General Agentic Memory Via Deep Research

B.Y. Yan[1], **Chaofan Li**[1], **Hongjin Qian**[1,3], **Shuqi Lu**[1], **Zheng Liu**[1,4*]

1. Beijing Academy of Artificial Intelligence   2. Renmin University of China
3. Peking University   4. Hong Kong Polytechnic University
{chienqhj,zhengliu1026}@gmail.com

Abstract

Memory is critical for AI agents, yet the widely-adopted static memory, aiming to create readily available memory in advance, is inevitably subject to severe information loss. To address this limitation, we propose a novel framework called **general agentic memory (GAM)**. GAM follows the principle of "**just-in time (JIT) compilation**" where it focuses on creating optimized contexts for its client at runtime while keeping only simple but useful memory during the offline stage. To this end, GAM employs a duo-design with the following components. 1) **Memorizer**, which highlights key historical information using a lightweight memory, while maintaining complete historical information within a universal page-store. 2) **Researcher**, which retrieves and integrates useful information from the page-store for its online request guided by the pre-constructed memory. This design allows GAM to effectively leverage the agentic capabilities and test-time scalability of frontier large language models (LLMs), while also facilitating end-to-end performance optimization through reinforcement learning. In our experimental study, we demonstrate that GAM achieves substantial improvement on various memory-grounded task completion scenarios against existing memory systems.

1   Introduction

"Intelligence is not the ability to store information, but to know where to find it."

—Albert Einstein

AI agents become increasingly popular thanks to the rapid advancement of large language models (LLMs) [1]. Today, prototypes of AI agents are being deployed across many crucial domains, such as information seeking, software engineering, and scientific research, showcasing huge potential in improving the productivity of human society [2, 3, 4]. This widespread application, however, creates an urgent need to manage complex and rapidly expanding contexts, as AI agents must continuously integrate vast amounts of information generated by both their internal reasoning and external feedback [5]. To address this challenge, there has been growing interest in developing specialized memory systems that provide agents with essential contextual information to support downstream tasks [6]. Most existing memory systems follow the principle of **Ahead-of-Time (AOT) Compilation**. Under this paradigm, substantial computation is performed during the offline stage to compress raw contexts as lightweight memory, while incoming requests are served primarily based on this pre-constructed memory [7, 8, 9, 10]. Although widely adopted, this AOT-style approach suffers from critical limitations.

⋆ **Memorization is a form of data compression; thus, it is inevitably subject to information loss**. The precomputed memory, being a compressed representation of raw data, inevitably suffers from *information loss*, making it difficult to satisfy the fine-grained information needs requested by client
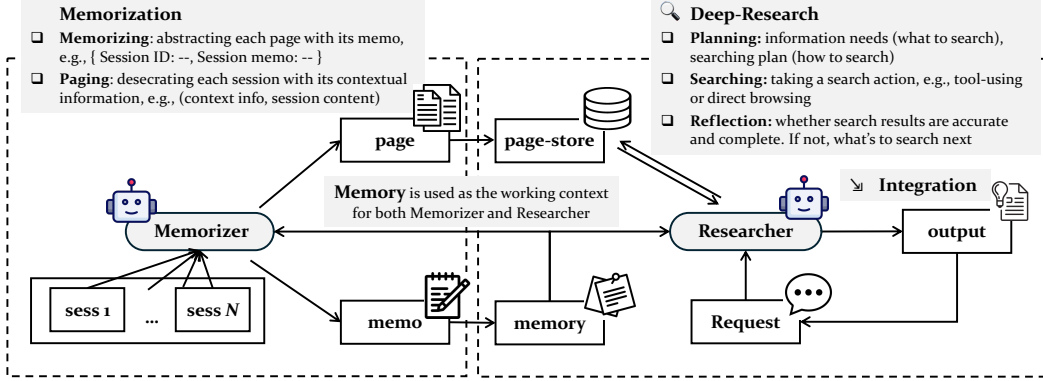
---

*Project lead

Figure 1: Overview of GAM. The memorizer generates a light memory by for agent history and keeps the complete history in the page-store during the offline stage. The researcher performs deep-research to retrieve and integrate useful information for its request in the online service.

agents. In addition, such memory systems generally assume a *static structure*, preventing them from flexibly adapting to ad-hoc or unforeseen requests that demand nuanced interpretation and integration of information. Finally, existing approaches often rely heavily on *domain expertise and handcrafted heuristics* to determine how memory is constructed and organized, which further constrains generalization across domains and tasks of the AOT-style memory systems.

⋆ **Search** *is made as the core of memory, while memorization is conducted to enable effective search.* We argue that lossless memory can only be realized via searching over a database of the complete history, where the pre-computed memory is introduced to support such a search process. With this insight, we propose **General Agentic Memory (GAM)**, a novel memory framework for general AI agents following the principle of **Just-in-Time (JIT) Compilation**. During the offline stage, it creates a light memory for the crucial historical information while maintaining the complete historical information in the database. At runtime, it performs intensive computation, namely deep research, to generate a customized, high-utility context for its request based on the pre-constructed memory.

⋆ **Dual-architecture**. Based on the above JIT principle, GAM is realized based on a dual-agent framework with two fundamental roles: the Memorizer and the Researcher (Figure 1):

- The **Memorizer** receives the client's streaming history as a sequence of sessions, where it takes two actions: 1) it dynamically compresses the key historical information with a lightweight memory, and 2) it merges each session and its corresponding memory into a page and save all pages into a page-store, ensuring that the historical information is coherently and inclusively preserved.
- The **Researcher** receives an online request from its client and performs deep research based on the pre-constructed memory to address the client's needs. It iteratively analyzes information need and plans search actions, retrieves relevant information from the page-store, and reflects on the results until the gathered information fully satisfies the client's request.

The above framework endows GAM with several key advantages. 1) *High-fidelity and task-adaptability*, enabling the generation of concise yet highly informative memory tailored to downstream tasks. 2) *Domain generalizability*, allowing GAM to operate effectively across general scenarios without relying on domain-specific expertise or handcrafted heuristics. 3) *Optimizability*, harnessing advanced LLMs' agentic capability and test-time scalability for performance optimization, while also facilitating continual improvement through reinforcement learning.

We evaluate GAM's performance through rigorous experimental studies. We jointly leverage the traditional memory benchmark LoCoMo [11], together with popular long-context benchmarks such as HotpotQA [7], RULER [12], and NarrativeQA [13]. Across all these experiments, GAM consistently and significantly outperforms existing methods, demonstrating its strong ability to preserve fine-grained historical information and to optimize downstream task-completion performance for its clients. Our project is made publicly available to facilitate future research in this field[2].

---

[2]https://github.com/VectorSpaceLab/general-agentic-memory

2 Methodology

2.1 Definition

LLM agents often require long trajectories, comprising multi-step reasoning and tool using, to accomplish complex tasks, e.g., software engineering and deep research. In our work, we define each historical trajectory (history for short) as a sequence of temporally ordered units called sessions: $\text{hist} : s_1, ..., s_T$. The rapidly growing history leads to several crucial challenges, including prohibitive computational costs, context window overflow, and performance degradation. To address these issues, a memory system is introduced to manage the information overload. Its primary objective is to extract useful yet concise information from the history, which is essential for the completion of the agent's task. That is to say, the memory system is to optimize the *cost-effectiveness* of the agent's task completion grounded on its produced context. This objective can be formulated as the following min–max optimization problem.

Definition 2.1 (**Memory**). A memory system produces the optimized context for an agent based on its task and history: $c^* \leftarrow \text{Memory}(\text{task}, \text{history})$, which is of the minimum size while optimizing the task completion performance: $c^* = \text{argmin}_{\mathcal{C}^*} |\text{c}|$, where $\mathcal{C}^* = \text{argmax}_{\mathcal{C}} \text{Agent}(\text{task}, \text{context})$.

2.2 General Agentic Memory

The overall architecture of GAM, depicted in Figure 1, consists of two main modules: the memorizer and the researcher. Both modules are LLM-based agents, each with customized prompts[3], working together to generate optimized memory that addresses requests from the client agent.

2.2.1 Memorizer

The memorizer is responsible for processing the agent's trajectory during the offline stage, ensuring that it can be efficiently stored and effectively utilized. Each memorization step is triggered by the arrival of a new session (s_i), where two operations are performed. 1. *Memorizing*, which produces memo (μ_i) as a concise and well-structured snapshot of the new session. The memo is generated based on both the new session and the existing memory (m_i), highlighting its crucial information for the entire trajectory. The memory is therefore incrementally updated with the addition of the memo:

$$\text{Memorizer.memorize}(s_i, m_i) \rightarrow \mu_i; \ m_i + \{\mu_i\} \rightarrow m_{i+1}. \tag{1}$$

2. *Paging*, which creates pages to maintain the complete information of the agent's trajectory. It begins by generating a header for the new session, which contains crucial contextual information from its preceding trajectory. The header is then used to decorate the session, forming a new page that is subsequently added to the page-store (p):

$$\text{Memorizer.page}(s_i, m_i) \rightarrow h_i; \ \{\text{header} : h_i, \ \text{content} : s_i\} \rightarrow p_i; \ p.\text{append}(p_i). \tag{2}$$

This process shares the same principle of BGE landmark retrieval [14] and Anthropic contextual retrieval [15], which preserve the consistency of page semantics, ensuring that they can be accurately retrieved in subsequent stages.

2.2.2 Researcher

The researcher is to address the client's request by retrieving and integrating useful information from the page-store. The process is iteratively conducted with three operations. 1) *Planning*, which performs a chain-of-thought reasoning based on the existing memory to analyze the underlying information needed by request (r). Based on this initial reasoning result, it further generates concrete search plans according to the provided search toolkit ($\mathcal{T}$):

$$\text{Researcher.plan}(r, m_i, \mathcal{T}) \rightarrow \{\text{tool} : t; \ \text{parameter} : \rho_t\}_{t \in \mathcal{T}}. \tag{3}$$

In our implementation, we offer three available tools for the researcher: an embedding model for vector search, a BM25 retriever for keyword-based search, and an ID-based retriever for direct page exploration. 2) *Searching*. Upon obtaining the search plan, the researcher executes each search action in parallel, retrieving relevant pages (p_t) from the page-store. The researcher then integrates the

---

[3]We include the detailed prompts of all functions in the appendix of the paper.

information from the union of the retrieved pages together with the last integration result ($\mathcal{I}$) for the request (r), leading to an updated temporal integration result:

$$\text{For each } t: t(\rho_t) \to p_t; \ \text{Researcher.integrate}(\bigcup\nolimits_{t \in \mathcal{T}} p_t, \mathcal{I}, r) \to \mathcal{I}. \tag{4}$$

3) *Reflection*. The researcher performs a reflection on whether the needed information in the request (r) has been fully collected by the integration result $\mathcal{I}$ using a binary indicator (y). If no, it further analyzes for the missing information, leading to a new request r' to drive another round of deep research. If yes, the research process will be concluded by returning the integration result:

$$\text{Researcher.reflect}(\mathcal{I}, r) \to y, r'; \ \text{if } y = \text{No, Researcher}(r', \mathcal{I}); \ \text{if } y = \text{Yes, return } \mathcal{I}. \tag{5}$$

Finally, the integrated result, along with the original information extracted from the associated pages, is returned to the client as the optimized context for its downstream task completion.

2.2.3 Optimization

A unified end-to-end performance optimization framework is introduced for GAM. Suppose a training dataset $\mathcal{D} = \{(\text{task}, \text{hist})\}$ is given, the system creates the memory and page-store as: $\text{M}, \text{P} \leftarrow \text{Memorizer}(\text{hist})$, and then generates a candidate context for the task via: $c \leftarrow \text{Researcher}(\text{task}, \text{M}, \text{P})$. Using this candidate context, the client samples an answer (ans), whose quality is measured by the reward function $\Gamma(\cdot)$. Thus, the expected reward is derived as:

$$\mathcal{R} = \mathbb{E}_{\text{task},\text{hist}\sim\mathcal{D}} \ \mathbb{E}_{\text{M},\text{P}\sim\text{Memorizer}(\text{hist})} \ \mathbb{E}_{c\sim\text{Researcher}(\text{task},\text{M},\text{P})} \ \mathbb{E}_{\text{ans}\sim\text{Client}(c,\text{task})} \ \Gamma(\text{ans}). \tag{6}$$

When focusing on optimizing GAM's performance, the memorizer and the researcher are learned via reinforcement, while the client is excluded from the learning process. Without loss of generality, the policy gradients for the memorizer and researcher are given by:

$$\begin{aligned}
\nabla_{\theta_m} &= \mathbb{E}_{\text{task},\text{hist}\sim\mathcal{D}} \ (\Gamma(\text{ans}) - \bar{\Gamma}_m)\nabla_{\theta_m} \log \pi_m(\text{M}, \text{P}|\text{hist}), \\
\nabla_{\theta_r} &= \mathbb{E}_{\text{task},\text{hist}\sim\mathcal{D}} \ (\Gamma(\text{ans}) - \bar{\Gamma}_r)\nabla_{\theta_r} \log \pi_r(c|\text{task}, \text{M}, \text{P}).
\end{aligned} \tag{7}$$

Here, θ_m and θ_r denote the model parameters of memorizer and researcher, respectively; $\bar{\Gamma}_m$ and $\bar{\Gamma}_r$ are the baseline answer rewards of the two modules; while $\theta_m(\cdot)$ and $\theta_r(\cdot)$ stand for the memorizer and researcher's generation likelihood.

3 Experiment

In this section, we conduct comprehensive experimental studies to evaluate the effectiveness of GAM. We focus on the investigation of the following three research questions. **RQ 1**: How does GAM perform compared with existing memory systems? **RQ 2**: How does GAM's performance vary across different scenarios? **RQ 3**: How do key technical factors within GAM influence its performance?

3.1 Experiment Setting

Datasets. To rigorously evaluate the effectiveness of GAM, specifically 1) the memory's ability to preserve historical information and 2) its ability to support downstream task completion, we employ the following benchmarks in our experimental studies. **1) LoCoMo** [11]. A widely used memory benchmark for conversational settings, designed to evaluate an agent's ability to maintain and recall information across extended multi-session dialogues. We adopt its single-hop, multi-hop, temporal-reasoning, and open-domain tasks in our experiments. **2) HotpotQA** [16]. A popular multi-hop question answering benchmark based on the Wikipedia corpus. We use the curated memory-evaluation dataset in MemAgent [7] that concatenates gold supporting documents with distracting passages. By varying the number of distractions, the dataset provides three versions with context lengths of 56K, 224K, and 448K tokens. **3) RULER** [12]. A popular long-context understanding benchmark with four types of evaluation tasks, including retrieval (Retri.), multi-hop tracing (MT), aggregation (AGG.), and question answering (QA). We use the 128K-token setting in our experiments. **4) NarrativeQA** [13]. A long-context question answering benchmark that provides an entire book or movie script as the input context for each sample. We randomly sample a subset of 300 questions for evaluation, whose average token length is 87K.

Baselines. We consider the following baselines in our experiments. **1) Memory-free methods**, including the *brute-force long-LLM* (long-LLM for brevity) and retrieval-augmented generation (RAG). The long-LLM baseline attempts to process the entire input within the model's context window. When the number of input tokens exceeds the maximum allowable context length L_{max}, the input is evenly partitioned into N chunks of length L_{max}: $\{S_1, ..., S_N\}$, where the final score is reported as the maximum over all chunks: $\max\{\text{LLM}(S_1)...\text{LLM}(S_N)\}$. For the RAG baseline, the input is uniformly partitioned into segments of 2,048 tokens, and the top-5 retrieved segments are used to perform the downstream task. **2) Memory-based methods**, including A-Mem [8], Mem0 [9], MemoryOS [10] and LightMem [17]. These approaches construct specialized memory structures to store historical information, which can be utilized to address memory-related tasks at runtime.

Implementation Details. In our experiments, we adopt GPT-4o-mini and Qwen2.5-14B-Instruct [18] as the backbone models for both GAM and all baselines. Both LLMs offer a long-context window of 128K tokens. We use BGE-M3 [19] as the default dense retriever. For GAM's detailed configuration, we set the maximum reflection depth to 3 and the maximum number of retrieved pages to 5. The input context is segmented into 2,048-token pages for stream processing in the memorization module.

3.2 Main Results: Overall Effectiveness

Table 1 presents the main results of GAM and baselines on the experimental benchmarks, from which the following observations can be made. First, GAM consistently outperforms all baselines, including both memory-free and memory-based approaches, across every benchmark. Moreover, its advantage becomes particularly pronounced on benchmarks like HotpotQA and RULER, where tasks require multi-step retrieval and reasoning over information dispersed within the input context. For instance, GAM achieves over 90% accuracy on the multi-hop tracing (MT) tasks in the RULER benchmark, which demand tracking variable values across multiple steps of assignment; in contrast, most baselines fail to achieve satisfactory performance under such complexity. Finally, GAM maintains stable and competitive performance under varying input-context lengths, as reflected in the results on HotpotQA. In summary, these experimental results preliminarily verify GAM's overall effectiveness and its robustness to task complexity and growing context lengths.

We obtain the following interesting things besides the main observations. First, the performance of long-LLMs is under-expectation compared with the other methods, despite that it has adopted LLMs with a 128K context window, long enough to fully cover the input context in LoCoMo, HotpotQA-56K, and NarrativeQA. This suggests that simply extending the context window is insufficient to effectively address long-context challenges. This also aligns with the recently discussed phenomenon of context rot[4], which indicates that the substantial distracting or irrelevant information within long contexts can severely degrade LLMs' performance. Second, direct applications of retrieval, i.e., RAG, exhibit highly variable performance across different scenarios. RAG improves performance over long-LLMs and the memory-based methods when the relevant information is explicitly presented, such as LoCoMo single-hop and RULER retrieval. However, it performs badly in HotpotQA, RULER multi-hop tracing, and RULER aggregation tasks, where relevant information is unobvious. In comparison, the memory-based methods show lower variance but remain constrained due to the loss of crucial details of the original context. In contrast, GAM leverages memory to support effective retrieval of task-relevant information, enabling it to achieve substantially improved performance.

3.3 Model's Impact

Table 2 presents the performance of GAM on HotpotQA and NarrativeQA implemented with different LLMs. We apply Qwen-2.5 variants of different sizes (from 0.5B to 32B) and GPT-4o-mini as the backbones of the memorization and research module. As demonstrated by the experiment result, larger and stronger LLM-backbones for both memorizer and researcher result in consistent performance improvement, indicating that GAM can effectively leverage the increased LLM capacity to improve its memory quality. However, we also observe that the research module is considerably more sensitive to the LLM's scale than the memorization module. Notably, GAM maintains strong performance even when the memorizer is downsized and remains competitive with the smallest Qwen-2.5-0.5B model. In contrast, GAM's overall performance deteriorates significantly when the research module's backbone is reduced to 7B or smaller. This discrepancy reflects the distinct complexity of the two

---

[4]https://research.trychroma.com/context-rot

Table 1: Results from GAM and baselines (memory-free and memory-based) on LoCoMo, HotpotQA, RULER, and NarrativeQA. Two LLMs, GPT-4o-mini and Qwen-2.5-14B, are used in experiment.

(a) Results on LoCoMo.

Model	Method	LoCoMo							
		Single Hop		Multi Hop		Temporal		Open Domain	
		F1	BLEU-1	F1	BLEU-1	F1	BLEU-1	F1	BLEU-1
GPT-4o-mini	LONG-LLM	46.68	37.54	29.23	22.76	25.97	19.42	16.87	13.70
	RAG	52.45	47.94	27.50	20.13	46.07	40.35	23.23	17.94
	A-MEM	44.65	37.06	27.02	20.09	45.85	36.67	12.14	12.00
	MEM0	47.65	38.72	38.72	27.13	48.93	40.51	28.64	21.58
	MEMORYOS	48.62	42.99	35.27	25.22	41.15	30.76	20.02	16.52
	LIGHTMEM	41.79	37.83	29.78	24.80	43.71	39.72	16.89	13.92
	GAM	**57.75**	**52.10**	**42.29**	**34.44**	**59.45**	**53.11**	**33.30**	**26.97**
Qwen2.5 14b	LONG-LLM	46.05	39.56	32.08	24.46	30.51	24.45	14.89	11.41
	RAG	47.87	42.79	26.38	19.54	30.78	25.97	14.16	10.52
	A-MEM	33.75	30.04	22.09	15.28	27.19	22.05	13.49	10.74
	MEM0	42.58	35.15	31.73	24.82	28.96	26.24	15.03	11.28
	MEMORYOS	46.33	41.62	38.19	29.26	32.24	27.86	20.27	15.94
	LIGHTMEM	34.92	31.22	25.45	19.61	32.03	27.70	15.81	11.81
	GAM	**58.93**	**53.74**	**42.96**	**34.48**	**51.52**	**44.43**	**30.63**	**26.04**

(b) Results on HotpotQA, RULER, and NarrativeQA.

| Model	Method	HotpotQA			RULER(128k)				NarrativeQA
		56K	224K	448K	Retri.	MT	AGG.	QA	
		F1	F1	F1	Acc.	Acc.	Acc.	Acc.	F1
GPT-4o-mini	LONG-LLM	56.56	54.29	53.92	80.30	60.60	36.70	61.60	31.26
	RAG	52.71	51.84	54.01	94.25	0.00	35.50	55.90	25.00
	A-MEM	33.90	30.22	31.37	44.23	0.00	29.20	46.50	27.07
	MEM0	32.58	31.74	27.41	46.83	53.80	34.10	51.70	29.16
	MEMORYOS	26.47	23.10	24.16	63.10	2.40	35.60	36.90	26.70
	LIGHTMEM	40.93	35.28	30.02	27.63	36.20	34.00	52.60	17.51
	GAM	**63.22**	**64.56**	**59.81**	**97.70**	**93.20**	**42.50**	**72.50**	**36.86**
Qwen2.5 14b	LONG-LLM	49.75	46.82	43.17	70.85	80.00	15.40	45.60	29.69
	RAG	51.81	46.72	48.36	92.78	0.00	24.70	47.80	18.29
	A-MEM	27.04	25.65	22.92	39.73	0.00	25.80	40.20	25.18
	MEM0	30.12	32.44	26.55	43.03	41.20	31.50	46.10	27.80
	MEMORYOS	24.58	30.25	23.13	54.58	3.00	5.20	34.60	23.45
	LIGHTMEM	37.30	27.72	28.25	27.53	17.40	25.60	53.00	16.57
	GAM	**64.07**	**55.99**	**57.87**	**93.43**	**90.20**	**36.10**	**74.50**	**34.77**

modules: the memorizer primarily extracts salient information from the input context, which is a relatively straightforward task, whereas the researcher must conduct iterative planning, searching, and reflection, which is much more complex and thus demands greater model capacity.

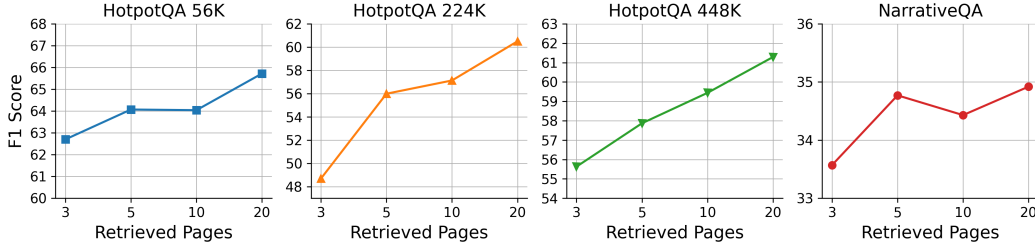## 3.4 Increasing Test-Time Computation

As shown in Figure 2, we investigate the impact of increasing test-time computation from two perspectives: 1) the depth of reflection and 2) the amount of retrieved pages. First, we vary the maximum reflection depth from 1 to 5 (3 by default), allowing GAM to perform additional research steps when necessary. Note that GAM autonomously determines the actual number of reflections

Table 2: Model's impact on memorizer (left) and researcher (right), reflected by GAM's performance.

(a) Memorizer

| Model | HotpotQA | | | NarrativeQA | Avg |
	56K F1	224K F1	448K F1	F1	F1
QWEN2.5 0.5B	56.46	55.96	53.33	29.55	48.83
QWEN2.5 3B	58.05	56.52	55.50	32.10	50.54
QWEN2.5 7B	59.06	58.34	56.17	32.53	51.53
QWEN2.5 14B	64.07	55.99	57.87	34.77	53.18
QWEN2.5 32B	63.05	59.75	56.26	34.94	53.50
GPT-4O MINI	64.77	59.29	57.25	34.87	54.05

(b) Researcher

| Model | HotpotQA | | | NarrativeQA | Avg |
	56K F1	224K F1	448K F1	F1	F1
QWEN2.5 0.5B	10.03	11.14	11.64	3.50	9.08
QWEN2.5 3B	39.76	37.16	33.04	23.96	33.48
QWEN2.5 7B	51.95	47.95	48.55	26.93	43.85
QWEN2.5 14B	64.07	55.99	57.87	34.77	53.18
QWEN2.5 32B	61.93	59.19	61.53	35.33	54.50
GPT-4O-MINI	62.06	62.97	61.54	35.24	55.45
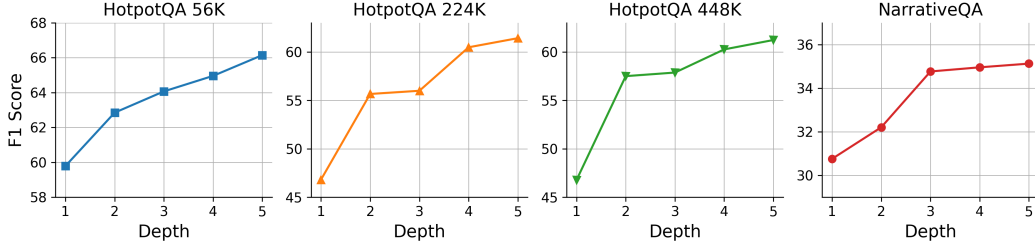
(a) Impact of maximum reflection depth.

(b) Impact of the amount of retrieved pages.

Figure 2: Impact of increasing test-time computation in reflection (top) and retrieval (bottom).

and does not always reach the maximum step. This increased flexibility enables GAM to collect more information from the page-store, thus yielding consistent performance improvements across all datasets. However, the marginal gains gradually diminish, as many tasks do not require deep multi-step reasoning. Second, we increase the number of retrieved pages from 3 to 20 (5 by default), enabling GAM to browse more pages in each step of research. The increase in retrieval results also leads to consistent performance improvements. Overall, both forms of increased test-time computation result in steady performance gains, which demonstrates GAM's ability to benefit from test-time scaling, an advantage that baseline methods lack due to their fixed workflows.

3.5 Detailed Factors' Analysis

We perform ablation studies to analyze other detailed influential factors, including searching tools, formation of GAM, and output formats.

First, we examine the impact of each searching tool and its combinations. As shown in Table 3, combining any two of the searching tools yields better performance than using each single tool alone, and the joint use of all three tools (i.e., GAM with the default setting) achieves the best performance. This observation validates the effectiveness of the search tools. Moreover, employing multiple tools enables broader exploration of the page-store, leading to better coverage of relevant information and, consequently, improved performance.

Table 3: Ablation study of detailed factors.

Method	HotpotQA			NarrativeQA	Avg
	56K	224K	448K		
	F1	F1	F1	F1	F1
GAM	64.07	55.99	57.87	34.77	53.18
Tools					
ONLY PAGE-ID	44.86	21.65	19.02	30.30	28.96
ONLY EMBEDDING	39.59	32.71	26.67	30.25	32.31
ONLY BM25	59.24	52.29	51.52	31.50	48.64
EMBEDDING+PAGE-ID	47.25	34.78	28.43	33.41	35.97
EMBEDDING+BM25	61.37	55.00	54.90	33.20	51.12
BM25+PAGE-ID	63.57	55.38	55.62	32.05	51.66
Modules					
RESEARCH WITHOUT MEMORY	57.40	49.72	53.98	31.97	48.27
MEMORY WITHOUT RESEARCH	42.67	19.75	17.38	30.18	27.50

Table 4: Performance across different output formats.

Model	Metric	HotpotQA			NarrativeQA	Avg
		56K	224K	448K		
INTEGRATION ONLY	F1	64.07	55.99	57.87	34.77	53.18
	Tokens	103.42	102.55	109.98	107.64	105.90
INTEGRATION WITH PAGE	F1	68.66	59.77	59.42	34.99	55.71
	Tokens	1444.30	499.23	620.11	6955.62	2379.82
INTEGRATION WITH EXTRACTION	F1	67.41	57.83	57.81	34.82	54.47
	Tokens	220.78	227.57	230.47	244.20	230.76

Second, we evaluate GAM's performance when each module is used in isolation, namely 1) research without memory, and 2) memory without research. According to the experiment result in Table 3, using the research module alone leads to a substantial performance drop compared with the complete GAM system, highlighting the crucial role of memory in supporting effective exploration of relevant information. Using the memory module alone results in even worse performance, indicating that the pre-computed memory is prone to severe information loss. This observation further echoes our previous conclusion that the pre-constructed memory used in traditional ahead-of-time paradigms is far more limited than the just-in-time approach adopted by GAM.

Third, we explore the impact of different forms of output, including 1) the researcher's integration result (default), 2) the integration result accompanied by the relevant pages that provided its source information, and 3) the integration result paired with extracted source snippets from those relevant pages. As shown in Table 4, using only the integration result already achieves highly competitive performance. However, augmenting it with source information from the relevant pages yields further improvements, as it helps mitigate the loss of fine-grained details that may occur during integration.

3.6 Efficiency

To assess the working efficiency of GAM, we measure the average time consumption, including both offline memory construction and online serving, when processing HotpotQA tasks under the 56K, 224K, and 448K settings. As shown in Table 5, GAM incurs a time cost comparable to Mem0 and MemoryOS, and is substantially faster than A-mem. All methods exhibit approximately linear growth in offline construction time as context length increases, while maintaining relatively stable online serving time. Overall, GAM delivers strong performance with competitive efficiency, offering the best cost-effectiveness among experimental approaches.

Table 5: Efficiency analysis on HotpotQA

Dataset	Metric	A-mem	Mem0	MemoryOS	LightMem	GAM
HotpotQA 56k	OFFLINE BUILD (s)	209.74	37.42	80.36	4.93	56.89
	ONLINE SERVE (s)	0.52	0.15	0.44	0.20	12.43
	TOTAL (s)	210.26	37.57	80.80	5.13	69.32
	ANSWER QUALITY (F1)	27.04	30.12	24.58	37.30	64.07
HotpotQA 224k	OFFLINE BUILD (s)	904.99	165.30	325.70	16.61	252.72
	ONLINE SERVE (s)	0.48	0.17	0.55	0.25	16.65
	TOTAL (s)	905.46	165.47	326.25	16.86	269.37
	ANSWER QUALITY (F1)	25.65	32.44	30.25	27.72	55.99
HotpotQA 448k	OFFLINE BUILD (s)	1796.82	274.87	702.72	40.56	557.16
	ONLINE SERVE (s)	0.47	0.18	0.46	0.21	18.49
	TOTAL (s)	1797.29	275.05	703.18	40.78	575.65
	ANSWER QUALITY (F1)	22.92	26.55	23.13	28.25	57.87

4 Conclusion

In this paper, we present a novel memory system called General Agentic Memory (GAM), which is developed under the just-in-time compilation principle. GAM employs a dual-framework comprising a memorizer and a researcher. During the offline stage, the memorizer extracts the key information for its incoming context with lightweight memory and preserve the complete information within a page-store. For each online request, the researcher performs deep-research over the page-store based on the pre-constructed memory, which generates concise yet informative memory to support the downstream task. We perform comprehensive empirical studies using a variety of popular memory and long-context benchmarks, whose result validates the effectiveness of GAM given its significant and consistent improvements over existing methods.

References

[1] Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, et al. Exploring large language model based intelligent agents: Definitions, methods, and prospects. *arXiv preprint arXiv:2401.03428*, 2024.

[2] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

[3] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. Magis: Llm-based multi-agent framework for github issue resolution. *Advances in Neural Information Processing Systems*, 37:51963–51993, 2024.

[4] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

[5] Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

[6] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

[7] 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.

[8] Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[9] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[10] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. *arXiv preprint arXiv:2506.06326*, 2025.

[11] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

[12] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[13] Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[14] Kun Luo, Zheng Liu, Shitao Xiao, Tong Zhou, Yubo Chen, Jun Zhao, and Kang Liu. Landmark embedding: a chunking-free embedding method for retrieval augmented long-context large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3268–3281, 2024.

[15] Anthropic. Introducing contextual retrieval. *https://www.anthropic.com/engineering/contextual-retrieval*, 2024.

[16] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

[17] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*, 2025.

[18] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[19] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 2024.

Appendix

Baseline Reproduction Details

When reproducing the baseline methods on the LocoMo dataset, we found that the category labels used for A-mem, Mem0, and MemoryOS were incorrect. Based on the official LocoMo annotations, we corrected the corresponding category–label mapping.

Prompts

Prompt for Memorizing

You are the MemoryAgent. Your job is to write one concise abstract that can be stored as long-term memory.

MAIN OBJECTIVE:
Generate a concise, self-contained and coherent abstract of INPUT_MESSAGE that preserves ALL important information in INPUT_MESSAGE.
MEMORY_CONTEXT is provided so you can understand the broader situation such as people, modules, decisions, ongoing tasks and keep wording consistent.

INPUTS:
MEMORY_CONTEXT: {memory_context}
INPUT_MESSAGE: {input_message}

YOUR TASK:
1. Read INPUT_MESSAGE and extract all specific, memory-relevant information, such as:
   - plans, goals, decisions, requests, preferences
   - actions taken, next steps, assignments, and responsibilities
   - problems, blockers, bugs, questions that need follow-up
   - specific facts such as names, dates, numbers, locations
2. Use MEMORY_CONTEXT to:
   - resolve or disambiguate the entities, components, tasks, or resources mentioned in INPUT_MESSAGE,
   - keep terminology (names of agents, modules, datasets, etc.) consistent with prior usage,
   - include minimal background context if it is required for the abstract to be understandable.
   You MUST NOT invent or add information that appears only in MEMORY_CONTEXT and is NOT implied or mentioned in INPUT_MESSAGE.
3. Your abstract MUST:
   - summarize all important content from INPUT_MESSAGE,
   - be understandable on its own without seeing INPUT_MESSAGE,
   - be factual and specific.

STYLE RULES:
- Output exactly ONE concise paragraph. No bullet points.
- Do NOT include meta phrases like "The user said..." or "The conversation is about...".
- Do NOT give advice, opinions, or suggestions.
- Do NOT ask questions.
- Do NOT include anything that is not grounded in INPUT_MESSAGE.

OUTPUT FORMAT:
Return ONLY the single paragraph. Do NOT add any headings or labels.

Prompt for Planning Part 1

You are the PlanningAgent. Your job is to generate a concrete retrieval plan for how to gather information needed to answer the QUESTION.
You must use the QUESTION and the current MEMORY (which contains abstracts of all messages so far).
QUESTION: {request}
MEMORY: {memory}

PLANNING PROCEDURE
1. Interpret the QUESTION using the context in MEMORY. Identify what is need to satisfy the QUESTION.
2. Break that need into concrete "info needs": specific sub-questions you must answer to fully respond to the QUESTION.
3. For each info need, decide which retrieval tools are useful. You may assign multiple tools to the same info need:
   - Use "keyword" for exact entities / functions / key attributes.
   - Use "vector" for conceptual understanding.
   - Use "page_index" if MEMORY already points to clearly relevant page indices.
4. Build the final plan:
   - "info_needs": a list of all the specific sub-questions / missing facts you still need.
   - "tools": which of ["keyword","vector","page_index"] you will actually use in this plan. This can include more than one tool.
   - "keyword_collection": a list of short keyword-style queries you will issue.
   - "vector_queries": a list of semantic / natural-language queries you will issue.
   - "page_index": a list of integer page indices you plan to read fully.

AVAILABLE RETRIEVAL TOOLS:
All of the following retrieval tools are available to you. You may select one, several, or all of them in the same plan to maximize coverage. Parallel use of multiple tools is allowed and encouraged if it helps answer the QUESTION.
1. "keyword"
   - WHAT IT DOES:
     Exact keyword match retrieval.
     It finds pages that contain specific names, function names, key attributes, etc.
   - HOW TO USE:
     Provide short, high-signal keywords.
     Do NOT write long natural-language questions here. Use crisp keywords and phrases that should literally appear in relevant text.
2. "vector"
   - WHAT IT DOES:
     Semantic retrieval by meaning.
     It finds conceptually related pages.
     This is good for high-level questions, reasoning questions, or "how/why" style questions.
   - HOW TO USE:
     Write each query as a short natural-language sentence that clearly states what you want to know, using full context and entities from MEMORY and QUESTION.
     Example style: "How does the DenseRetriever assign GPUs during index building?"

3. "page_index"
   - WHAT IT DOES:
     Directly ask to re-read full pages (by page ID) that are already known to be relevant.
     MEMORY may mention specific page IDs or indices that correspond to important configs, attributes, or names.
     Use this if you already know specific page indices that should be inspected in full.
   - HOW TO USE:
     Return a list of those integer page indices (e.g. [0, 2, 5]), max 5 pages.
     You MUST NOT invent or guess page indices.

RULES
- Avoid simple repetition. Whether it's keywords or sentences for search, make them as independent as possible rather than duplicated.
- Be specific. Avoid vague items like "get more details" or "research background".
- Every string in "keyword_collection" and "vector_queries" must be directly usable as a retrieval query.
- You may include multiple tools. Do NOT limit yourself to a single tool if more than one is useful.
- Do NOT invent tools. Only use "keyword", "vector", "page_index".
- Do NOT invent page indices. If you are not sure about a page index, return [].
- You are only planning retrieval. Do NOT answer the QUESTION here.

THINKING STEP
- Before producing the output, think through the procedure and choices inside <think>...</think>.
- Keep the <think> concise but sufficient to validate decisions.
- After </think>, output ONLY the JSON object specified below. The <think> section must NOT be included in the JSON.

OUTPUT JSON SPEC
Return ONE JSON object with EXACTLY these keys:
- "info_needs": array of strings (required)
- "tools": array of strings from ["keyword","vector","page_index"] (required)
- "keyword_collection": array of strings (required)
- "vector_queries": array of strings (required)
- "page_index": array of integers (required), max 5.

All keys MUST appear.
After the <think> section, return ONLY the JSON object. Do NOT include any commentary or explanation outside the JSON.

Prompt for Integrating Part 1

You are the IntegrateAgent. Your job is to build an integrated factual summary for a QUESTION.

YOU ARE GIVEN:
- QUESTION: what must be answered.
- EVIDENCE_CONTEXT: newly retrieved supporting evidence that may contain facts relevant to the QUESTION.
- RESULT: the current working notes / draft summary about this same QUESTION (may be incomplete).

YOUR OBJECTIVE:
Produce an UPDATED_RESULT that is a consolidated factual summary of all information that is relevant to the QUESTION.
This is NOT a final answer to the QUESTION. It is an integrated summary of all useful facts that could be used to answer the QUESTION.

The UPDATED_RESULT must:
1. Keep useful, correct, on-topic information from RESULT.
2. Add any new, relevant, well-supported facts from EVIDENCE_CONTEXT.
3. Remove anything that is off-topic for the QUESTION.

QUESTION: {question}
EVIDENCE_CONTEXT: {evidence_context}
RESULT: {result}

INSTRUCTIONS:
1. Understand the QUESTION. Identify exactly what needs to be answered.
2. From RESULT:
   - Keep any statements that are relevant to the QUESTION.
3. From EVIDENCE_CONTEXT:
   - Extract every fact that helps describe, clarify, or support an answer to the QUESTION.
   - Prefer concrete details such as entities, numbers, versions, decisions, timelines, outcomes, responsibilities, constraints.
   - Ignore anything unrelated to the QUESTION.
4. Synthesis:
   - Merge the selected content from RESULT with the selected content from EVIDENCE_CONTEXT.
   - The merged text MUST read as one coherent factual summary related to the QUESTION (not the direct answer).
   - The merged summary MUST collect all important factual information needed to answer the QUESTION, so it can stand alone later without needing RESULT or EVIDENCE_CONTEXT.
   - Do NOT add interpretation, recommendations, or conclusions beyond what is explicitly stated in RESULT or EVIDENCE_CONTEXT.

Prompt for Integrating Part 2

RULES:
- "content" MUST ONLY include factual information that is relevant to the QUESTION.
- You are NOT producing a final answer, decision, recommendation, or plan. You are producing a cleaned, merged factual summary.
- Do NOT invent or infer facts that do not appear in RESULT or EVIDENCE_CONTEXT.
- Do NOT include meta language (e.g. "the evidence says", "according to RESULT", "the model stated").
- Do NOT include instructions, reasoning steps, or analysis of your own process.
- Do NOT include any keys other than "content" and "sources".
- "sources" should on incluede the page_ids of the pages that supported the included facts.

THINKING STEP
- Before producing the output, think about selection and synthesis steps inside <think>...</think>.
- Keep the <think> concise but sufficient to ensure correctness and relevance.
- After </think>, output ONLY the JSON object. The <think> section must NOT be included in the JSON.

OUTPUT JSON SPEC:
Return ONE JSON object with EXACTLY:
- "content": string. This is the UPDATED_RESULT, i.e. the integrated final information related to the QUESTION, if there not exist any useful information, just provide "".
- "sources": array of strings/objects.

Both keys MUST be present.
After the <think> section, return ONLY the JSON object. Do NOT output Markdown, comments, headings, or explanations outside the JSON.

Prompt for InfoCheck

You are the InfoCheckAgent. Your job is to judge whether the currently collected information is sufficient to answer a specific QUESTION.

YOU ARE GIVEN:
- REQUEST: the QUESTION that needs to be answered.
- RESULT: the current integrated factual summary about that QUESTION. RESULT is intended to contain all useful known information so far.

YOUR OBJECTIVE:
Decide whether RESULT already contains all of the information needed to fully answer REQUEST with specific, concrete details.
You are NOT answering REQUEST. You are only judging completeness.

REQUEST: {request}
RESULT: {result}

EVALUATION PROCEDURE:
1. Decompose REQUEST:
   - Identify the key pieces of information that are required to answer REQUEST completely (facts, entities, steps, reasoning, comparisons, constraints, timelines, outcomes, etc.).
2. Check RESULT:
   - For each required piece, check whether RESULT already provides that information clearly and specifically.
   - RESULT must be specific enough that someone could now write a final answer directly from it without needing further retrieval.
3. Decide completeness:
   - "enough" = true ONLY IF RESULT covers all required pieces with sufficient clarity and specificity.
   - "enough" = false otherwise.

THINKING STEP
- Before producing the output, perform your decomposition and evaluation inside <think>...</think>.
- Keep the <think> concise but ensure it verifies completeness rigorously.
- After </think>, output ONLY the JSON object with the key specified below. The <think> section must NOT be included in the JSON.

OUTPUT REQUIREMENTS:
Return ONE JSON object with EXACTLY this key:
- "enough": boolean. true if RESULT is sufficient to answer REQUEST fully; false otherwise.

RULES:
- Do NOT invent facts.
- Do NOT answer REQUEST.
- Do NOT include any explanation, reasoning, or extra keys.
- After the <think> section, return ONLY the JSON object.

Prompt for Requests Generating

You are the FollowUpRequestAgent. Your job is to propose targeted follow-up retrieval questions for missing information.

YOU ARE GIVEN:
- REQUEST: the original QUESTION that we ultimately want to be able to answer.
- RESULT: the current integrated factual summary about this QUESTION. RESULT represents everything we know so far.

YOUR OBJECTIVE:
Identify what important information is still missing from RESULT in order to fully answer REQUEST, and generate focused retrieval questions that would fill those gaps.

REQUEST: {request}
RESULT: {result}

INSTRUCTIONS:
1. Read REQUEST and determine what information is required to answer it completely (facts, numbers, definitions, procedures, timelines, responsibilities, comparisons, outcomes, constraints, etc.).
2. Read RESULT and determine which of those required pieces are still missing, unclear, or underspecified.
3. For each missing piece, generate ONE standalone retrieval question that would directly obtain that missing information.
   - Each question MUST:
     - mention concrete entities / modules / components / datasets / events if they are known,
     - ask for factual information that could realistically be found by retrieval (not "analyze", "think", "infer", or "judge").
4. Rank the questions from most critical missing information to least critical.
5. Produce at most 5 questions.

THINKING STEP
- Before producing the output, reason about gaps and prioritize inside <think>...</think>.
- Keep the <think> concise but ensure prioritization makes sense.
- After </think>, output ONLY the JSON object specified below. The <think> section must NOT be included in the JSON.

OUTPUT FORMAT:
Return ONE JSON object with EXACTLY this key:
- "new_requests": array of strings (0 to 5 items). Each string is one retrieval question.

RULES:
- Do NOT include any extra keys besides "new_requests".
- After the <think> section, do NOT include explanations, reasoning steps, or Markdown outside the JSON.
- Do NOT generate vague requests like "Get more info".
- Do NOT answer REQUEST yourself.
- Do NOT invent facts that are not asked by REQUEST.
After the <think> section, return ONLY the JSON object.