid records from 18 probes, with grouped cross-validation MAE 14.6 ms and fitted $R^2 = 0.9970$; GPT-OSS uses 320 idle-subset records, with MAE 10.8 ms and $R^2 = 0.9977$. Host and storage static unit costs are loaded with the model configuration. The static p_S provides the startup fallback until online recovery statistics become available.

Arena updates S2H and H2D estimates after each recovery window. Given previous estimate e_{t-1} and new sample z_t, it uses $e_t = 0.8e_{t-1} + 0.2z_t$, with the first valid sample initializing the estimate. At each scheduling boundary after native prefix matching, Arena creates a request-level snapshot containing the committed storage prefix, recovery-window count, queue depth, and current $R(r)$. The snapshot is used only for the current CAA ordering and is recomputed at the next scheduling boundary.

Let U_r be the uncached prefill tokens, x_r the continuous matched-prefix length, and $K_r = \lceil U_r/m \rceil$. For chunk j, let $u_{r,j} = \min\{m, U_r - jm\}$. Then

$$P(r) = \sum_{j=0}^{K_r-1} \left[f + p_u u_{r,j} + p_s \frac{(x_r + jm)u_{r,j}}{m} \right]. \tag{A.1}$$

For $k_r > 0$ storage recovery windows, let a and b be the S2H and H2D per-window EWMAs in the same units as $W(r)$. CAA estimates

$$R(r) = a + b + (k_r - 1)\max\{a, b\}, \qquad k_r > 0. \tag{A.2}$$

The complete UMAO system uses the recovery critical-path estimate above. When runtime statistics are unavailable, it falls back to $R(r) = S_r p_S$, where S_r is the storage-token count. With H_r host-hit tokens and per-token cost p_H, and recovery-queue depth q_r,

$$D(r) = \begin{cases} q_r R(r)/\max\{k_r, 1\}, & \text{EWMA available,} \\ 0, & \text{static fallback,} \end{cases} \qquad W(r) = P(r) + H_r p_H + R(r) + D(r). \tag{A.3}$$

Lemma 1 (Recovery pipeline). If each window passes through S2H and H2D stages with constant times a and b, and adjacent windows can be pipelined, the completion time for k windows is given by Eq. (A.2).

Proof. The first window completes at $a + b$. Each subsequent completion is separated by the slower stage, $\max\{a, b\}$, giving $a + b + (k - 1)\max\{a, b\}$.

A.3.2 Optimality of Work-Based Ordering

Assumption 1 (Single-boundary service model). At one scheduling boundary, N arrived candidates are served by a single non-preemptive sequence with no new arrivals. Their true remaining service times $s_i \geq 0$ do not depend on order, the snapshot work estimates $W(i)$ are fixed, and no age-fence or recovery-ready handoff is triggered.

For order $\pi = (\pi_1, \ldots, \pi_N)$, the total completion time from the boundary is

$$F_s(\pi) = \sum_{k=1}^{N} (N - k + 1) s_{\pi_k}. \tag{A.4}$$

Decision-time waiting is order-independent, so minimizing Eq. (A.4) also minimizes mean flow time.

Proposition 1 (Optimal order under exact work). Under Assumption 1, if $W(i) = s_i$, sorting candidates by nondecreasing $W(i)$ minimizes $F_s(\pi)$.

Proof. For adjacent requests i, j, the difference between orders $i \to j$ and $j \to i$ is $s_i - s_j$. Swapping does not affect other requests; when $s_i \leq s_j$, placing i first cannot increase the objective. Eliminating all inversions proves the claim.

Proposition 2 (Bound under estimation error). Under Assumption 1, let π_W be the order induced by W and π^* the true optimal order. If $0 \leq \epsilon < 1$ and $(1 - \epsilon)s_i \leq W(i) \leq (1 + \epsilon)s_i$ for every candidate, then

$$F_s(\pi_W) \leq \frac{1 + \epsilon}{1 - \epsilon} F_s(\pi^*). \tag{A.5}$$

Proof. Let F_W denote Eq. (A.4) evaluated with W. By Proposition 1 and the error bounds,

$$F_s(\pi_W) \leq \frac{F_W(\pi_W)}{1 - \epsilon} \leq \frac{F_W(\pi^*)}{1 - \epsilon} \leq \frac{1 + \epsilon}{1 - \epsilon} F_s(\pi^*).$$

A.3.3 Age Fence Promotion Bound

CAA promotes the earliest request to the work-based head when its waiting time reaches $\rho \max\{W(r_{\text{head}}), f\}$.

Assumption 2 (Bounded scheduling opportunities). The age fence is enabled and arrival times are valid. Once a request becomes earliest, it remains in the candidate set; adjacent scheduling boundaries are at most Δ apart, each has $W(r_{\text{head}}) \leq B$, and no recovery-ready handoff overrides the age fence.

Proposition 3 (Promotion bound). Under Assumption 2, the request is promoted to the head within

$$\rho \max\{B, f\} + \Delta \tag{A.6}$$

time.

Proof. After $\rho \max\{B, f\}$ time, the trigger condition must hold. The next scheduling boundary occurs within Δ, proving Eq. (A.6).

Lemma 1 describes the recovery pipeline. Propositions 1 and 2 characterize the ordering objective under Assumption 1, with relative estimation error bounded by ϵ in Proposition 2. The applicability of the service model and error condition depends on the target model, hardware, and workload. Under Assumption 2, Proposition 3 bounds the time to promotion at the ordering layer; execution after promotion remains subject to recovery-ready handoff and native resource checks. The end-to-end results and ablations in Section 5 evaluate these mechanisms under dynamic batching, multiple execution slots, and overlap between recovery and computation.

A.3.4 Complete Admission Algorithm

Algorithm 1 summarizes CAA's admission logic at a single scheduling boundary: the underlying policy first determines the candidate set, Arena then provides the recoverable-prefix summary used to compute each request's service work, and CAA applies work-based ordering, the age fence, and recovery handoff before returning the updated order to the underlying engine.

Algorithm 1 Cache-Aware Agent Admission

Require: Candidate queue Q, Arena recoverable-prefix summary
1: $Q_{\text{base}} \leftarrow \text{BasePolicy}(Q)$
2: $A \leftarrow$ requests awaiting admission in Q_{base}
3: **for** each r in A **do**
4: Compute $P(r)$, H_r, $R(r)$, and $D(r)$
5: $W(r) \leftarrow P(r) + H_r p_H + R(r) + D(r)$
6: **end for**
7: Sort A by increasing $W(r)$, then arrival time and request identifier
8: $r_{\text{old}} \leftarrow$ earliest-arriving request in A; $r_{\text{head}} \leftarrow$ head of A
9: **if** waiting time of $r_{\text{old}} \geq \rho \max(W(r_{\text{head}}), f)$ **then**
10: Promote r_{old} to the head of A
11: **end if**
12: **if** a `recovery-ready` request exists **then**
13: Promote the first `recovery-ready` request in A to the head (overriding the order above)
14: **end if**
15: Update the candidate order in Q_{base} using the order in A
16: Return to the underlying engine for resource checks and prefill

A.4 Extended Evaluation

This section supplements two comparisons omitted from the main text. Figure 11 separates capacity allocation from page-write timing, while Figure 12 examines whether publication and recovery stage costs remain stable under external load. Both experiments use the models, traces, and timing definitions from Section 5.

A.4.1 Capacity Allocation and Eager Persistence

Figure 11 compares capacity allocation and eager page persistence in the GPT-OSS-20B, Hermes C8 setting. The experiment replays the same 50 complete sessions, comprising 910 turn requests, with FCFS, two execution slots, and a fixed 309,696-token device+host KV budget. The *Original* split assigns 147,456/162,240 tokens to device/host, and the *Matched* split assigns 293,312/16,384 tokens. Each split runs *Write Through Sel* and *Write Through*; *Arena* uses the *Matched* split. All other model, trace, and backend settings remain fixed.

Capacity reallocation improves native HiCache. Under *Write Through Sel*, *Matched* raises reuse by 15.91 percentage points over *Original* and reduces mean/P99 TTFT by 22.2%/9.7% and mean/P99 JCT by 19.9%/4.7%. Under *Write Through*, reuse rises by 18.18 percentage points and the four latency metrics decrease by 26.1%, 10.1%, 22.7%, and 6.3%, respectively. On the same *Matched* split, changing from *Write Through Sel* to *Write Through* changes reuse by 0.03 percentage points, reduces mean/P99 TTFT by 1.3%/1.0%, and reduces mean/P99 JCT by 0.7%/1.2%. The two native write policies perform similarly under matched capacity in this replay setting.

Compared with *Write Through* under *Matched* capacity, *Arena* further raises reuse by 23.89 percentage points and reduces mean/P99 TTFT by 50.1%/60.8% and mean/P99 JCT by 47.1%/66.6%. With capacity allocation and FCFS admission held fixed, this comparison measures the combined effect of Arena's lifecycle scheme: incremental publication, retries of incomplete pages, compound commit, and continuous-prefix recovery.

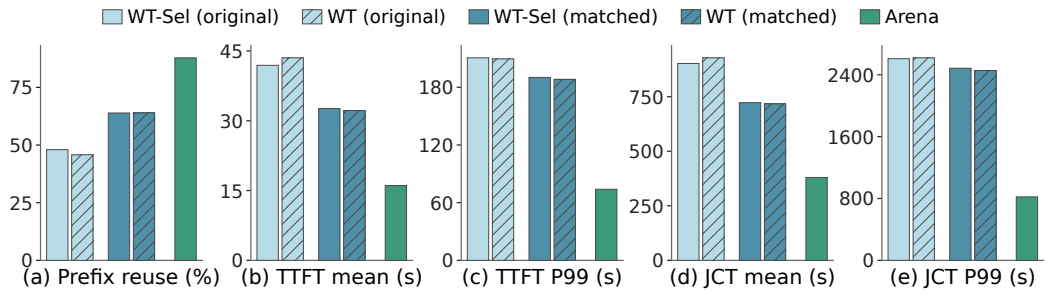

Figure 11: Capacity allocation and eager-persistence controls. Panels (a)–(e) report prefix reuse, mean TTFT, P99 TTFT, mean JCT, and P99 JCT. All configurations replay the same 50 sessions with FCFS, two execution slots, and a fixed 309,696-token Device+Host KV budget. Diagonal hatching marks eager *Write Through*, solid native bars use *Write Through Sel*, and *Arena* uses the matched split.

A.4.2 Publication and Recovery Stage Overhead under Agent Traces and Concurrency Pressure

Figure 12 uses the Hermes Reasoning trace as a fixed agent workload and compares Qwen3.5-35B-A3B and GPT-OSS-20B at C4, C6, and C8. It measures *Arena* and *UMAO* with CAA enabled to test whether external load changes the stage costs of the publication and recovery paths. D2H and H2S are the publication-side device-to-host and host-to-storage latencies per operation, while S2H and H2D are the recovery-side storage-to-host KV I/O and host-to-device CUDA-copy latencies per window. Each point reports the mean stage latency, calculated as the accumulated stage duration divided by the number of timed operations or windows.

Across C4–C8, the four stage latencies remain within similar ranges for each model, with no monotonic increase as concurrency rises. The curves for *Arena* and *UMAO* largely overlap, indicating similar per-operation publication and per-window recovery costs with CAA enabled. GPT-OSS has fewer recovery-window samples at low concurrency. Figure 7(b) complements these stage measurements with request-level Recovery E2E latency.

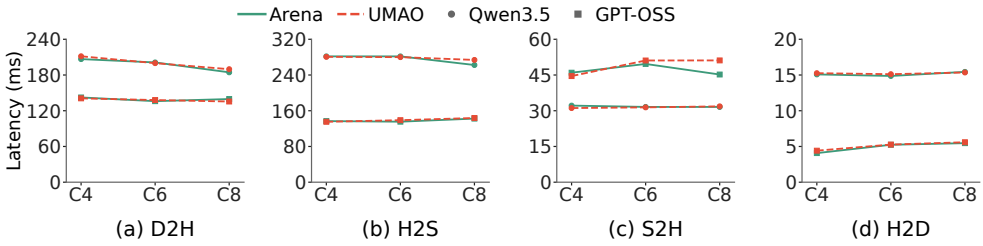

Figure 12: Publication and recovery stage latency on Hermes replays. (a) Publication Device-to-Host and (b) Host-to-Storage are measured per timed publication; (c) Recovery Storage-to-Host KV I/O and (d) Host-to-Device CUDA copy are measured per timed window. Points show the mean latency of each stage. Solid green and dashed red curves denote *Arena* and *UMAO*; circles and squares distinguish Qwen3.5 and GPT-OSS.

A.4.3 Runtime Memory Headroom under a Fixed KV Budget

We sample external memory during the Qwen3.5-35B-A3B & Hermes C8 replay with *UMAO*, starting after the service becomes ready and sampling once per second. All 50 sessions, comprising 910 requests, complete successfully. Figure 13(b) shows that whole-machine `MemAvailable` reaches a minimum of 26.91 GiB during the replay, remaining 1.91 GiB above the 25 GiB reference line. In Figure 13(d), the aligned logical working set peaks at 789,041 tokens, or 2.55× the configured 309,696-token device+staging KV budget. Under this fixed KV budget, the Qwen3.5-35B-A3B &

Hermes C8 replay therefore combines logical capacity pressure with a minimum observed whole-machine `MemAvailable` of approximately 27 GiB.

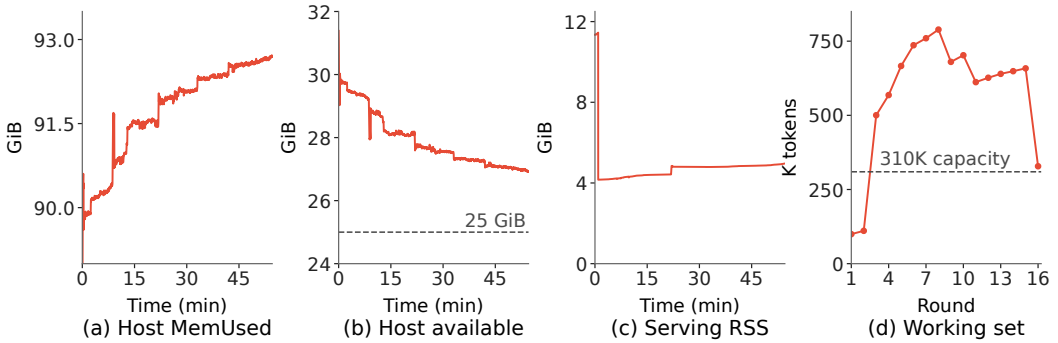

Figure 13: Runtime memory headroom for Qwen3.5-35B-A3B/Hermes C8 with *UMAO*. (a) Whole-machine MemUsed; (b) whole-machine MemAvailable, with the dashed line marking the 25 GiB reference; (c) RSS of the Serving process tree; and (d) the round-aligned logical working set relative to the configured Device+staging KV budget. Panels (a)–(c) use elapsed replay time after service readiness, while panel (d) uses aligned rounds. GPU memory measurements were unavailable.

A.5 Extended Related Work

Section 2 surveys prior work on cache hierarchies, cross-round prefix reuse, and request admission. We further compare their state units, lifecycle boundaries, and scheduling information with those of UMAO.

A.5.1 Hierarchical KV Caching and Persistence Policies

PagedAttention organizes device KV in pages, while RadixAttention indexes shared prefixes using a radix tree (Kwon et al., 2023; Zheng et al., 2024). For cross-tier caching, Strata uses *Write Through*, *Write Through Sel*, and *Write Back* to control page writes to lower tiers (Xie et al., 2026b). LMCache's `skip_l1` uses host primarily as staging for writing to and recovering from lower-tier caches (Cheng et al., 2025). Mooncake and CachedAttention retain session KV in storage for reuse by later requests (Qin et al., 2025; Gao et al., 2024).

These methods center on when a page or cache object is written, where it resides, and how it is loaded again. They treat device, host, and storage as distinct logical tiers and generally manage persistence at page or object granularity. For a multi-round agent, however, persisting an individual page does not make the entire prefix from the start of its history to that page continuously recoverable.

Arena manages the lifecycle of cross-round history under fixed shared capacity. It restricts host to bounded transfer staging, publishes newly appended history at each round boundary, and retries incomplete pages that remain accessible through the current cache path. Arena advances the `committed` boundary only after both the continuous KV pages and the endpoint state required by a hybrid model are durable. Recovery reads only up to this boundary, rather than treating individually acknowledged pages beyond a gap as a recoverable prefix.

A.5.2 Multi-round Agent State Management and Cross-round Reuse

Agent state management systems use tool-call stages, execution dependencies, or session boundaries to decide how state is retained. InferCept retains or evicts state during tool interception (Abhyankar et al., 2024). Continuum sets KV-cache lifetimes according to tool-call stages and combines them with FCFS for multi-round agent scheduling (Li et al., 2026). KVFlow manages prefix caches for multi-agent workflows (Pan et al., 2026), and Learning Agent Execution adapts KV-cache management using agent execution information (Zhang et al., 2026). Agent Memory uses an agent cache as its persistence and recovery unit (Shkolnikov, 2026).

These systems use tool calls, execution paths, or session structure to guide retention, eviction, and prefetching. Their state units are typically a tool-call stage, an execution dependency, or an agent cache. UMAO instead tracks publication of history appended at each round and uses the continuous prefix as its recovery unit. Round-aligned incremental publication, retries of incomplete pages, and compound commit together determine the historical range available to the next request.

For hybrid models, cross-round reuse also requires recurrent state and KV cache to reach the same prefix boundary (Gu & Dao, 2023; Yang et al., 2025). UMAO therefore checks the corresponding state checkpoint as well as KV-page persistence when advancing its commit boundary.

A.5.3 Prefix Indexing and Request Scheduling

PagedAttention and RadixAttention organize and index shared prefixes within device, while SGLang admits requests based on the current longest-prefix match (Kwon et al., 2023; Zheng et al., 2024). The central scheduling signal is the currently matched prefix length. By itself, this length does not distinguish the logical tier of a prefix or the work needed for a direct hit, cross-tier recovery, or remaining prefill.

FastServe organizes serving queues by input length within a distributed execution engine (Wu et al., 2024). Bidaw combines compute-side request execution state with storage-side KV loading and eviction decisions (Hu et al., 2026). PEEK uses prefix sharing and waiting-queue state for prefix-aware scheduling and KV management (Xie et al., 2026a). These systems account for input size, compute and storage costs, or queue state, but their scheduling decisions primarily operate within a serving engine or its multi-GPU execution group rather than routing requests across independent server instances.

CAA reads the recoverable-prefix summary generated by Arena at the same scheduling boundary. The summary covers direct device/host hits, the storage-recoverable prefix, remaining prefill, recovery-window count, and recovery-queue state. CAA estimates service work from these components and orders candidates with an age fence and `recovery-ready` handoff. Its admission state thus describes not only logical prefix length but also whether that prefix can be served under UMA's capacity and transfer constraints.

A.5.4 Positioning Summary

UMAO connects cache lifecycle and request admission through a shared recoverable-prefix state for multi-round agent serving on UMA. It jointly addresses bounded host staging under fixed shared capacity, round-aligned history publication, continuous-prefix recovery, and admission based on recovery and recomputation work. These mechanisms differ from approaches centered on individual page writes, agent-state retention, input length, or queue state.

Arena's compound commit defines the prefix boundary shared by cache management and recovery. CAA reads the summary at that boundary to estimate request work. The experiments separately evaluate complete UMAO, Arena's lifecycle scheme, and CAA admission to distinguish the contributions of cache lifecycle and request ordering.