RR-EVICT: FINE-GRAINED PREFIX CACHE EVICTION BEYOND LRU FOR AGENTIC LLM SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based agents execute long-horizon tasks through repeated model calls interleaved with tool execution and user interaction. As each call extends the history accumulated in previous turns, prefix caching avoids repeated prefill of the agent's entire context. However, the aggregate cache footprint grows with context length and concurrency, forcing serving systems to reclaim cached KV tensors. We identify *recency synchronization*: accesses to an agent's cached history refresh its cache nodes together, causing least recently used (LRU) eviction to concentrate on a few agents' private histories. When a fully evicted agent returns, it must recompute nearly its entire accumulated context, producing a large time-to-first-token (TTFT) outlier even if most other requests retain substantial cache reuse. We present RR-EVICT, a fine-grained prefix-cache eviction strategy that distributes reclamation across idle agent trajectories. RR-EVICT visits agents in round-robin order and evicts a tail chunk from each, preserving reusable prefixes for more agents when capacity remains for idle state. Returning agents can reuse these partial histories and recompute smaller missing suffixes. The policy requires no prediction of future arrivals or tool latency. We implement RR-EVICT in SGLang and evaluate conversational and coding-agent workloads under colocated and prefill–decode-disaggregated serving. Compared with LRU, RR-EVICT reduces P99 TTFT by up to 75.4% and P99 uncached prompt tokens by up to 65.7%.

1 INTRODUCTION

Large language model (LLM) agents (Yao et al., 2023; Shinn et al., 2023; Yang et al., 2024; Wang et al., 2025) are increasingly used to execute long-horizon tasks that span many rounds of model inference, tool invocations, and user interactions. Unlike conventional stateless requests, each new agent request typically extends the history accumulated in previous rounds. As a result, the input context grows over the agent's lifetime, potentially reaching hundreds of thousands or even millions of tokens (Packer et al., 2023; Anthropic, 2025; Curme & Daugherty, 2026).

Prefix caching (Zheng et al., 2024; Gim et al., 2024; Gao et al., 2024; Yu et al., 2025; Pan et al., 2025b) is essential for serving such workloads efficiently. After an agent completes an LLM invocation and starts executing a tool or waiting for user feedback, the serving system can retain the key-value (KV) cache associated with its context. When the agent issues its next request, the system reuses the cached KV tensors for the shared prefix and computes the uncached suffix. If the previous history remains cached, this suffix contains only the newly appended tokens. Prefix caching therefore avoids repeatedly prefilling the agent's entire history and reduces time to first token (TTFT).

However, retaining prefix caches for long-horizon agents introduces significant GPU memory pressure (Qin et al., 2025; Liu et al., 2025; Xie et al., 2026). The cache footprint of each agent grows with its context, while the aggregate footprint increases further with the number of concurrent agents. When the total demand exceeds the available capacity, existing LLM serving systems typically reclaim cached state using least recently used (LRU) eviction (Zheng et al., 2024; vLLM Team, 2025). LRU orders individual cache entries by their last-access times and is effective when different entries exhibit heterogeneous reuse patterns. This is often the case in conventional LLM serving workloads. For example, a system prompt shared by many requests is accessed frequently, whereas request-specific suffixes may be accessed only once. Their last-access times therefore provide a useful signal for distinguishing broadly reusable prefixes from less valuable suffixes. By evicting

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.

the least recently accessed entries, LRU can preserve frequently reused prefixes while incrementally reclaiming less reusable parts of the cache.

Long-horizon agent workloads change this access pattern. Each new request from an agent extends its previous context and therefore matches the agent's entire surviving cached history. Prefix matching refreshes all cache nodes along this path at approximately the same time, regardless of when those nodes were originally created or how much recomputation each represents. Moreover, different agents typically share only a short common prefix, while most of their accumulated histories remain private. Consequently, the private cache nodes of each agent form a large group with nearly identical recency. We refer to this phenomenon as *recency synchronization*.

Under recency synchronization, nominally node-level LRU can behave like coarse-grained, agent-level eviction. When the system needs to reclaim more memory than a small number of cache nodes provide, LRU repeatedly selects nodes belonging to the oldest idle agent. It therefore tends to remove most or all of that agent's cached context before moving on to another agent. Although reclamation still operates on individual nodes, cache loss becomes concentrated on entire agent histories.

Such concentrated eviction is particularly harmful to tail latency. When a fully evicted agent becomes active again, it receives little prefix-cache reuse and must recompute nearly its entire accumulated history. Because the recomputation cost grows with context length, a small subset of agents can experience severe TTFT spikes even when most other requests retain substantial reuse. Therefore, optimizing only aggregate cache reuse is insufficient for long-horizon agent serving. The system must also consider how cache loss is distributed across agents.

To address this problem, we present RR-EVICT, a fine-grained prefix-cache eviction strategy for long-horizon agent workloads. Instead of reclaiming one agent's entire context at a time, RR-EVICT distributes eviction across idle agents in a round-robin manner. Whenever memory reclamation is required, it evicts a tail chunk from one agent and then advances to the next, continuing cyclically until sufficient memory has been released. Evicting from the tail preserves a valid, contiguous cached prefix, while round-robin selection distributes the reclamation demand across multiple histories.

This design replaces concentrated cache loss with gradual degradation. When capacity remains for idle state, RR-EVICT retains partial histories for more agents under the same reclamation demand. Returning agents can reuse their remaining prefixes and recompute only the missing suffixes instead of prefilling their entire histories. By balancing cache loss across agents, RR-EVICT reduces both the frequency and magnitude of cold-prefill events, thereby improving tail TTFT without requiring knowledge of future agent execution.

We implement RR-EVICT in SGLang (Zheng et al., 2024) and evaluate Qwen3 (Yang et al., 2025) on NVIDIA H100 GPUs using conversational and coding-agent workloads. Across colocated and prefill–decode (PD) disaggregated configurations, RR-EVICT reduces P99 TTFT by up to 75.4% and P99 uncached prompt tokens by up to 65.7% compared with LRU. Comparisons with completion-aware LRU isolate the additional benefit of distributing eviction across live histories. Request-level distributions and a trace-derived memory-demand timeline further connect these gains to the policy's operating regime.

In summary, this paper makes the following contributions:

- We identify *recency synchronization* in long-horizon agent workloads and explain how it makes node-level LRU concentrate cache loss on a few trajectories, producing large tail-TTFT outliers.
- We propose RR-EVICT, a fine-grained eviction strategy that reclaims tail chunks in round-robin order. We characterize when preserving partial prefixes for more agents can reduce tail recomputation.
- We implement RR-EVICT in SGLang and evaluate conversational and coding agent benchmarks, showing improvements over both LRU and completion-aware LRU across the tested configurations.

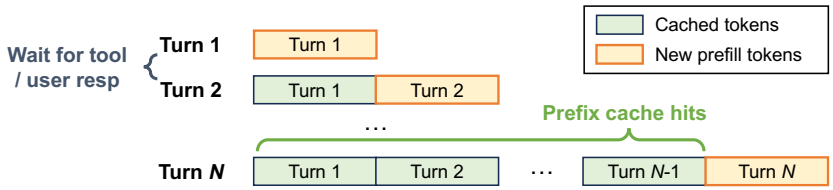


Figure 1: Prefix caching across agent turns. The KV tensors of the accumulated history are reused, and only newly appended tokens require prefill computation.

2 Background

Multi-turn agent workloads. LLM agents solve long-horizon tasks through repeated interactions with users and external environments (Yao et al., 2023; Shinn et al., 2023; Yang et al., 2024; Wang et al., 2025; Hu et al., 2026b). Each LLM invocation, which we call an *agent turn*, reasons over the current context and produces a response or tool call. As Figure 1 shows, a turn may be followed by tool execution or a wait for user feedback, during which the agent issues no LLM request. The returned observation is appended to the next request together with the preceding history. Agent workloads are therefore both stateful and intermittent: requests from the same agent share an evolving context but are separated by variable external delays (Anthropic, 2024). As the agent proceeds, its context grows while retaining most tokens from the previous turn.

Prefix caching for multi-turn agents. LLM inference consists of a prefill phase followed by autoregressive decoding (Yu et al., 2022). During prefill, the model processes the input tokens and materializes their key-value (KV) tensors in every attention layer. Prefix caching retains these tensors across requests and reuses them when a later request begins with the same token sequence (Zheng et al., 2024; Gim et al., 2024; Gao et al., 2024; Yu et al., 2025). As Figure 1 illustrates, Turn 1 computes its entire input, whereas Turn N can reuse the KV tensors for Turns 1 through $N-1$ and prefill only the newly appended tokens. A longer prefix-cache hit avoids more recomputation and reduces time-to-first-token (TTFT).

Modern serving systems manage reusable KV tensors in a shared cache pool (Kwon et al., 2023; Zheng et al., 2024; vLLM Team, 2025). Systems such as SGLang organize cached sequences as a radix tree, allowing requests to reuse the KV blocks along their longest matching prefix. In agent workloads, short prefixes such as system instructions may be shared across agents, while the longer interaction histories are typically private to each agent. As these histories and the number of concurrent agents grow, their aggregate demand can exceed the pool's GPU memory capacity. The eviction policy then determines how much history remains reusable when each agent resumes and, consequently, its next-turn prefill latency.

3 RR-Evict: Fine-Grained Prefix-Cache Eviction

In this section, we first examine why conventional LRU eviction is inefficient for long-horizon agent workloads and then present RR-Evict, which distributes cache reclamation across agents at fine granularity. Finally, we characterize the operating regime in which round-robin eviction improves tail latency. Figure 2 contrasts the two policies.

3.1 Inefficiency of LRU Eviction

Serving systems index reusable KV tensors at different granularities. SGLang uses radix-tree nodes representing variable-length context spans, whereas vLLM manages fixed-size KV blocks (Zheng et al., 2024; vLLM Team, 2025). We use *cache node* to denote an entry in the prefix index. The following description focuses on SGLang's tree representation.

As shown in Figure 2(a), nodes referenced by executing requests are active and cannot be reclaimed, whereas the cached trajectories of agents waiting for tool outputs or user responses are idle and eligible for eviction. To preserve a valid cache tree, eviction begins at idle leaf nodes and progressively

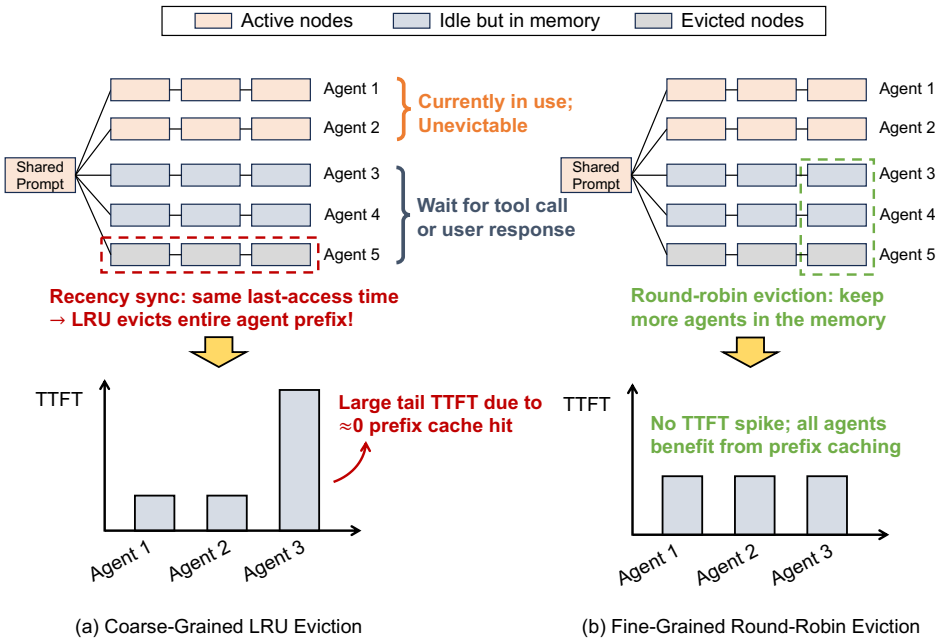


Figure 2: Schematic comparison of LRU and RR-EVICT. (a) Recency synchronization concentrates successive LRU evictions on one idle agent, causing a large prefill when it returns. (b) Round-robin tail eviction preserves partial prefixes for more agents. The TTFT bars illustrate the mechanism rather than measured latencies.

exposes their parents as new leaves. Conventional LRU inserts eligible leaves into a heap ordered by their last-access time. This node-level policy is effective when nodes have sufficiently diverse access times because it can incrementally reclaim the least recently used data.

Agent workloads reduce this diversity. When an agent starts a turn, prefix matching accesses every node along its surviving cached prefix, updating those nodes at approximately the same time. After the turn completes, its private path becomes eligible for eviction with identical or nearly identical recency. Because agents typically share only a short system prompt while keeping most of their histories private, each agent contributes a large group of private nodes whose recency is synchronized. This is the *recency synchronization* described in Section 1.

During reclamation, the synchronized nodes of the oldest idle agent remain at or near the top of the LRU heap as their descendants are removed and their parents become leaves. Successive evictions therefore concentrate on that agent until most or all of its private prefix is removed, effectively turning the agent into the unit of eviction. When the agent resumes, it can reuse little beyond the short shared prompt and must prefill nearly its entire history. As illustrated at the bottom of Figure 2(a), this concentrated cache loss produces a large TTFT spike for the evicted agent even when the aggregate cache hit rate remains high.

3.2 RR-EVICT: ROUND-ROBIN TAIL EVICTION

RR-EVICT retains the existing cache-tree traversal and active-node filtering but changes how eviction candidates are ordered. When memory must be reclaimed, the system scans the leaf nodes and skips those currently used by active requests. Instead of inserting the remaining leaves into a global heap ordered by LRU priority, RR-EVICT groups them by their agent trajectories and visits these groups in round-robin order. It reclaims one *tail chunk* from an agent before moving to the next. A tail chunk is the backend-specific amount reclaimed in one step and may be an entire leaf node or only a suffix within that node. A partially reclaimed leaf remains the trajectory's next candidate. Removing it completely exposes its parent, which can be selected only if it is also eligible for eviction. The process continues until the requested amount of memory has been released.

Round-robin selection converts concentrated cache loss into gradual degradation across agents. Under the same reclamation demand, several agents can retain partial histories rather than one agent losing its complete history. As illustrated at the bottom of Figure 2(b), returning agents can reuse their surviving prefixes and prefill only the missing suffixes. This reduces concentrated cold-prefill events when capacity remains for idle state. RR-EVICT requires no prediction of future arrivals or tool latency. It uses only the current active/idle state and cache-tree metadata.

Algorithm 1 summarizes the round-robin policy. At the start of reclamation, eligible idle trajectories enter a single queue. After evicting one tail chunk, the algorithm moves the trajectory to the back of the queue if it still has an evictable suffix. Eligibility is rechecked before every eviction because an agent may resume concurrently. The backend-specific SELECTTAILCHUNK operation chooses either a complete leaf or a suffix of one.

Algorithm 1 Round-Robin Tail Eviction

Require: Idle agent trajectories $\mathcal{A}$, memory to reclaim M
$Q \leftarrow$ queue of trajectories in $\mathcal{A}$ with evictable suffixes
Freed memory $M_{\text{freed}} \leftarrow 0$
while $M_{\text{freed}} < M$ and $Q \neq \emptyset$ **do**
    Agent trajectory $\tau \leftarrow Q$.POPFRONT
    if τ is active or has no evictable suffix **then**
        continue
    end if
    $c \leftarrow$ SELECTTAILCHUNK$(\tau, M - M_{\text{freed}})$
    $M_{\text{freed}} \leftarrow M_{\text{freed}} +$ EVICT(c)
    if τ still has an evictable suffix **then**
        Q.PUSHBACK(τ)
    end if
end while

Handling stale agent trajectories. Direct round-robin has one limitation: because it rotates uniformly across all idle trajectories, a completed trajectory may retain a large cached prefix for many eviction rounds, wasting memory that could benefit live agents. Our implementation supports two ways to handle this stale state. First, the application can notify the serving backend when an agent completes, allowing the backend to immediately release the trajectory's unshared cache. Second, when such a lifecycle signal is unavailable, RR-EVICT combines LRU with round-robin through *recency tiering*. It partitions idle trajectories by their last-access times, processes tiers from least to most recent, and applies round-robin eviction among trajectories within the same tier. The cross-tier LRU ordering prioritizes trajectories that are more likely to be stale, while within-tier round-robin distributes cache loss among similarly recent agents. The evaluation uses completion signals for both RR-EVICT and the completion-aware LRU baseline. It does not evaluate recency tiering.

3.3 ANALYTICAL COMPARISON OF LRU AND RR-EVICT

We next formalize the benefit of fine-grained eviction under a simplified homogeneous workload with no cross-agent prefix sharing. The analysis first identifies when memory pressure leaves room to retain idle prefixes and then compares how LRU and ideal round-robin eviction distribute prefix hits and recomputation across returning agents.

Workload and memory assumptions. Consider n live trajectories with equal-sized private prefixes and no cross-agent sharing. We normalize each complete prefix to one unit and hold its size fixed during reclamation. Each agent spends a fraction s of its lifetime idle and a fraction $a = 1 - s$ active, with $0 < s < 1$. If agent phases are sufficiently desynchronized, approximately $n_a = na$ trajectories are active and $n_i = ns$ are idle at a given time. Let m be the cache capacity in prefix-equivalents. Eviction is unnecessary when $m \geq n$. When $m \leq na$, active demand consumes all capacity or exceeds it, leaving no space for idle prefixes and creating a risk of thrashing (Chen et al., 2026; Kang et al., 2026). Distributing eviction across idle agents is therefore useful only when

$$na < m < n. \tag{1}$$

Here, na is a fluid approximation. A deployment must also account for bursts in the instantaneous active set, which this mean-based model does not capture.

Within this operating regime, the remaining capacity determines how much of each idle prefix can be preserved. After pinning the na active trajectories, the system can retain $m - na$ prefix-equivalents for the ns idle trajectories. Ignoring chunk-size effects, ideal round-robin eviction retains the same fraction of every idle prefix:

$$h = \frac{m - na}{ns}, \qquad q = 1 - h = \frac{n - m}{ns}, \tag{2}$$

where h is the retained fraction and q is the fraction of the old prefix that must be recomputed on return. For comparison, we model LRU's concentrated eviction as retaining $m - na$ idle prefixes completely and removing the remaining $n - m$, ignoring a possible partially retained boundary prefix. For a uniformly selected returning idle agent, the normalized old-prefix recomputation cost X_π under policy $\pi \in \{\mathrm{LRU}, \mathrm{RR}\}$ is then

$$X_{\mathrm{LRU}} = \begin{cases} 1, & \text{with probability } q, \\ 0, & \text{with probability } 1 - q, \end{cases} \qquad X_{\mathrm{RR}} = q. \tag{3}$$

Both policies have the same mean recomputation cost, $\mathbb{E}[X_\pi] = q$, but distribute it differently. LRU places the full cost on a fraction q of returning agents, whereas ideal round-robin spreads it equally. If prefill latency is approximately proportional to uncached prefix length, this reduces the worst-case old-prefix recomputation component of TTFT from 1 to q. At percentile p, LRU has a full miss when $q > 1 - p$, while round-robin incurs a q-fraction miss. When $q \leq 1 - p$, LRU's p-quantile is instead zero, so round-robin does not improve every quantile. This comparison excludes new prompt tokens, queueing, and other components of measured TTFT.

The useful regime is consequently bounded on both sides. As $m \to na$, $q \to 1$ and no policy can retain meaningful idle state. As $m \to n$, full-prefix misses under LRU become too rare to affect a fixed tail percentile. RR-EVICT is most useful between these extremes, where active contexts fit but concentrated misses remain frequent enough to dominate the tail.

4 EVALUATION

We implement RR-EVICT in SGLang (Zheng et al., 2024) and evaluate it on conversational and coding-agent benchmarks.

Experimental setup. We run our experiments on NVIDIA H100 GPUs connected by NVLink. For conversational agents, we serve Qwen3-Coder-30B-A3B-Instruct (Yang et al., 2025) in BF16 with tensor parallelism two (TP2) and run 222 banking task trajectories from the local τ^2-bench suite (Shi et al., 2026). For coding agents, we serve Qwen3-8B on SWE-bench (Jimenez et al., 2024) under prefill–decode (PD) disaggregation. The 1P3D configuration uses one TP1 prefill worker and three TP1 decode workers, with prefix caching on the prefill worker.

We compare RR-EVICT against LRU and a completion-aware LRU variant. LRU is the default prefix-cache eviction policy in serving engines such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024). The variant, which we denote as *LRU + Completion Signals*, allows the frontend to notify the serving backend when an agent completes so that the backend can immediately reclaim that agent's unshared cache. RR-EVICT uses the same completion signals together with 1,024-token round-robin eviction chunks for live agents.

We vary concurrency to control the number of admitted agents (Kang et al., 2026; Chen et al., 2026). To ensure a fair comparison, we first execute each benchmark once to record its input contexts, output lengths, and inter-turn delays. We then replay the resulting trace under every policy, preserving the recorded input contexts, output lengths, request order, and inter-turn gaps and thereby reproducing both context growth and the idle periods between agent turns. The replay initially fills the available agent slots. Once the target concurrency is reached, it admits a new agent only after a resident agent exits. Waiting agents remain admitted while executing tools or awaiting user responses, so their cached histories continue to compete for memory.

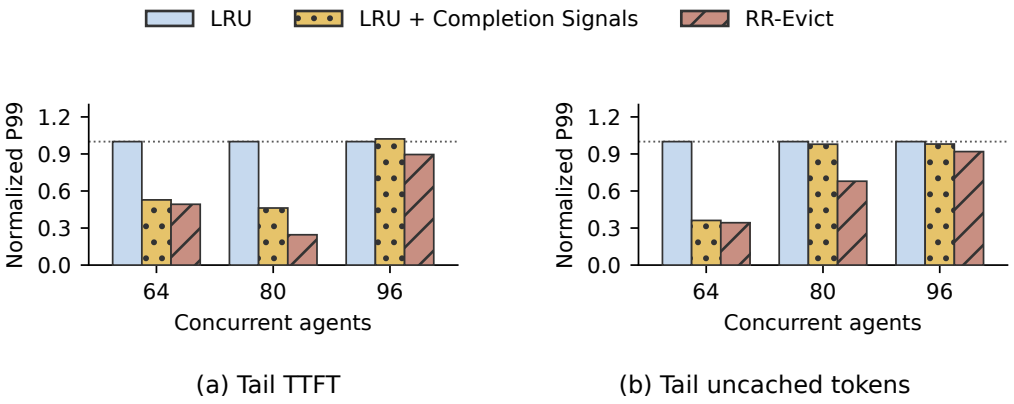


Figure 3: Tail TTFT and uncached prompt tokens on τ^2-bench under different concurrency levels. Results for LRU, LRU + Completion Signals, and RR-EVICT are normalized to LRU at the same concurrency. Lower is better.

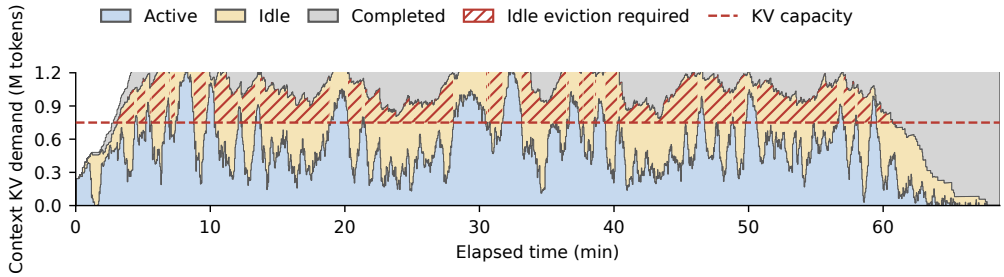
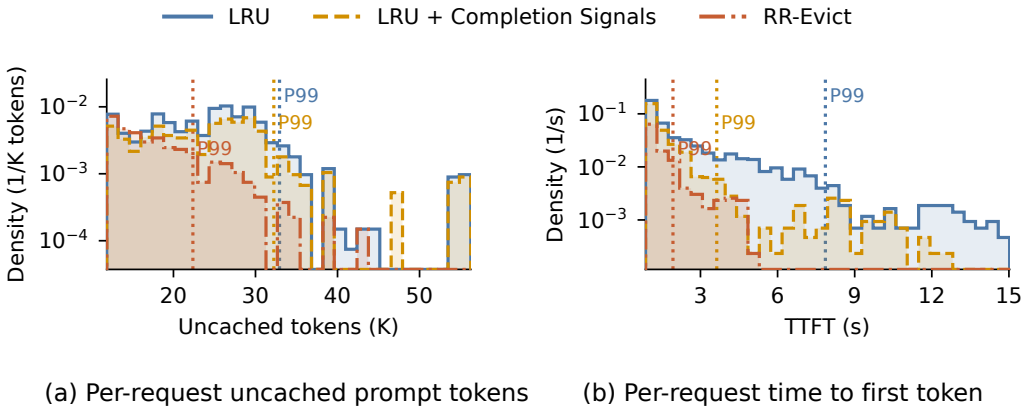


Figure 4: Token-derived logical context-KV demand on τ^2-bench at concurrency 80. The stack separates active requests, live idle trajectories, and completed trajectories. The dashed line marks KV-cache capacity. Red hatching marks intervals in which active KV fits but active plus idle KV does not, so a portion of live idle state must be evicted.

We report P99 TTFT and P99 uncached prompt tokens as our two primary metrics. TTFT measures the time from request arrival until the first output token is ready and captures user-visible tail latency. For each request, uncached tokens equal the prompt length minus the cached prefix length and quantify the input requiring prefill after prefix lookup, including new tokens and uncached history. Together, the metrics capture both the outcome of cache eviction and the recomputation that RR-EVICT is designed to reduce.

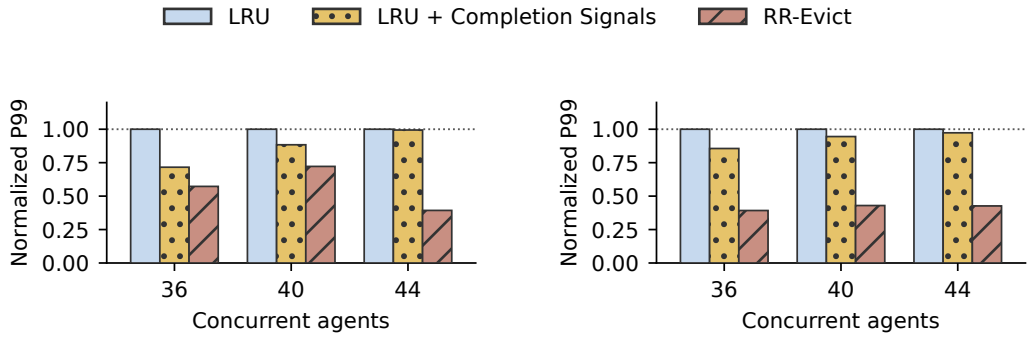
Results on conversational benchmarks. Conversational agents frequently pause for user feedback, leaving many admitted trajectories idle and creating the sparse activity analyzed in Section 3.3. Figure 3 compares LRU, LRU + Completion Signals, and RR-EVICT on this workload. Across all evaluated concurrency levels, RR-EVICT consistently outperforms both baselines. Relative to LRU, it reduces P99 TTFT and P99 uncached tokens by up to 75.4% and 65.7%, respectively. Relative to LRU + Completion Signals, the corresponding reductions are up to 46.9% and 30.6%. At lower concurrency, the incremental benefit over completion signals is smaller, consistent with completed-state reclamation relieving much of the memory pressure. At the highest concurrency, the narrower benefit is consistent with active demand leaving less capacity for idle prefixes.

Trace-level memory pressure. Figure 4 further explains the τ^2-bench results from the concurrency-80 run. We derive each trajectory's logical KV demand from its prompt tokens and classify it as active while a request is in flight, idle between turns, and completed after its final matched turn. Completion signals allow the system to release completed trajectories directly. The trace nevertheless contains sustained intervals in which active demand remains below capacity but active plus live idle



(a) Per-request uncached prompt tokens (b) Per-request time to first token

Figure 5: Upper-tail probability densities for per-request uncached prompt tokens and time to first token on τ^2-bench at concurrency 80. The panels compare LRU, LRU + Completion Signals, and RR-EVICT using shared bins and full-sample normalization. Dotted lines mark P99.



(a) Tail TTFT (b) Tail uncached tokens

Figure 6: Tail TTFT and uncached prompt tokens for Qwen3-8B on SWE-bench under 1P3D disaggregation at different concurrency levels. Results for LRU, LRU + Completion Signals, and RR-EVICT are normalized to LRU at the same concurrency. TTFT includes prefill queueing. Lower is better.

demand exceeds it, as indicated by the hatched region. During these intervals, active requests fit, yet the system must evict part of the live idle state. This is precisely the regime in Equation 1: eviction is necessary, but enough capacity remains for RR-EVICT to spread the loss across idle trajectories instead of fully evicting a few of them.

Uncached token and TTFT distributions. Figure 5 shows the tail uncached token and TTFT distributions of τ^2-bench requests at concurrency 80. RR-EVICT substantially shortens the extreme uncached-token tail, which in turn shifts the TTFT tail to lower values. The TTFT improvement is more pronounced than the reduction in uncached tokens. This larger shift is consistent with RR-EVICT avoiding exceptionally long prefills that would otherwise introduce head-of-line blocking and additional queueing delay for subsequent requests. Some long cache-miss events remain even under RR-EVICT. The active-demand bursts in Figure 4 suggest a possible explanation: less capacity remains for idle histories during these intervals, so returning agents may still incur large prefills.

Results on PD-disaggregated coding agents. Coding agents typically spend less time in tool calls (Li et al., 2025), yielding less inter-turn sparsity for RR-EVICT to exploit directly. PD disaggregation creates an additional idle interval for prefix caching: reusable prefix KV remains on the

prefill worker while decoding proceeds elsewhere. Figure 6 reports results on SWE-bench, where RR-EVICT consistently outperforms both baselines across all evaluated concurrency levels. Relative to LRU, it reduces P99 TTFT and P99 uncached tokens by up to 60.7% and 60.8%, respectively. Relative to LRU + Completion Signals, the reductions reach 60.5% and 56.2%. Completion signals alone have little effect at the highest concurrency, while distributing eviction across live agents reduces both cache misses and prefill queueing. As a result, the TTFT benefit becomes especially pronounced under higher load.

5 RELATED WORK

LLM serving systems. LLM serving systems optimize scheduling, operator execution, and KV-cache management to reduce latency and improve hardware utilization. At the scheduling layer, Orca (Yu et al., 2022) enables continuous batching through iteration-level scheduling. Chunked-prefill techniques (Agrawal et al., 2024; Holmes et al., 2024) split long prefills into chunks and mix the resulting prefill chunks with decoding, reducing interference between the two phases. Prefill-decode disaggregation (Zhong et al., 2024; Patel et al., 2024) places the two phases on separate instances to further eliminate such interference. At the operator-execution layer, prior work optimizes GPU kernels and compilers (Dao et al., 2022; Dong et al., 2024; Pan et al., 2025a; Kamath et al., 2025; Sanovar et al., 2025; Wu et al., 2025; Cheng et al., 2025; Zheng et al., 2023; Xia et al., 2023; Ye et al., 2026), with FlashInfer (Ye et al., 2025) as a representative serving-oriented kernel library. For KV caches, vLLM (Kwon et al., 2023) reduces fragmentation with PagedAttention, and SGLang (Zheng et al., 2024) captures prefix sharing with a radix tree. CachedAttention (Gao et al., 2024) and Pensieve (Yu et al., 2025) preserve KV state across conversation turns, while Mooncake (Qin et al., 2025), LMCache (Liu et al., 2025), and SGLang HiCache (Xie et al., 2026) expand capacity through pooled or hierarchical storage. These systems target general request streams rather than long-running agents, whose private histories repeatedly grow across tool or user interactions.

Agent serving systems. As agent applications become increasingly important, recent work has tailored LLM serving to their execution structure. Parrot (Lin et al., 2024), Ayo (Tan et al., 2025), and HedraRAG (Hu et al., 2025) model workflows as dependency graphs to enable compile-time optimization. Agentix (Luo et al., 2026) proposes program-level scheduling to reduce head-of-line blocking. SmoothAgent (Pan et al., 2026b) and FlashAgents (Fang et al., 2026) exploit agent structure to prepare or overlap prefills. ThunderAgent (Kang et al., 2026) and CONCUR (Chen et al., 2026) prevent overload-induced memory thrashing through admission and concurrency control. FlashEvolve (Hu et al., 2026a) uses asynchronous stage orchestration to accelerate agent self-evolution. These systems do not specifically optimize KV-cache management for agentic workloads, which is the focus of this paper. InferCept (Abhyankar et al., 2024) and Continuum (Li et al., 2025) use tool-duration estimates to guide cache retention, while KVFlow (Pan et al., 2025b) and ScaleSim (Pan et al., 2026a) rely on application information like workflow structures. Such signals may be unavailable or inaccurate in practice. In contrast, RR-EVICT distributes cache loss across idle agents without workflow graphs or tool-execution-time predictors, preserving partial prefixes for more agents.

6 CONCLUSION AND DISCUSSION

Long-horizon agents expose a key limitation of LRU: synchronized accesses concentrate eviction on a few agents, causing large prefix recomputations and tail-TTFT spikes. RR-EVICT distributes round-robin tail eviction across idle agents, preserving partial prefixes without predicting future arrivals. Our analysis identifies its useful regime, where active contexts fit but all live histories do not. Across the tested colocated and PD-disaggregated agent workloads, our SGLang implementation reduces P99 TTFT by up to 75.4% and P99 uncached prompt tokens by up to 65.7% relative to LRU.

Our evaluation is limited to relatively short idle intervals and prefix caches resident in GPU HBM. Future work will evaluate RR-EVICT at larger scale, where agents may wait much longer for user feedback and their idle state may need to spill to CPU memory or SSD, while accounting for KV transfer and reload costs.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, document organization, code implementation, and the development of plotting scripts. The authors independently designed the method and experiments, produced the measurements, and verified all claims. The authors take full responsibility for the final manuscript.

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.

Anthropic. Building effective agents. `https://www.anthropic.com/engineering/building-effective-agents`, 2024.

Anthropic. Effective context engineering for ai agents. `https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents`, 2025.

Qiaoling Chen, Zhisheng Ye, Tian Tang, Peng Sun, Boyu Tian, Guoteng Wang, Shenggui Li, Yonggang Wen, Zhenhua Han, and Tianwei Zhang. Concur: High-throughput agentic batch inference of llm via congestion-based concurrency control. *arXiv preprint arXiv:2601.22705*, 2026.

Xinhao Cheng, Zhihao Zhang, Yu Zhou, Jianan Ji, Jinchen Jiang, Zepeng Zhao, Ziruo Xiao, Zihao Ye, Yingyi Huang, Ruihang Lai, Hongyi Jin, Bohan Hou, Mengdi Wu, Yixin Dong, Anthony Yip, Zihao Ye, Songting Wang, Wenqin Yang, Xupeng Miao, Tianqi Chen, and Zhihao Jia. Mirage persistent kernel: A compiler and runtime for mega-kernelizing tensor programs. *arXiv preprint arXiv:2512.22219*, 2025.

Chester Curme and Mason Daugherty. Context management for deep agents. `https://blog.langchain.com/context-management-for-deepagents`, 2026.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2(3):4, 2024.

Taosong Fang, Zhen Zheng, Zhengzhao Ma, Yaojie Lu, Hongyu Lin, Xianpei Han, and Le Sun. Flashagents: Accelerating multi-agent llm systems via streaming prefill overlap. *Proceedings of Machine Learning and Systems*, 2026.

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 (USENIX ATC 24)*, pp. 111–126, 2024.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.

Zhengding Hu, Vibha Murthy, Zaifeng Pan, Wanlu Li, Xiaoyi Fang, Yufei Ding, and Yuke Wang. Hedrarag: Co-optimizing generation and retrieval for heterogeneous rag workflows. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 623–638, 2025.

Zhengding Hu, Mingge Lu, Zhen Wang, Jixuan Ruan, Chang Chen, Zaifeng Pan, Yue Guan, Ruiyi Wang, Zhongkai Yu, Chao Zhang, and Yufei Ding. Flashevolve: Accelerating agent self-evolution with asynchronous stage orchestration. *arXiv preprint arXiv:2605.08520*, 2026a.

Zhengding Hu, Kuntal Talit, Zhen Wang, Haseeb Ahmad, Yichen Lin, Prabhleen Kaur, Christopher Lane, Elizabeth A. Peterson, Zhiting Hu, Elizabeth A. Nowadnick, and Yufei Ding. Tritondft: Automating dft with a multi-agent framework. *arXiv preprint arXiv:2603.03372*, 2026b.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Aditya K Kamath, Ramya Prabhu, Jayashree Mohan, Simon Peter, Ramachandran Ramjee, and Ashish Panwar. Pod-attention: Unlocking full prefill-decode overlap for faster llm inference. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 897–912, 2025.

Hao Kang, Ziyang Li, Xinyu Yang, Weili Xu, Yinfang Chen, Junxiong Wang, Beidi Chen, Tushar Krishna, Chenfeng Xu, and Simran Arora. Thunderagent: A simple, fast and program-aware agentic inference system. *arXiv preprint arXiv:2602.13692*, 2026.

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, Alvin Cheung, Joseph E. Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of {LLM-based} applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945, 2024.

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.

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

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Zaifeng Pan, Yitong Ding, Yue Guan, Zheng Wang, Zhongkai Yu, Xulong Tang, Yida Wang, and Yufei Ding. Fasttree: Optimizing attention kernel and runtime for tree-structured llm inference. In *Proceedings of Machine Learning and Systems*, 2025a.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows. *arXiv preprint arXiv:2507.07400*, 2025b.

Zaifeng Pan, Yipeng Shen, Zhengding Hu, Zhuang Wang, Aninda Manocha, Zheng Wang, Zhongkai Yu, Yue Guan, and Yufei Ding. Scalesim: Serving large-scale multi-agent simulation with invocation distance-based memory management. *arXiv preprint arXiv:2601.21473*, 2026a.

Zaifeng Pan, Qianxu Wang, Zhengding Hu, Chang Chen, Yue Guan, Yanbo Zhou, Steven Swanson, and Yufei Ding. Smoothagent: Efficient long-horizon LLM-based agent serving with lookahead context engineering. *arXiv preprint arXiv:2607.00151*, 2026b.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.

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, 2025.

Rya Sanovar, Srikant Bharadwaj, Renee St Amant, Victor Rühle, and Saravan Rajmohan. Leanattention: Hardware-aware scalable attention mechanism for the decode-phase of transformers. *Proceedings of Machine Learning and Systems*, 7, 2025.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barres. τ-knowledge: Evaluating conversational agents over unstructured knowledge. *arXiv preprint arXiv:2603.04370*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Xin Tan, Yimin Jiang, Yitao Yang, and Hong Xu. Towards end-to-end optimization of llm-based applications with ayo. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 1302–1316, 2025.

vLLM Team. Automatic Prefix Caching. `https://docs.vllm.ai/en/latest/features/automatic_prefix_caching.html`, 2025.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents. In *ICLR*, 2025.

Mengdi Wu, Xinhao Cheng, Shengyu Liu, Chunan Shi, Jianan Ji, Man Kit Ao, Praveen Velliengiri, Xupeng Miao, Oded Padon, and Zhihao Jia. Mirage: A {Multi-Level} superoptimizer for tensor programs. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*, pp. 21–38, 2025.

Haojun Xia, Zhen Zheng, Yuchao Li, Donglin Zhuang, Zhongzhu Zhou, Xiafei Qiu, Yong Li, Wei Lin, and Shuaiwen Leon Song. Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity. *Proceedings of the VLDB Endowment*, 17(2):211–224, 2023.

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, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos E 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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. In *Eighth Conference on Machine Learning and Systems*, 2025.

Zihao Ye, Yingyi Huang, Hongyi Jin, Bohan Hou, Junru Shao, Zhongming Yu, Jinqi Chen, Meghan Cowan, Shiyi Cao, Shanli Xing, Hanfeng Chen, Vinod Grover, Tianqi Chen, and Luis Ceze. Cake: Compiler-agent co-design for frontier kernel evolution. *arXiv preprint arXiv:2608.12629*, 2026.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX symposium on operating systems design and implementation (OSDI 22)*, pp. 521–538, 2022.

Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 144–158, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. *Advances in Neural Information Processing Systems*, 37:62557–62583, 2024.

Zhen Zheng, Zaifeng Pan, Dalin Wang, Kai Zhu, Wenyi Zhao, Tianyou Guo, Xiafei Qiu, Minmin Sun, Junjie Bai, Feng Zhang, Xiaoyong Du, Jidong Zhai, and Wei Lin. Bladedisc: Optimizing dynamic shape machine learning workloads via compiler approach. *Proceedings of the ACM on Management of Data*, 1(3):1–29, 2023.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 193–210, 2024.