SEM: Accelerating Tool-Augmented Language Agents via Semantic Environment Memory

Anonymous ACL submission

Abstract

Large language model (LLM) agents have demonstrated remarkable capabilities in solving complex tasks by interacting with external tools and environments. However, their practical deployment is severely hindered by high end-to-end latency. We observe that tool-augmented LLMs are bottlenecked not only by generation delays but also by repeated, blocking waits for external tool use (*e.g.*, search). Unlike existing speculative execution methods that rely on brittle exact-action matching, we propose ***Semantic Environment Memory*** (SEM), a latency-aware layer that indexes environment interactions by semantic intent. Without altering the agent's reasoning policy, SEM masks tool latency through Intent-Keyed Semantic Reuse and a Speculative Future Launcher: the former retrieves intent-compatible completed observations or pending futures, while the latter proactively launches likely future environment calls. This paradigm reframes agent acceleration from predicting exact action strings to proactively reusing semantic environment information. Extensive evaluations on multi-hop reasoning (*e.g.*, HotpotQA, 2WikiMultiHopQA) and retail tool-use benchmarks (τ^2-bench and its read-heavy subsets) show that SEM significantly reduces end-to-end latency while preserving the agent's original execution trajectory and task accuracy.

1 Introduction

The rapid advancement of large language models (LLMs) has empowered language agents to interact with external environments and tools, transforming them from passive text generators into active systems for complex task solving (Yao et al., 2023b; Schick et al., 2023; Karpas et al., 2022; Patil et al., 2023; Qin et al., 2023). These agents have shown promise in web search and browsing (Nakano et al., 2021; Zhou et al., 2023; Deng et al., 2023), software engineering (Jimenez et al., 2024; Yang et al., 2024), and transactional tool-use (Yao et al., 2024). However, their practical deployment remains hindered by high end-to-end latency, especially in long-horizon tasks that necessitate iterative reasoning, tool execution, and environmental feedback.

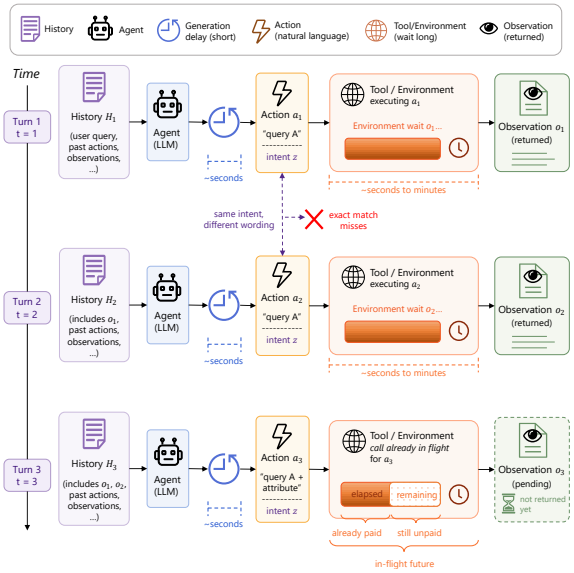

Figure 1: Motivation: agent execution repeatedly exposes short action-generation delays and longer environment-observation waits; semantically equivalent actions and in-flight calls create reuse opportunities.

Recent efforts to accelerate agent execution have explored multiple avenues. Token-level speculative decoding reduces LLM generation latency with draft tokens verified by a target model (Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2023). Other systems optimize tool execution itself: LLMCompiler parallelizes independent function calls (Kim et al., 2024), while speculative-action methods predict likely future actions and pre-execute their corresponding environment calls (Ye et al., 2025). These methods reveal that agent latency is also shaped by the structure of the agent-environment loop.

Despite these advances, existing acceleration

methods leave an important gap. As illustrated in Fig. 1, agent execution suffers from two coupled delays: action generation and environment observation. Our profiling shows that environment waits are often repeated around the same underlying information needs. This makes observation acquisition a reuse problem, yet existing action-level speculation assumes reuse requires exact matching. This leaves two challenges unresolved. First, because agent actions are natural-language expressions, exact matching misses reusable information when agents paraphrase or reformulate earlier queries. Second, agent latency stems not only from repeated completed calls but also from in-flight calls. A useful acceleration layer should therefore recover both completed observations that satisfy the same intent and pending futures that can hide the remaining environment wait.

To this end, we propose *Semantic Environment Memory* (SEM), a latency-aware memory layer situated between the actor agent and the environment. Specifically, an *Intent-Keyed Semantic Reuse* module is first introduced to map actor actions to information intents, retrieving reusable environment work. It handles both completed observations (returned immediately if intent-matched) and pending futures (waited on for remaining latency). To preemptively supply these futures, SEM utilizes a *Speculative Future Launcher* to predict information needs, pre-execute environment calls, and cache the pending futures. Together, these form a unified acquisition pipeline: intent-keyed reuse transcends exact string matching, pending-future reuse hides incurred latency, and speculative launching prepares future data. Crucially, SEM preserves the actor's reasoning policy, leaving action selection to the actor while optimizing observation retrieval. Experiments conducted on multiple benchmarks (*i.e.*, HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), and τ^2-bench (Yao et al., 2024)) demonstrate that SEM can reduce end-to-end latency while preserving the task quality.

Our contributions are as follows:

- We propose *Semantic Environment Memory* (SEM), a latency-aware layer between the actor and the environment that preserves the actor's reasoning policy while changing how observations are acquired.
- We introduce *Intent-Keyed Semantic Reuse*, which indexes environment interactions by information intent and reuses both completed observations and pending futures when they satisfy the actor's current information need.
- We introduce a *Speculative Future Launcher*, which proactively prepares future environment calls and stores them as pending futures for later reuse.
- We demonstrate that SEM reduces end-to-end latency on multi-hop search and retail tool-use benchmarks, and that it can be further combined with token-level speculation for nested acceleration.

2 Related Work

LLM Agents. LLM agents extend language models with external actions, enabling web browsing, API use, software engineering, and transactional tool-use (Yao et al., 2023b; Nakano et al., 2021; Schick et al., 2023; Patil et al., 2023; Qin et al., 2023; Yang et al., 2024; Yao et al., 2024). A parallel line of work improves long-horizon reasoning and decision making through search, reflection, memory, and skill reuse. Tree of Thoughts and Language Agent Tree Search expand reasoning into explicit search over intermediate thoughts or action trajectories (Yao et al., 2023a; Zhou et al., 2024); Reflexion uses verbal feedback to improve future attempts (Shinn et al., 2023); Voyager stores reusable skills for open-ended embodied exploration (Wang et al., 2023); Generative Agents and MemGPT develop memory mechanisms for persistent behavior and long-context interaction (Park et al., 2023; Packer et al., 2023). These works primarily improve agent capability, planning, or context management. In contrast, our work targets the systems-level latency of obtaining environment observations during agent execution.

LLM & Agents inference acceleration. To reduce latency in LLM systems, a large body of work accelerates execution at the model, token, and action levels. Model-serving systems improve throughput and latency through IO-aware attention, batching, scheduling, and KV-cache management (Dao et al., 2022; Yu et al., 2022; Kwon et al., 2023; Zheng et al., 2024). Token-level methods accelerate autoregressive generation by proposing and verifying multiple tokens per step, including draft-model speculative decoding (Leviathan et al., 2023; Chen et al., 2023), tree-based or multi-head extensions (Miao et al., 2023; Cai et al., 2024), and feature-level speculative sampling (Li et al., 2024). At the action level, recent systems exploit structure in the agent trajectory to move work off

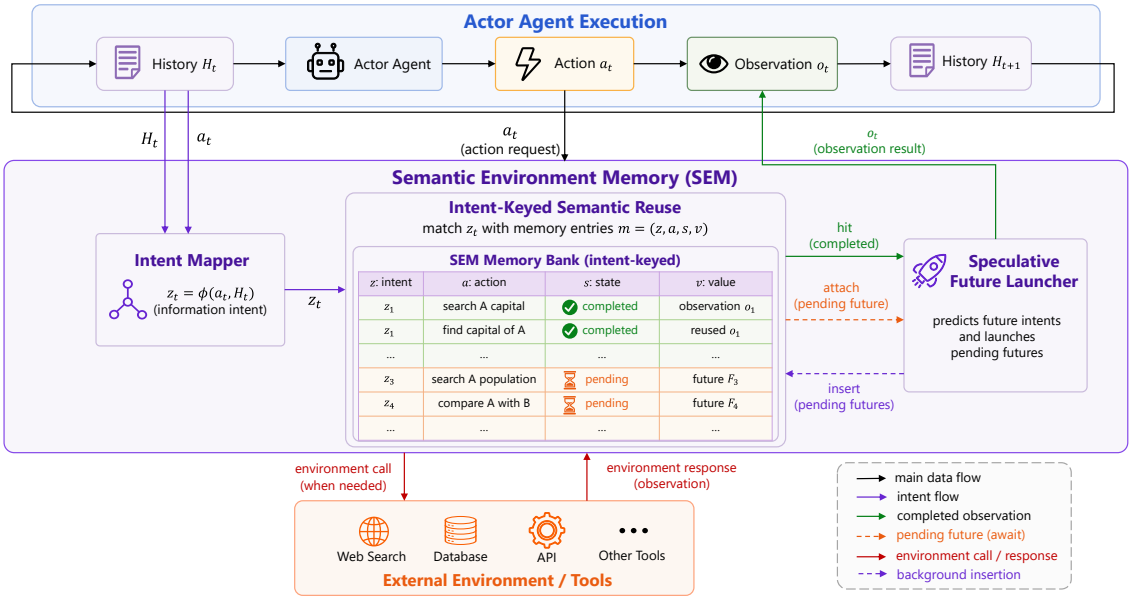

Figure 2: Overview of Semantic Environment Memory (SEM). SEM sits between the actor agent and external environment, mapping actions to information intents, reusing intent-compatible completed observations or pending futures, and launching speculative futures for later reuse.

the critical path: LLMCompiler identifies independent function calls for parallel execution (Kim et al., 2024), while Speculative Actions predicts future actions and pre-launches their associated environment calls (Ye et al., 2025). SEM is orthogonal to these methods: rather than accelerating token generation or exact predicted actions, it reduces exposed environment-observation latency by reusing intent-compatible completed observations and pending futures.

Semantic caching for LLM systems. Semantic caching reduces serving cost by reusing responses for semantically similar user queries rather than exact string matches (Bang et al., 2024; Liu et al., 2024; Wang et al., 2025). Context-aware variants further consider dialogue history when deciding cache reuse (Li et al., 2025). Unlike systems that cache LLM responses, SEM caches environment interactions (e.g., tool observations, futures, and rollouts) within an agent loop. This is crucial because reusing interactions for intermediate agent actions must preserve reasoning trajectories while reducing observation latency.

3 Semantic Environment Memory

In this section, we introduce ***Semantic Environment Memory*** (SEM), a latency-aware layer between the actor agent and the external environment as shown in Fig. 2. SEM maintains a unified memory over environment interactions, where each entry is associated with an information intent and may correspond to either a returned observation or an in-flight tool call. The memory is accessed through *Intent-Keyed Semantic Reuse*, which determines if the actor's current action can be served from an existing entry. SEM is then continuously enriched by a *Speculative Future Launcher*, which runs in the background and adds new pending entries for likely future information needs. We begin with an overview of the shared memory representation and observation acquisition procedure, then explain how Intent-Keyed Semantic Reuse accesses this memory and how the Speculative Future Launcher populates it with reusable pending futures.

3.1 Overview

Given the actor history H_t and action a_t, SEM first maps the action to an information intent:

$$z_t = \phi(a_t, H_t), \tag{1}$$

where z_t captures the tool type, entity, attribute, slots, or contextual references needed to identify the environment information requested by a_t. SEM then searches memory $\mathcal{M}$ using a_t and z_t: exact action lookup is attempted first, followed by intent-keyed lookup. A SEM entry is represented as:

$$m = (z, a, s, v), \tag{2}$$

where z is the stored information intent, a is the original action, s is the entry state, and v

Algorithm 1 Observation acquisition with Semantic Environment Memory

Require: History H_t, actor action a_t, memory $\mathcal{M}$, environment $\mathcal{E}$

$z_t \leftarrow \phi(a_t, H_t)$
$m \leftarrow \textsc{ExactLookup}(\mathcal{M}, a_t)$
if $m = \varnothing$ **then**
  $m \leftarrow \textsc{IntentLookup}(\mathcal{M}, z_t)$
end if
if $m.\text{state} = \textsc{completed}$ **then**
  return $m.\text{observation}$
else if $m.\text{state} = \textsc{pending}$ **then**
  $o_t \leftarrow \textsc{Await}(m.\text{future})$
  $\mathcal{M} \leftarrow \textsc{MarkCompleted}(\mathcal{M}, m, o_t)$
  return o_t
else
  $F_t \leftarrow \mathcal{E}.\textsc{Call}(a_t)$
  $\mathcal{M} \leftarrow \textsc{Insert}(\mathcal{M}, z_t, a_t, \textsc{pending}, F_t)$
  $o_t \leftarrow \textsc{Await}(F_t)$
  $\mathcal{M} \leftarrow \textsc{MarkCompleted}(\mathcal{M}, z_t, a_t, o_t)$
  return o_t
end if

is the stored value. The state s can be COMPLETED, PENDING, FAILED, or EXPIRED. When s = COMPLETED, v is an observation. When s = PENDING, v is a future that will resolve to an observation. Entries inserted by the Speculative Future Launcher also use the same form in Eq. 2, with a corresponding to the predicted future action and v corresponding to the launched future.

This unified representation allows $\mathcal{M}$ to store both returned environment results and in-flight environment calls. Intent-Keyed Semantic Reuse operates over entries $m \in \mathcal{M}$ by checking whether their stored intent z is compatible with the current intent z_t. The Speculative Future Launcher expands $\mathcal{M}$ by creating pending entries before the actor explicitly requests them.

Algorithm 1 summarizes observation acquisition. SEM first computes z_t, tries exact lookup, and then intent-keyed lookup. Completed entries return $m.v$ directly, pending entries await $m.v$, and misses fall back to a fresh environment call that is inserted into $\mathcal{M}$ as pending before being marked completed. Thus, SEM shortens observation acquisition without changing the actor's selected action.

3.2 Intent-Keyed Semantic Reuse

Existing action-level reuse is often tied to exact action strings, which is too restrictive for language agents whose actions are natural-language requests. To address this limitation, we propose *Intent-Keyed Semantic Reuse*, which uses the information intent z_t in Eq. 1 as the semantic key for retrieving reusable environment work from $\mathcal{M}$.

Concretely, the module receives the current actor action a_t and computes $z_t = \phi(a_t, H_t)$. The function ϕ extracts task-relevant fields from the action and its context. For search tasks, these fields may include tool type, entity, relation, and requested attribute. For retail tasks, they may include tool name, customer identifier, order identifier, product identifier, policy category, and slot constraints. The goal is not to solve full semantic parsing, but to obtain a lightweight representation precise enough to identify when two tool interactions are likely to serve the same information need.

Intent-Keyed Semantic Reuse performs reuse in two stages. It first applies EXACTLOOKUP over a_t, which captures the strongest and safest reuse case. If this lookup fails, the module applies INTENTLOOKUP over z_t, searching for an entry $m_i = (z_i, a_i, s_i, v_i) \in \mathcal{M}$ whose stored intent z_i is compatible with z_t. We define compatibility conservatively:

$$\begin{aligned}\mathrm{Comp}(z_t, z_i) = \; & \mathrm{ToolMatch}(z_t, z_i) \\ & \wedge \mathrm{EntityOverlap}(z_t, z_i) \\ & \wedge \mathrm{AttrCompat}(z_t, z_i).\end{aligned} \tag{3}$$

In this way, tool matching prevents reuse across tools with different semantics. Entity overlap ensures that m_i concerns the same target object or a contextually equivalent reference. Attribute compatibility prevents over-broad reuse, such as using an observation about one attribute to answer a request about another.

The same compatibility rule enables two distinct forms of reuse. If s_i = COMPLETED, then v_i represents a completed observation, which Intent-Keyed Semantic Reuse can directly return for a_t. Conversely, if s_i = PENDING, v_i acts as a future and the module can await it rather than launching another environment call. Consequently, Intent-Keyed Semantic Reuse unifies both observation reuse and future reuse within the single memory interface defined in Eq. 2.

This design is intentionally conservative. While Intent-Keyed Semantic Reuse expands reuse beyond exact surface strings, it avoids reusing entries based solely on the linguistic similarity of actions. In the default implementation, observation compat-

ibility is approximated by $\mathrm{Comp}(z_t, z_i)$, thereby avoiding the overhead of an additional LLM judge or embedding verifier on the critical path. Furthermore, completed entries in $\mathcal{M}$ also serve as a drift recovery cache: when an agent reformulates a query, revisits an entity, or repeats grounding after an uncertain step, a later intent z_t can recover a compatible completed entry m_i rather than issuing another environment call.

3.3 Speculative Future Launcher

Although Intent-Keyed Semantic Reuse can reuse existing entries, it only leverages them if such entries already exist in $\mathcal{M}$. To further improve the reuse efficiency, we introduce a *Speculative Future Launcher* to proactively launch environment calls for likely future information needs and store them as pending entries.

This module is motivated by the fact that environment calls become valuable before they are completed. Suppose a future v_i for intent z_i is launched at time τ_{launch}, and the actor later requests a compatible intent z_t at time τ_{need}. If the call would normally take L_{env}, the exposed latency is:

$$L_{\text{exposed}} = \max\left(0,\, L_{\text{env}} - (\tau_{\text{need}} - \tau_{\text{launch}})\right). \tag{4}$$

Therefore, a pending future represents more than a simple binary hit or miss. The earlier the launcher registers a compatible pending entry $m_i = (z_i, a_i, \text{PENDING}, v_i)$, the less latency remains when Intent-Keyed Semantic Reuse later matches z_t to z_i.

Pending entries in $\mathcal{M}$ are created in two ways. During foreground execution, the observation-acquisition path immediately inserts $(z_t, a_t, \text{PENDING}, F_t)$ into $\mathcal{M}$ before awaiting the environment call F_t. This allows duplicate or reformulated requests that arrive while F_t is still running to attach to the same future. Beyond this foreground insertion, the Speculative Future Launcher runs in the background to create pending entries before the actor explicitly asks for them.

To construct speculative futures, the launcher leverages the current history H_t and action a_t. For search-style tasks, the next useful action often depends on the observation that the current action will return. Therefore, the launcher first predicts a compact guessed observation, denoted as $\hat{o}_t$, and constructs a virtual next state from H_t, a_t, and $\hat{o}_t$. From this virtual state, it generates candidate future actions, maps each candidate $\hat{a}^{(j)}_{t+1}$ to an intent $\hat{z}^{(j)}_{t+1} = \phi(\hat{a}^{(j)}_{t+1}, \hat{H}_{t+1})$, launches the corresponding environment call as a future $\hat{F}^{(j)}_{t+1}$, and inserts $(\hat{z}^{(j)}_{t+1}, \hat{a}^{(j)}_{t+1}, \text{PENDING}, \hat{F}^{(j)}_{t+1})$ into $\mathcal{M}$. For retail tool-use tasks, where tool-call sequences are often governed by business workflows, the launcher can directly predict candidate future tool calls from the dialogue and tool history.

The guessed observation $\hat{o}_t$ is never returned to the actor and does not affect the real trajectory; it only helps the launcher decide which real environment calls to start early. If a later intent z_t matches a speculative intent $\hat{z}^{(j)}_{t+1}$, Intent-Keyed Semantic Reuse awaits the actual future $\hat{F}^{(j)}_{t+1}$, hiding latency already paid since launch; otherwise, the branch expires unused. The launcher width k controls the coverage-overhead trade-off: larger k prepares more pending futures but adds speculative calls. Since all speculative futures use the memory form in Eq. 2, duplicate or semantically equivalent entries can be merged in $\mathcal{M}$.

3.4 Nested Token- & Action-Level Speculation

The preceding sections describe how SEM reduces the exposed latency of environment observation acquisition once the actor's action a_t is known, with Intent-Keyed Semantic Reuse serving the corresponding observation o_t from $\mathcal{M}$ whenever possible. This action-level latency reduction is orthogonal to another source of delay in the agent loop: the time required for the actor model to generate its next thought and action. We therefore view token-level speculative decoding not as a component of SEM, but as an additional acceleration layer that can be nested with SEM.

At step t, the actor conditions on history H_t and generates an action a_t. Token-level speculative decoding accelerates this generation process by using a draft model to propose candidate tokens and a target model to verify them. This reduces the generation latency $T_{\text{gen}}(a_t \mid H_t)$, but does not shorten the wait for the next observation o_t.

SEM addresses this second latency source. After a_t is selected, the Intent Mapper computes $z_t = \phi(a_t, H_t)$, and Intent-Keyed Semantic Reuse searches $\mathcal{M}$ to obtain o_t from a completed entry, a pending future, or, if no compatible entry exists, a fresh environment call. Thus SEM reduces the exposed environment latency $T_{\text{env}}(o_t \mid a_t, z_t, \mathcal{M})$, while leaving the actor's action choice unchanged.

Table 1: Main results on multi-hop search tasks. SEM combines intent-keyed reuse of completed observations and pending futures, while NESTED SEM further adds token-level speculation. Lower time is better.

Method	HotpotQA					2WikiMultiHopQA				
	Acc. ↑	Time ↓	Speedup ↑	Future Hit Rate ↑	Obs Reuse / Task ↑	Acc. ↑	Time ↓	Speedup ↑	Future Hit Rate ↑	Obs Reuse / Task ↑
Serial Agent	66%	23.70s	1.00×	–	–	65%	21.73s	1.00×	–	–
Token Spec.	66%	20.87s	1.14×	–	–	65%	18.86s	1.15×	–	–
Exact Future	67%	20.08s	1.18×	28.57%	–	65%	17.82s	1.22×	31.58%	–
Intent Future	65%	18.85s	1.26×	42.86%	–	63%	16.54s	1.31×	47.37%	–
Intent Obs.	64%	18.43s	1.29×	–	1.50	60%	16.17s	1.34×	–	1.86
SEM	66%	15.97s	1.48×	40.91%	1.17	63%	13.43s	1.62×	42.95%	1.44
Nested SEM	66%	**13.47s**	**1.76**×	40.91%	1.17	63%	**11.24s**	**1.93**×	42.95%	1.44

Table 2: Main results on τ^2-bench retail tool-use tasks. SEM combines intent-keyed observation and future reuse, while NESTED SEM further adds token-level speculation. Lower time is better.

Method	τ^2-bench Base					τ^2-bench Read-heavy				
	Success ↑	Time ↓	Speedup ↑	Future Hit Rate ↑	Obs Reuse / Task ↑	Success ↑	Time ↓	Speedup ↑	Future Hit Rate ↑	Obs Reuse / Task ↑
Serial Agent	70%	44.10s	1.00×	–	–	83%	39.71s	1.00×	–	–
Token Spec.	70%	37.84s	1.17×	–	–	83%	34.17s	1.16×	–	–
Exact Future	72%	36.53s	1.21×	32.26%	–	83%	31.94s	1.24×	35.09%	–
Intent Future	68%	33.93s	1.30×	43.33%	–	80%	29.87s	1.33×	47.37%	–
Intent Obs.	70%	39.12s	1.13×	–	1.65	78%	33.73s	1.18×	–	1.23
SEM	68%	31.18s	1.41×	43.33%	1.25	80%	27.29s	1.46×	45.61%	1.15
Nested SEM	68%	**26.19s**	**1.68**×	43.33%	1.25	80%	**22.96s**	**1.73**×	45.61%	1.15

The resulting nested execution time is as follows:

$$T_{\text{agent}}^{\text{nested}} \approx \sum_{t=1}^{T} \Big(T_{\text{gen}}^{\text{token-spec}}(a_t \mid H_t) + T_{\text{env}}^{\text{SEM}}(o_t \mid a_t, z_t, \mathcal{M}) \Big). \tag{5}$$

where $T_{\text{gen}}^{\text{token-spec}}(a_t \mid H_t)$ denotes the latency of producing the actor action a_t under token-level speculative decoding, and $T_{\text{env}}^{\text{SEM}}(o_t \mid a_t, z_t, \mathcal{M})$ denotes the remaining latency for acquiring o_t through SEM's memory-mediated observation-acquisition path. The two terms correspond to different parts of the reasoning-action-observation loop: token-level speculation shortens action generation, whereas SEM shortens observation acquisition by reusing completed observations and pending futures. Because SEM only changes how o_t is obtained for the already chosen a_t, it can be composed with token-level acceleration while preserving the original actor policy.

4 Experiment

4.1 Experimental Setup

Benchmarks. For multi-hop search, we use HotpotQA (Yang et al., 2018) and 2WikiMultiHopQA (Ho et al., 2020). HotpotQA requires agents to gather evidence across multiple Wikipedia articles, while 2WikiMultiHopQA emphasizes compositional multi-hop reasoning over linked entities. For tool-use, we use the retail domain from τ^2-bench (Barres et al., 2025), a conversational agent benchmark with domain-specific tools, policies, and mutable environment states. We report results on the base retail split and a read-heavy subset that emphasizes information lookup and retrieval-style tool calls.

Models and hardware. We use Qwen3 models (Yang et al., 2025) throughout. The actor is Qwen3-32B. SEM uses Qwen3-1.7B as the speculator, and Nested SEM also uses Qwen3-1.7B as the draft model for token-level speculation. The actor is loaded on two GPUs with tensor parallelism, and the speculator/draft model is loaded on the remaining GPU via in-process vLLM.

Latency setting. We use offline environments with controlled latency injection. For HotpotQA and 2WikiMultiHopQA, search calls use a 2.0s base delay, ±0.5s jitter, and 0.1s per returned document. For τ^2-bench, read tools use 2.0s ±0.5s delay, write tools use 4.0s ±0.5s delay, and local

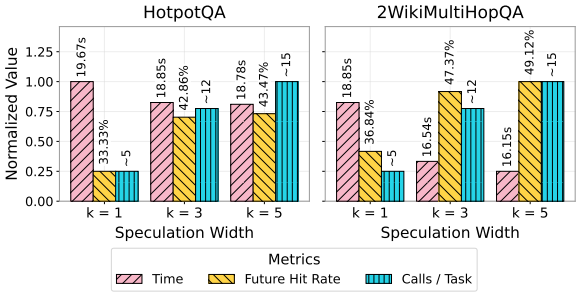

Figure 3: Ablation on speculation width k.

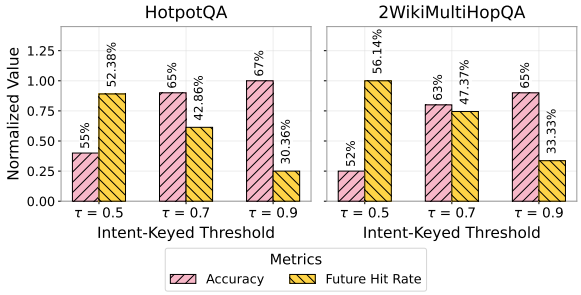

Figure 4: Ablation on the matching threshold τ.

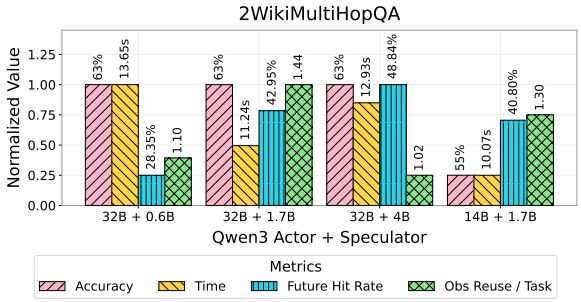

Figure 5: Ablation on Qwen3 actor and speculator scale.

tools use no delay. This isolates latency-hiding effects from external service noise.

Compared methods. SERIAL AGENT is the unaccelerated baseline. TOKEN SPEC. applies only token-level speculation. EXACT FUTURE prelaunches futures but reuses them only under exact action matching. INTENT FUTURE reuses launched futures by information intent. INTENT OBS. reuses completed observations by information intent. SEM combines intent-keyed future and observation reuse. NESTED SEM further adds token-level speculation.

Metrics. We report accuracy for HotpotQA and 2WikiMultiHopQA, success rate for τ^2-bench, wall-clock time, speedup over SERIAL AGENT, *Future Hit Rate*, and *Obs Reuse / Task*.

4.2 Main Results

Tab. 1 and Tab. 2 show the main results. SEM consistently reduces wall-clock time while maintaining baseline task quality. On HotpotQA, EXACT FUTURE achieves 1.18× speedup with a 28.57% Future Hit Rate, while INTENT FUTURE improves it to 42.86% and speedup to 1.26×. Full SEM reaches 1.48×, and NESTED SEM further improves to 1.76×. On 2WikiMultiHopQA, SEM reduces time from 21.73s to 13.43s, and NESTED SEM achieves the best speedup of 1.93×. Retail tool-use shows the same trend. On τ^2-bench Base, SEM improves speedup to 1.41×, and NESTED SEM reaches 1.68×. On the read-heavy subset, SEM reduces time from 39.71s to 27.29s, while NESTED SEM reaches 22.96s (1.73× speedup).

These results show the contribution of each module. INTENT FUTURE improves over EXACT FUTURE by relaxing exact action matching into intent-keyed future reuse. INTENT OBS. captures reusable completed observations. Combining both in SEM gives stronger latency reduction, and adding token-level speculation in NESTED SEM further reduces the remaining generation cost.

4.3 Ablation Study

To further verify the effectiveness of each design in SEM, we conduct extensive ablation studies as shown in Tab. 1 and Tab. 2 and key design choices in Tab. 3–Tab. 5.

Reuse criteria. As shown in Tab. 1 and Tab. 2, Intent-Keyed Semantic Reuse improves future reuse over exact action matching. On HotpotQA, INTENT FUTURE increases Future Hit Rate from 28.57% to 42.86% over EXACT FUTURE, and improves speedup from 1.18× to 1.26×; on 2WikiMultiHopQA, the hit rate similarly rises from 31.58% to 47.37%. This shows that mapping a_t to $z_t = \phi(a_t, H_t)$ lets SEM match compatible pending entries in $\mathcal{M}$ beyond surface-form actions.

Future vs. observation reuse. As shown in Tab. 1 and Tab. 2, INTENT FUTURE and INTENT OBS. leverage different states of the memory tuple $m = (z, a, s, v)$. Specifically, $s =$ PENDING masks the remaining tool latency by awaiting the observation v, whereas $s =$ COMPLETED directly reuses the cached observation v. On 2WikiMultiHopQA, INTENT FUTURE and INTENT OBS. reduce the execution time to 16.54s and 16.17s, respectively. Conversely, on the τ^2-bench Base

Table 3: Ablation on speculation width k. Larger k lets the Speculative Future Launcher insert more pending futures into $\mathcal{M}$, which increases Future Hit Rate but launches more speculative environment calls per task.

Dataset	k	Time ↓	Future Hit Rate ↑	Calls / Task ↑
HotpotQA	1	19.67s	33.33%	∼5
	3	18.85s	42.86%	∼12
	5	18.78s	43.47%	∼15
2WikiMultiHopQA	1	18.85s	36.84%	∼5
	3	16.54s	47.37%	∼12
	5	16.15s	49.12%	∼15

Table 4: Ablation on the information-intent matching threshold τ. Lower τ admits more intent-compatible entries in $\mathcal{M}$, increasing Future Hit Rate but risking less reliable reuse; higher τ is more conservative and improves accuracy at the cost of fewer future hits.

Dataset	τ	Acc. ↑	Future Hit Rate ↓
HotpotQA	0.5	55%	52.38%
	0.7	65%	42.86%
	0.9	67%	30.36%
2WikiMultiHopQA	0.5	52%	56.14%
	0.7	63%	47.37%
	0.9	65%	33.33%

dataset, future reuse proves more effective, reducing the time to 33.93s compared to 39.12s for observation reuse. Combining both strategies in SEM yields the optimal non-token-level acceleration across all evaluated tasks.

Token Spec., SEM, and Nested SEM. Tab. 1 and Tab. 2 show that TOKEN SPEC. reduces generation latency, while SEM mitigates observation latency. Compared to the SERIAL AGENT, TOKEN SPEC. yields modest speedups, such as $1.14\times$ on HotpotQA and $1.17\times$ on the τ^2-bench Base. SEM demonstrates stronger performance by reusing intent-compatible futures and observations, reaching speedups of $1.48\times$ and $1.41\times$ on the same tasks. NESTED SEM addresses both latency bottlenecks simultaneously, achieving peak speedups—up to $1.93\times$ on 2WikiMultiHopQA and $1.73\times$ on the τ^2-bench Read-heavy.

Speculation width k. Fig. 3 and Tab. 3 show that increasing launcher width k creates more pending futures in $\mathcal{M}$, improving coverage but increasing speculative calls. On 2WikiMultiHopQA, Future Hit Rate rises from 36.84% at $k = 1$ to 49.12% at $k = 5$, while calls per task grow from about 5 to 15. Since $k = 3$ already achieves most of the latency gain with fewer calls, we use $k = 3$ as the default trade-off between coverage and overhead.

Table 5: Ablation on Qwen3 model scale on 2WikiMultiHopQA. Each setting reports the Qwen3 actor and the Qwen3 speculator used by the Speculative Future Launcher. Larger speculators improve Future Hit Rate, while smaller actors or speculators reduce latency with different accuracy trade-offs.

Model Setting	Acc.	Time	Future Hit Rate	Obs Reuse / Task
32B + 0.6B	63%	13.65s	28.35%	1.10
32B + 1.7B	63%	11.24s	42.95%	1.44
32B + 4B	63%	12.93s	48.84%	1.02
14B + 1.7B	55%	10.07s	40.80%	1.30

Intent-keyed threshold τ. Fig. 4 and Tab. 4 show that the threshold τ controls how conservatively Intent-Keyed Semantic Reuse matches z_t with stored intents z_i. Lower τ improves hit rate but can hurt reliability: on HotpotQA, $\tau = 0.5$ gives 52.38% Future Hit Rate but 55% accuracy, while $\tau = 0.9$ gives 30.36% Future Hit Rate and 67% accuracy. We use $\tau = 0.7$, which provides a better balance between reuse coverage and task accuracy.

Model scale. Fig. 5 and Tab. 5 study Qwen3 actor/speculator scale on 2WikiMultiHopQA. With a Qwen3-32B actor, increasing the Qwen3 speculator from 0.6B to 1.7B improves Future Hit Rate from 28.35% to 42.95% and reduces time from 13.65s to 11.24s on 2WikiMultiHopQA. A larger 4B speculator further improves hit rate but increases time, while a 14B actor reduces latency at a clear accuracy cost. Thus, Qwen3-32B + Qwen3-1.7B gives the best overall setting for SEM.

5 Conclusion

In this paper, we introduced *Semantic Environment Memory* (SEM), a latency-aware layer that accelerates tool-using LLM agents by changing how observations are acquired rather than how the actor reasons. SEM indexes environment interactions by information intent, enabling reuse of both completed observations and pending futures while preserving the actor's original execution policy. Experiments on multi-hop search and retail tool-use benchmarks show that SEM consistently reduces end-to-end latency while maintaining task quality, and can be combined with token-level speculation for further acceleration. These results suggest that agent acceleration should consider not only faster generation, but also how external observations are prepared, matched, and reused.

Limitations

SEM shares limitations common to current tool-using agent acceleration methods. Its effectiveness depends on reliable intent matching: permissive matching may reuse unsuitable observations, while conservative matching misses reuse opportunities. Speculative launching also introduces extra tool calls, making it less suitable for costly, rate-limited, or side-effectful tools without additional safeguards. Finally, SEM reduces exposed observation latency but does not solve broader challenges in long-horizon planning, dynamic environments, or tools whose outputs change rapidly over time.

References

Fuad Bang, Haodong Chen, Yuxuan Luo, Shahbaz Sohail, and Ahmed Ali. 2024. GPT semantic cache: Reducing LLM costs and latency via semantic embedding caching. *arXiv preprint arXiv:2411.05276*.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, and Huan Sun. 2023. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, and 1 others. 2022. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An llm compiler for parallel function calling. In *International Conference on Machine Learning*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*.

Minghao Li, Yifan Zhang, Wei Chen, and Xiang Liu. 2025. ContextCache: Context-aware semantic cache for multi-turn queries in large language models. *arXiv preprint arXiv:2506.22791*.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*.

Yuhan Liu, Heng Xu, Zheng Li, Yuxin Zhang, Rui Wang, and Tianxiang Chen. 2024. MeanCache: User-centric semantic caching for LLM web services. *arXiv preprint arXiv:2403.02694*.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Zhang, Zhuoming Wong, Zhuohan Chen, Daiyaan Arfeen, Ritik Abhyankar, and Zhihao Jia. 2023. SpecInfer: Accelerating generative large language model serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

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

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th*

Annual ACM Symposium on User Interface Software and Technology.

Shishir G. Patil and 1 others. 2023. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*.

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

Guanzhi Wang and 1 others. 2023. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Zhe Wang, Yuxuan Zhang, Chang Liu, and Tianxiang Chen. 2025. Semantic caching for low-cost LLM serving: From offline learning to online adaptation. *IEEE INFOCOM*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36.

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

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. 2025. Speculative actions: A lossless framework for faster agentic systems. *arXiv preprint arXiv:2510.04371*.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation*, pages 521–538.

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

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.