COMEM: CONTEXT MANAGEMENT WITH A DECOUPLED LONG-CONTEXT MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Context management enables agentic models to solve long-horizon tasks through iterative summarization of previous interaction histories. However, this process typically incurs substantial decoding overhead for the extra summarization tokens, which significantly affect the end-to-end response latency at deployment. In this paper, we introduce COMEM, a novel framework that decouples memory management from the primary agent workflow, enabling these processes to execute in parallel. We propose a k-step-off asynchronous pipeline that overlaps the memory model's summarization with the agent's inference, effectively masking the latency of context processing. To ensure robustness under this asynchronous setting, we introduce a reward-driven training strategy that aligns the memory model to capture sufficient statistics for the agent's decision-making. Theoretical analysis confirms that COMEM offers a superior efficiency-effectiveness trade-off compared to coupled architectures. Our extensive experimental results on SWE-Bench-Verified show that COMEM provides 1.4x latency improvements upon vanilla long-context solutions while preserving most of the performance. Furthermore, we demonstrate that these latency gains scale favorably with increased system throughput, offering a modular path forward for the independent optimization of agent reasoning and memory compression.

1 INTRODUCTION

The capabilities of Large Language Model (LLM) agents have expanded significantly, moving beyond simple conversational assistants to autonomous systems capable of tackling complex, multi-step problems in domains such as software engineering (Jimenez et al., 2023), scientific discovery Lu et al. (2024), and open-ended exploration He et al. (2024); Team et al. (2025). Unlike standard question-answering tasks, these applications are inherently stateful: success depends not only on the immediate instruction but on the agent's ability to maintain a coherent understanding of its entire interaction history. For instance, in repository-level code generation, an agent must recall decisions made thousands of steps prior to generate valid subsequent actions, such as function definitions, dependency changes, or debugging results. Consequently, the ability to process and reason over extremely long contexts has become a non-negotiable requirement for high-performing agentic systems Zhou et al. (2025); Wu et al. (2025b); Lu et al. (2025); Sun et al. (2025); Yu et al. (2025).

Despite the increasing attention to increase effective context window, there are surprisingly fewer attention spread on the severe computational cost introduced from long-context processing during agentic inference. While modern LLMs can natively accept context windows spanning millions of tokens, utilizing them during online agent deployment is often prohibitively expensive Yuan et al. (2024). The primary challenge lies in the inference latency of the decoding stage. As the interaction history grows, the KV cache footprint expands linearly, and the attention computation cost scales, saturating the High Bandwidth Memory (HBM) of modern GPUs. Unlike the prefilling phase, which is compute-bound and parallelizable, the auto-regressive decoding phase is memory-bound; fetching the entire KV cache for every generated token results in high latency that degrades the user experience and limits the throughput of real-time systems. This creates a "memory wall" where the agent's reasoning power is throttled by the sheer volume of its past observations Tang et al. (2024).

To address these computational challenges, previous solutions have largely pursued two directions: context reduction and system-level optimization. Context reduction strategies attempt to limit the

number of tokens the model must attend to. Sliding-window attention (Fu et al., 2025) restricts the agent to a fixed local context, inherently sacrificing the ability to recall distant dependencies—such as the initial user goal or a bug identified thousands of steps prior. Retrieval-Augmented Generation (RAG) Wu et al. (2025a); Shi et al. (2026) alleviates this by fetching relevant snippets from the history. However, RAG relies heavily on semantic similarity, which often fails to capture the high-level state or procedural trajectory of an agent, leading to fragmented reasoning.

On the other hand, system-level Optimizations aim to make processing the full context more efficient without reducing the token count. Modern inference engines like vLLM Kwon et al. (2023) have revolutionized memory management through PagedAttention, significantly reducing fragmentation. Similarly, Sparse Attention mechanisms Xiao et al. (2024) and KV Cache Quantization Hooper et al. (2025) reduce the compute and memory bandwidth requirements by sparsifying interactions or lowering numerical precision. KV cache can also be offloaded to CPU for future decoding Jin et al. (2024). While these methods provide substantial throughput improvements, they do not solve the fundamental inefficiency of the agent loop: the redundancy of re-encoding or re-attending to a massive, largely static history for every single decoding step. As a result, even highly optimized engines eventually hit a latency wall as the interaction trace grows indefinitely.

To bridge this gap, we introduce COMEM, a framework that decouples the distinct responsibilities of memory management and agentic reasoning. Unlike prior approaches that force a single large model to handle both history compression and policy generation, COMEM offloads the heavy lifting of long-context processing to a dedicated, lightweight summarization model. This architectural separation allows us to propose a novel k-step-off asynchronous pipeline, where the memory model continuously compresses history in the background, freeing the main agent to decode with a significantly reduced context window. To ensure this compressed state effectively guides the agent, we introduce a reward-driven alignment strategy that trains the memory model to capture the "sufficient statistics" required for optimal decision-making.By shifting the computational burden from the critical path of decoding to a parallelizable background process, COMEM achieves the best of both worlds: the global reasoning capability of long-context models and the low-latency inference of short-context systems.

Our contributions are summarized as follows

- We propose COMEM, a framework that separates memory management from reasoning, enabling the use of specialized, lightweight models for efficient history compression.
- Asynchronous Inference: We design a k-step-off pipeline that overlaps memory summarization with agent execution, masking the latency of context processing and reducing the decoding overhead by up to $1.4\times$.
- We introduce a novel training methodology that aligns the memory model using a functional equivalence reward, ensuring the compressed summary recovers the full-context agent's policy without expensive online interaction.
- Extensive experiments on SWE-Bench-Verified demonstrate that COMEM significantly reduces latency while maintaining competitive performance against state-of-the-art long-context baselines.

2 PRELIMINARIES

2.1 STANDARD AGENT

Standard agent interaction is a **Partially Observable Markov Decision Process** (POMDP), where each process starts from an initial observation o_1 that usually contains a problem statement, and then a multi-turn interaction history of length T is generated as $\tau = (o_1, a_1, o_2, a_2, \ldots, o_T)$ where a_i usually contains a reasoning and a tool call, and o_i usually represents an observation returned from the environment. Different from a vanilla MDP process, the LLM agent has to carry the entire interaction history as input at each step, which results in a linearly scaling context length through the interaction.

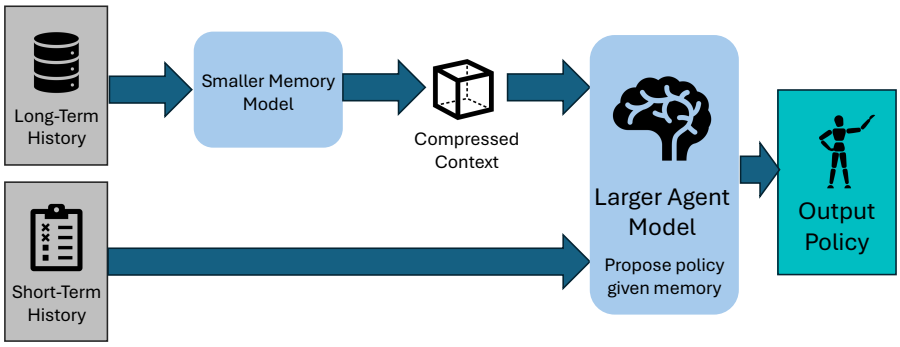

Figure 1: CoMem framework: a decoupled agent framework that offloads long-context compression to an asynchronous, lightweight memory model, significantly reducing inference latency without compromising reasoning performance.

2.2 LLM Inference

Modern LLMs usually employ a two-stage inference process: **Prefill Stage** and **Decode Stage**. The Prefill Stage serves as the initial step where the key-value cache (KV cache) for the prompt (also called prefix) sequences are built for each KV head in the Transformer architecture[1]. Suppose that each attention head has weights $\mathbf{W}_q$, $\mathbf{W}_k$ and $\mathbf{W}_o$, then the attention layers will create three vectors

$$\text{Query: } \mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_q, \quad \text{Key: } \mathbf{K} = \mathbf{X} \cdot \mathbf{W}_k, \quad \text{Value: } \mathbf{V} = \mathbf{X} \cdot \mathbf{W}_v \tag{1}$$

where $\mathbf{X} \in \mathbb{R}^{l \times d}$ is the input vector. After prefilling, all the key-value pairs are preserved in the memory so that they can be reused during the decoding of each token. This will result in two vectors $\mathbf{K}_{\text{cache}} = [\mathbf{K}_1; \ldots; \mathbf{K}_l] \in \mathbb{R}^{l \times d \times h}$ and $\mathbf{V}_{\text{cache}} = [\mathbf{V}_1; \ldots; \mathbf{V}_l] \in \mathbb{R}^{l \times d \times h}$, where we omitted the layers and use h to represent total number of KV heads in the architecture. The GPU High Bandwidth Memory (HBM) typically needs to spare a significant amount of spaces to store these KV cache. For instance, for `Qwen/Qwen3-32B` in FP16 precision, it will take up 16 GB to save a sequence of 64K tokens. In contrast, `Qwen/Qwen3-4B` requires roughly half of the space to save these amount of KV cache[2]. Finally, the Prefill Stage is usually **compute-bound**, since it involves matrix-matrix multiplications that utilize the GPU's arithmetic units efficiently. Typically, we use **Time to First Token** (TTFT) that calculates the time spent from waiting the first token to emit to represent the prefilling latency.

Once the prompt is processed, the model enters the Decoding Stage, which decodes token one at a time. For each step t, the model only processes the single previous token and concatenate a new KV pair,

$$\mathbf{K}_{\text{cache}}^{(t)} = [\mathbf{K}_{\text{cache}}^{(t-1)}; \mathbf{K}^{(t)}] \quad \mathbf{V}_{\text{cache}}^{(t)} = [\mathbf{V}_{\text{cache}}^{(t-1)}; \mathbf{V}^{(t)}] \tag{2}$$

and at the same time emits the output token. Different from Prefill Stage, Decode Stage is memory-bound since at each decoding step, the GPU spends more time waiting for the KV cache to arrive from HBM. Consequently, the decoding speed is usually bottlenecked by the memory bandwidth. When the HBM is saturated, part of the KV cache will be either emitted or offloaded to CPU Jin et al. (2024). Since CPU bandwidth is linear does not incur quadratic compute in long-context prefilling, **CPU offload** is often a better choice which results in significantly cheaper prefilling costs. Typically, we measure decoding efficiency through **Time per Output Token** (TPOT) , which tracks the average generation speed per token, excluding the initial prefill latency.

2.3 Group Relative Policy Optimization (GRPO)

In Group Relative Policy Optimization (GRPO) (Shao et al., 2024), for each prompt x, we sample a group of K trajectories $\mathcal{G}(x) = \{\tau^{(1)}, \ldots, \tau^{(K)}\}$ where each trajectory $\tau^{(k)} = (o_0^{(k)}, a_0^{(k)}, o_1^{(k)}, a_1^{(k)}, \ldots)$ is generated autoregressively by the policy π_θ and receives a trajectory-level scalar return $R(\tau^{(k)})$.

[1]Note that here we discriminate KV head from attention head because of Grouped Query Attention (GQA)

[2]`https://lmcache.ai/kv_cache_calculator.html` provides calculations for some other models

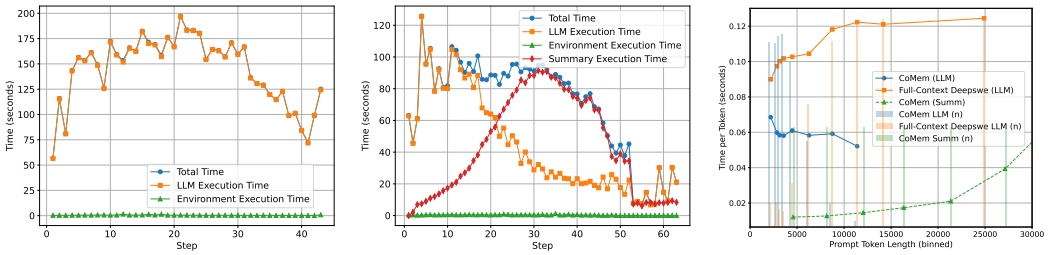

Figure 2: Profiling results for agentic inference scenario. Left is the execution time for Full-Context baseline. Middle the execution time for CoMem. Right is the time per completion token results.

GRPO constructs a *relative* advantage within each group by subtracting a group-wise baseline from the individual return. Concretely, we define $\tilde{A}^{(k)} = R(\tau^{(k)}) - b(\mathcal{G}(x))$ where $b(\mathcal{G}(x))$ is a baseline that depends only on the group statistics: $b(\mathcal{G}(x)) = \frac{1}{K}\sum_{j=1}^{K} R(\tau^{(j)})$.

The resulting policy gradient objective over a batch of groups is

$$J(\theta) \;=\; \mathbb{E}_{x,\,\mathcal{G}(x)}\left[\frac{1}{K}\sum_{k=1}^{K}\tilde{A}^{(k)}\sum_{t}\log \pi_\theta\big(a_t^{(k)} \mid s_t^{(k)}\big)\right], \tag{3}$$

where the inner sum runs over all decision tokens in the trajectory. In practice, we use a PPO-style clipped surrogate to stabilize training. Let $r_t^{(k)} = \frac{\pi_\theta(a_t^{(k)}|s_t^{(k)})}{\pi_{\theta_{\text{old}}}(a_t^{(k)}|s_t^{(k)})}$ denote the likelihood ratio; the GRPO loss is

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\mathbb{E}\left[\frac{1}{K}\sum_{k,t}\min\left(r_t^{(k)}\tilde{A}^{(k)} \quad \text{clip}\left(r_t^{(k)}, 1-\epsilon, 1+\epsilon\right)\tilde{A}^{(k)}\right)\right] \quad +\beta\,\mathbb{E}\left[D_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big)\right] \tag{4}$$

where ϵ controls the clipping range, β weights a KL penalty to a reference policy π_{ref}, and the expectation is taken over prompts, groups, and time steps.

3 Design of CoMem

We design CoMem to optimize the latency during high-throughput agentic inference. By decoupling the long-context processing from agentic process and then overlaps them via k-step-off generation, our method effectively alleviates the memory-bound bottlenecks and improves the end-to-end performance while keeping most of the performance. In the following sections, we first analyze the prefill and decoding latency for standard agentic inference. Next, we present the key insights of our proposed framework CoMem which includes a novel k-step-off generation algorithm and a new reward design that encourages the model to capture sufficient statistics. Finally, we show some theoretical analysis on the determination of a proper summary length.

3.1 Analysis of Agent Inference

In agentic inference, since at each step, only the current observation are prefilled, the total time is typically dominated by the decoding stage, which is often memory-bound because of the loading of KV cache. Thus, we mainly study the decoding speed in this section in both synthetic and real-time inference scenarios.

Long-context Scenario Real-time agentic inference often involves additional overheads from other processes such I/O and batch size invariants or token length variants during rollout. In order to better understand the relationships between the context length and decoding time, we first perform profiling analysis on a synthetic long-context scenario where the inputs and outputs are both random data with fixed length and the batch size is always constant. As shown in Figure 3, we can observe that when batch size is small (batch=1), the TPOT does not scale significantly with context length because the time spent on loading KV cache is negligible compared other overheads such as loading model weights or computing. For batch=32, at lower context ($< 2^{13}$), the KV cache is

still small enough that the GPU bandwidth is not saturated and the TPOT is dominated by the overheads. And at higher context ($> 2^{13}$), the KV cache becomes massive and every extra token adds a linear delay. For large batch (batch=128), the GPU will be fully saturated even at short contexts.

Agentic Scenario To further understand how the slow-down affects the inference latency in agentic scenario, we average the time for each step and components across multiple trajectories generated from SWE-Bench-Verified and then show the results in the left of Figure 2. First, the total time at each step is dominated by LLM execution while the environment execution is negligible. Second, we see that the LLM execution time increases from beginning to around 20 steps, and then starts to decrease until the last few steps. We attribute this behavior to the dynamics of the batch size. Initially, the large batch size renders the decoding process memory-bound. As the generation progresses and individual requests are completed, the effective batch size decreases, resulting in reduced inference latency. To isolate the behavior of decoding latency with respect to the prompt length, we further provide the plot of execution time per completion token in the right panel of Figure 2. It shows that the decoding latency per token increases as the prompt length grows, thereby corroborating our earlier hypothesis.

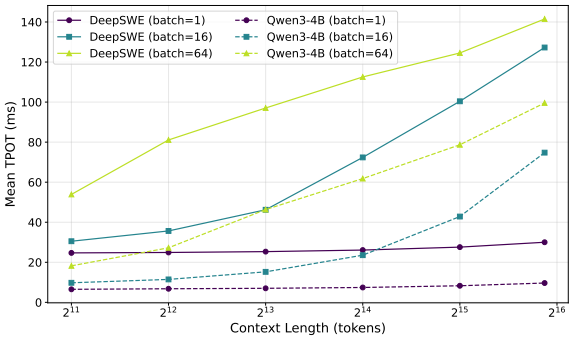

Figure 3: TPOT for two models on various context lengths and batch sizes. DeepSWE is a dense model with 32B parameters.

3.2 CoMem Framework

Through previous analysis, we show that the inference latency is mainly bottlenecked by the decoding stage. While most agent models are cumbersome large reasoning models, there exists several light weight long-context models that can offload the burden of long-context processing during decoding. In this section, we introduce CoMem (Figure 1), that decouples memory management from the main agent workflow by employing a dedicated smaller summarization model that compresses prior interactions into a concise representation for the agent. This design reduces latency and memory footprint since fewer parameters are involved, while effectively maintaining a long-context processing capability. Specifically, (1) **memory model** (f) is a native long-context model with a lightweight decoding mechanism (we choose a model with smaller parameter count) and designed to capture sufficient statistics from long-term history while (2) **agent model** (π) is a more capable model (usually with larger parameter counts) that proposes policies based on the outputs from f and the short-term memory.

Formally, the memory model f maps the long-term history to a compressed state representation $s_t = f(\tau_{<t}; \theta_f)$, where $|\theta_f| \ll |\theta_\pi|$, ensuring that the computational burden of long-context encoding is handled by a smaller, more efficient transformer. Moreover, the constraint $|s_t| < |\tau_t|$ guarantees that the input length for the agent is significantly reduced, resulting in lower decoding latency compared to standard full-context inference. Then the agent model proposes policy based on the compressed summary. $a_t \sim \pi(a|s_t, C_t; \theta_\pi)$ A naïve implementation of the workflow described above introduces a sequential bottleneck, as the agent model must idle until the memory update concludes. Next, we introduce a novel pipeline to remedy this extra overhead via overlapping the generation of two models.

k-step-off Pipeline To mitigate this, we propose a novel asynchronous pipeline detailed in Algorithm 1. In this design, the memory model operates in the background, continuously compressing history with a fixed lag of k steps ($\tau_{\leq t-k}$). This allows the agent model to proceed without interruption. To compensate for the latency of the memory model, the agent constructs its context by concatenating the available and slightly stale summary s with the recent raw interaction buffer τ_{t-k+1}. Formally, the policy generation at step t is defined as:

$$a_t \sim \pi(a_t \mid \underbrace{f(\tau_{\leq t-k})}_{\text{Latent Summary}}, \underbrace{\tau_{t-k+1:t}}_{\text{Recent Buffer}}; \theta_\pi) \tag{5}$$

Algorithm 1 k-step-off Pipeline

Require: Agent Model π, Memory Model f, Initial History τ_0, Retain Turns k
Initialize: $s \leftarrow \emptyset$, $t \leftarrow 0$, $\mathcal{P}_{mem} \leftarrow$ None
while not finished **do**
  {**Retrieve Summary from Previous Step**}
  if $\mathcal{P}_{mem} \neq$ None **then**
    $s \leftarrow$ await $\mathcal{P}_{mem}$ {Summary of $\tau_{\leq t-1-k}$}
  end if
  {**Start Async Memory Compression (Background)**}
  $\mathcal{P}_{mem} \leftarrow$ `async` $f(\tau_{\leq t-k})$ {Will be used at step $t+1$}
  {**Agent Inference (Parallel with $\mathcal{P}_{mem}$)**}
  Construct context $C_t \leftarrow (s, \tau_{[t-k+1:t]})$ {Summary + recent k turns}
  Sample action $a_t \sim \pi(a \mid C_t)$
  {**Environment Execution**}
  Execute a_t, observe o_{t+1}
  $\tau_{t+1} \leftarrow \tau_t \cup \{a_t, o_{t+1}\}$
  $t \leftarrow t+1$
end while

This design ensures that while the heavy lifting of long-context compression happens asynchronously, the agent always has access to the full causal history via both summary and the explicit short-term buffer.

3.3 Determination of Summary Length

While CoMem significantly reduces memory overhead during the decoding stage, the introduction of the summary representation s requires an additional prefilling stage for the agent model before further decoding, as shown in Figure 4. To ensure a net reduction in end-to-end latency, the decoding speedup must sufficiently amortize the prefilling cost. We formalize this trade-off below.

Theoretical Analysis Let S denote the KV cache memory size required per token (in GB), and let W represent the GPU's HBM bandwidth (in GB/s). Assuming the decoding process is memory-bound, the latency reduction per decoding step achieved by CoMem compared to a full-context baseline, is determined by the difference in data transfer volume

$$\Delta t_{\text{decode}} = (L_{\text{full}} - L_{\text{sum}}) \cdot S/W \tag{6}$$

where L_{full} and L_{sum} are the context lengths of the baseline and CoMem, respectively. However, generating the summary representation incurs a one-time prefilling latency. Assuming a prefilling throughput of P (tokens/s), the system must satisfy the condition where the cumulative time saved over Y generated tokens exceeds the summary prefilling latency:

$$Y \cdot \underbrace{\frac{(L_{\text{full}} - L_{\text{sum}}) \cdot S}{W}}_{\text{Time saved per token}} > \underbrace{\frac{L_{\text{sum}}}{P}}_{\text{Prefill overhead}} \tag{7}$$

Thus we derive an upper bound for the compression ratio:

$$L_{\text{sum}}/L_{\text{full}} < \frac{Y \cdot S \cdot P}{Y \cdot S \cdot P + W} \tag{8}$$

Case study Applying this bound to a standard deployment of `Qwen/Qwen3-32B` on a single A100 GPU, where the HBM bandwidth is $W \sim 2$ TB/s and the prefilling throughput is $P = 3,000$ tokens / s. And the KV size taken by each token is $2 \times 10^{-4} GB$. Assume that the average number of completion tokens is $Y = 1K$. Thus the ratio should be less than 0.23, which is achievable through summarization.

3.4 Training

Empirical analysis suggests that the inherent summarization capability of base models is often insufficient for complex agentic tasks. To bridge this gap, we introduce a targetted training pipeline designed to align the memory model's summarization performance with a full-context baselnie.

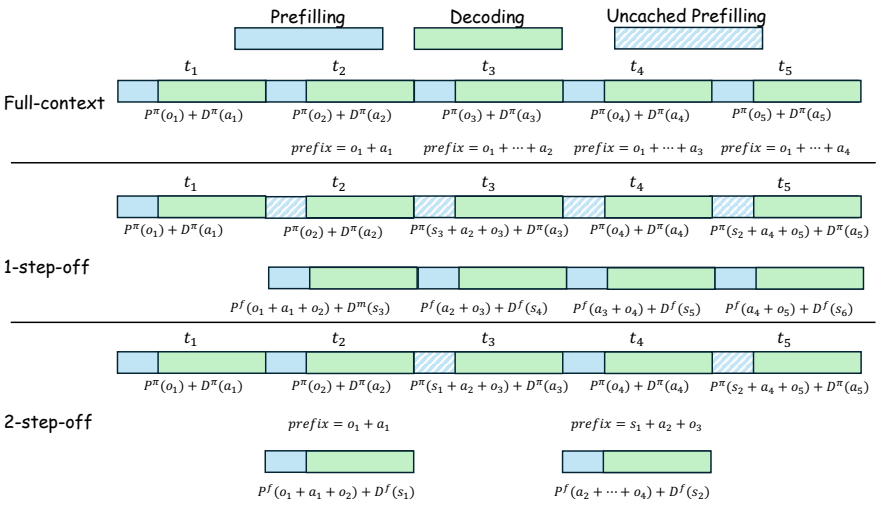

Figure 4: An example Gantt plot for k-step-off. We include two scenarios. For 1-step-off, the agent model has to prefill at each step. For 2-step-off, agent model can reuse summarize every 2 steps.

Crucially, our framework assumes the agent model π remains frozen throughout this process. This design choice decouples the training of the memory model f from the agent's reasoning policy, enabling a highly efficient offline training regime that circumvents the need for costly online environment interactions.

Specifically, we initialize the process by generating reference trajectories using the standard full-context agent pipeline. For each step t in a trajectory, we construct a training instance where the input is the historical context $\tau_{<t}$ and the target is the ground-truth action a_t^* originally generated by the full-context agent. We then employ GRPO (Eq. 4) to fine-tune the memory model. The core innovation lies in our reward formulation , which directly optimizes for functional equivalence rather than semantic similarity of the summary text. Mathematically, given a generated summary $s = f(\tau_{<t-k})$ and the immediate raw context $C_t = \tau_{t-k:t-1}$ containing the recent k turns. These are fed into the frozen agent model to obtain a proposed action. The reward is defined as the similarity between the proposal and the groundtruth

$$R(s) = \text{sim}(\pi(\cdot \mid s, C_t), a_t^*) \tag{9}$$

By maximizing this reward, the memory model learns to compress the long-term history into a representation that induces the agent to behave exactly as if it had access to the full context.

4 Experimental Results

4.1 Evaluation

We evaluate CoMem on SWE-Bench-Verified Jimenez et al. (2023), a human-validated subset of the original SWE-bench dataset designed to ensure reliable and reproducible evaluation of autonomous software engineering agents. Specifically, following Jain et al. (2025), we train our memory model using R2E-Gym-Lite and then test on 500 github issues on the SWE-Bench-Verified test set. We use the default scaffold provided by R2E-Gym and set maximum steps to 40 with an extended allowance to 100 steps to let the LLM submit the answer. Finally, we measure resolve rate, number of tool calls and inference latency for each batch of 128 issues.

4.2 Baselines and Implementations

We choose `Qwen3-4B` as our smaller memory model because of its scale and long-context capability. We always use the maximum summary length of 2048 for consistency. We pair the memory model with agent models with various sizes and capabilities. Specifically, we choose `DeepSWE`, `Qwen3-Coder-Max` and `GLM-4.7`. We compare with the following baselines: **Full-Context** is the standard agentic inference scenario with long-context inference. **No Summary** removes the

Table 1: Main results. For latency, we show the total inference time for 128 issues.

Agent	Memory	%Resolved	#Tool Calls	w/o CPU Offload		w/ CPU Offload	
				Latency (×128/s)	Speedup	Latency (×128/s)	Speedup
DeepSWE (32B Dense)	Full-Context	40.4	34.39	9457.78	1×	6390.62	1×
	Qwen3-4B (base)	29.9	24.74	4360.72	2.17×	3649.61	1.75×
	Qwen3-4B (SFT)	39.8	35.61	5232.64	1.81×	4168.85	1.53×
	GRPO$_{AC}$	**41.0**	40.69	5622.33	1.68×	4400.89	1.45×
Qwen3-Coder-Max (480B A35B)	Full-Context	**57.2**	36.68	6291.11	1×	5129.90	1×
	No Summary	46.6	25.20	3780.74	1.66×	3421.78	1.50×
	Qwen3-4B (base)	42.2	39.40	3421.63	1.84×	3138.97	1.63×
	Qwen3-4B (SFT)	47.3	43.39	3819.70	1.65×	3337.58	1.54×
	GRPO$_{AC}$	51.0	44.55	3917.68	1.61×	3594.51	1.43×
GLM-4.7 (355B 32B)	Full-Context	**69.0**	55.11	6361.49	1×	5869.42	1×
	No Summary	59.4	46.82	4216.83	1.51×	3842.17	1.53×
	Qwen3-4B (base)	58.3	49.32	3928.44	1.62×	3526.91	1.66×
	Qwen3-4B (SFT)	61.3	52.74	4087.92	1.56×	3668.53	1.60×
	GRPO$_{AC}$	62.7	51.21	3318.42	1.92×	2821.38	2.08×

summaries from the agent model prompt and leave the other components intact. **Qwen3-4B (base)** uses off-the-shelf model as memory model. Finally, **GRPO**$_{AC}$ is a model trained using our proposed framework in Section 3.4. For `DeepSWE` and `Qwen3-Coder-Max`, we use $k = 2$, and for `GLM-4.7` we use $k = 4$. For latency comparison, we start two different vLLM engines for both agent model and memory model. We consistently choose vLLM version `0.14.0` with prefix caching and chunked prefilling. We further compare latency under two scenarios: with or without cpu offloading, to demonstrate the capability of CoMem on wide deployment settings. For `DeepSWE`, we run on A100 (80GB), and for the other two models, we run on H200. Notice that in production, one memory model server can be used for multiple agent models. See further discussions in Section 6.

4.3 Main Results

We evaluate CoMem on SWE-bench Verified across three agent backbones: `DeepSWE` (32B), `Qwen3-Coder-Max` (480B), and `GLM-4.7` (355B). In terms of effectiveness, our framework consistently outperforms standard compression baselines. Notably, on the `DeepSWE` backbone, CoMem (GRPO) achieves a 41.0% resolution rate, slightly surpassing the full-context baseline (40.4%), suggesting that aligned summarization can effectively filter irrelevant noise for mid-sized models. For larger models like `Qwen3-Coder-Max` and `GLM-4.7`, our reward-driven training significantly recovers performance compared to base and SFT variants, closing the gap with the full-context upper bound while restoring tool-use frequency. In terms of efficiency, the decoupled architecture delivers robust inference speedups ranging from 1.45× to 2.08× across different hardware configurations. The gains are most pronounced on GLM-4.7 (up to 2.08×), confirming that CoMem successfully mitigates the decoding bottleneck without compromising the agent's ability to solve complex, long-horizon tasks.

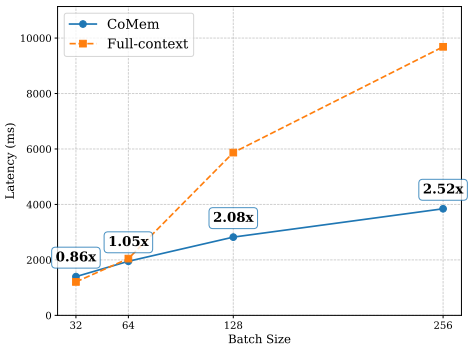

Figure 5: Latency and Speedup results for `GLM-4.7` over various batch sizes. CoMem's speedup scales up with throughput.

4.4 Scalable Speedup Gains with Increased Batch Size

We evaluate the inference latency of CoMem against a Full-context baseline across batch sizes ranging from 32 to 256. While the baseline performs competitively at the smallest scale, it exhibits

poor scaling as batch size increases, with latency rising from 1206.60 ms to 9684.24 ms. In contrast, COMEM demonstrates superior computational efficiency and linear scaling; it achieves a $2.08\times$ speedup at a batch size of 128 and a $2.52\times$ speedup at 256, effectively mitigating the memory-bandwidth bottlenecks typical of large-batch long-context processing.

5 RELATED WORKS

Recent advancements in scaling large language models (LLMs) to long-horizon tasks have largely focused on overcoming fixed context window constraints through dynamic context management and memory optimization. One primary approach involves context compression and summarization, where methods like ReSum Wu et al. (2025b), ACON Kang et al. (2025), and SUPO Lu et al. (2025) employ reinforcement learning to condense interaction histories into dense reasoning states, effectively discarding redundant information while retaining critical evidence. A parallel paradigm explores context folding and Markovian state reconstruction, illustrated by AgentFold Ye et al. (2025), Context-Folding Sun et al. (2025), and The Markovian ThinkerAghajohari et al. (2025). These frameworks restructure linear history into branching or collapsible sub-trajectories, enforcing Markovian properties to decouple reasoning depth from input length. Finally, approaches such as Mem-α Wang et al. (2025), MEM1 Zhou et al. (2025), and DeepMiner Tang et al. (2025) transition from passive context maintenance to active memory construction, training agents via RL to proactively update, retrieve, and synergize external memory stores with reasoning processes, thereby enabling sustained performance over indefinite horizons without linear computational scaling.

6 DISCUSSION

Relationship with Sparse Attention. A prominent line of research for efficient long-context processing involves sparse attention mechanisms Child et al. (2019); Beltagy et al. (2020); Zaheer et al. (2020), which mitigate the quadratic computational complexity of standard self-attention by limiting token interactions to predefined patterns or dynamic selections. More recent approaches, such as H_2O Zhang et al. (2023) and StreamingLLM Xiao et al. (2023), further optimize inference by identifying "heavy hitter" tokens or utilizing attention sinks to prune the KV cache significantly. We emphasize that COMEM is orthogonal and complementary to these architectural optimizations. Sparse attention techniques operate at the *intra-model* level, optimizing how a single model attends to its context. In contrast, COMEM operates at the *system* level by decoupling the context management process entirely. Consequently, these approaches can be combined synergistically; for instance, the underlying Memory Model in COMEM could employ sparse attention or KV cache eviction policies to process raw interaction histories with even greater efficiency, while the Agent Model continues to benefit from the high-level, compressed state representations provided by our framework.

Practical Deployment and Model Synergy. For real-world applications, COMEM enables a flexible and modular deployment strategy. We envision the memory model operating as a distinct high-throughput service, capable of handling context compression for multiple concurrent agent processes. This shared service architecture leverages the inherent efficiency of smaller models to offload context management, allowing for independent scaling of memory and reasoning components without necessitating specialized heterogeneous hardware. Consequently, this deployment paradigm resolves the tension between large and small model utilization.

7 CONCLUSION

We proposed COMEM, a framework that decouples memory management from reasoning via a novel k-step-off asynchronous pipeline. Experiments on SWE-bench Verified confirm that COMEM achieves up to $2.1\times$ inference speedup while maintaining full-context performance through our reward-driven alignment training. This work effectively mitigates the latency bottleneck in long-horizon tasks, offering a robust and scalable architecture for future agentic systems.

REFERENCES

Milad Aghajohari, Kamran Chitsaz, Amirhossein Kazemnejad, Sarath Chandar, Alessandro Sordoni, Aaron Courville, and Siva Reddy. The markovian thinker. *arXiv e-prints*, pp. arXiv–2510, 2025.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Zichuan Fu, Wentao Song, Yejing Wang, Xian Wu, Yefeng Zheng, Yingying Zhang, Derong Xu, Xuetao Wei, Tong Xu, and Xiangyu Zhao. Sliding window attention training for efficient large language models, 2025. URL https://arxiv.org/abs/2502.18845.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024. URL https://arxiv.org/abs/2401.13919.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2025. URL https://arxiv.org/abs/2401.18079.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Z Morley Mao. Compute or load kv cache? why not both? *arXiv preprint arXiv:2410.03065*, 2024.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. Acon: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023. URL https://arxiv.org/abs/2309.06180.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Scaling llm multi-turn rl with end-to-end summarization-based context management. *arXiv preprint arXiv:2510.06727*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Yaorui Shi, Yuxin Chen, Siyuan Wang, Sihang Li, Hengxing Cai, Qi Gu, Xiang Wang, and An Zhang. Look back to reason forward: Revisitable memory for long-context llm agents, 2026. URL https://arxiv.org/abs/2509.23040.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon llm agent via context-folding. *arXiv preprint arXiv:2510.11967*, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Qiaoyu Tang, Hao Xiang, Le Yu, Bowen Yu, Yaojie Lu, Xianpei Han, Le Sun, WenJuan Zhang, Pengbo Wang, Shixuan Liu, et al. Beyond turn limits: Training deep search agents with dynamic context window. *arXiv preprint arXiv:2510.08276*, 2025.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, et al. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025a. URL https://arxiv.org/abs/2410.10813.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, et al. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025b.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, et al. Agentfold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

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.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. *arXiv preprint arXiv:2506.15841*, 2025.

A Prompt for Agent Model

```
Consider the following github issue:
<github_issue>
{problem_statement}
</github_issue>
```

```
Can you help me implement the necessary changes to the repository
to fix the <github_issue>? Please refer to the summary and recent
messages for context, which contains our previous conversations
and what you have done so far.

Here are the summary of our previous conversations:
<summary>
{summary}
</summary>
Here are the most rencent messages exchanged that are not included
in the summary:
<recent_messages>
{recent_messages}
</recent_messages>
```

B PROMPT FOR MEMORY MODEL

```
You are a helpful assistant that summarizes conversations for
another model to use. You need to make sure that the summary
captures all important details from the conversation so that
the other model can understand the context just like reading
the full conversation.

\#\#\# Conversation:\\
\{conversation\}

Please provide a detailed summary of the above conversation. Make
sure to include all necessary details for the other model to
understand the context, especially entities, actions taken,
and outcomes. But NEVER include any suggestions or recommendations
for future actions-only summarize what has already occurred.
```