SelfMem: Self-Optimizing Memory for AI Agents

Anonymous ACL submission

Abstract

While current AI agents support increasingly long context windows, tool use, and skill execution for long-horizon tasks, they still require memory systems to effectively leverage historical experience. Existing memory frameworks typically rely on *fixed* storage, retrieval, and summarization mechanisms, which can be rigid across different tasks and often require manual tuning. To address this limitation, we propose SelfMem, a self-optimizing memory framework. Inspired by prior work on self-improving AI, we follow the principle of "teaching an agent to fish rather than giving it a fish." Instead of forcing the model to follow a predefined memory strategy or format, SelfMem provides an environment with memory tools and feedback signals that allow the agent to explore, evaluate, and refine its own memory strategy. Our results show that SelfMem consistently outperforms retrieval, compression, and agent-memory baselines on BEAM across conversation scales from 100K to 1M tokens. Compared with the strongest baseline, SelfMem improves the official score by 48.7%, 40.8%, and 41.9% at 100K, 500K, and 1M, respectively. Further question-type analysis shows broad robustness across diverse memory demands, and our optimization study shows that model-guided strategy refinement further improves performance.

1 Introduction

With the growing ability to process longer contexts (Bai et al., 2024), perform multi-step planning (Li et al., 2025), use external tools (Schick et al., 2023), and incorporate environmental feedback (Yao et al., 2023; Shinn et al., 2023; Zhou et al., 2024), AI agents have demonstrated strong capabilities across increasingly diverse long-horizon tasks (Liu et al., 2024b; Merrill et al., 2026). However, these advances also place greater demands on agents' ability to remember user preferences, project backgrounds, and important factual details over extended interactions, making effective memory-system support necessary (Liu et al., 2024a; Packer et al., 2023; Zhong et al., 2024; Tavakoli et al., 2025).

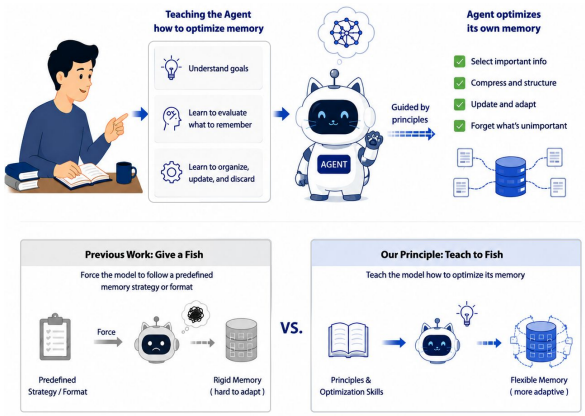

Figure 1: "Teach agent to fish rather than give it a fish": guiding agents to self-optimize memory by learning principles and trade-offs, rather than relying on rigid, predefined memory strategies.

Existing agent-memory methods, such as LoCoMo, retrieval-augmented generation (RAG), MemGPT, and MemoryBank, typically rely on manually specified strategies to construct, update, retrieve, and present historical knowledge to the model (Maharana et al., 2024; Packer et al., 2023; Zhong et al., 2024). Although these methods improve agents' ability to use past conversation histories, their memory structures and operations are still largely predefined and limit their ability in broader tasks. A-Mem (Xu et al., 2026) takes a more agentic step by allowing agents to write, index, retrieve, and link memories themselves, following the Zettelkasten principle of maintaining an evolving knowledge network. However, the core challenge remains: different tasks and conversation histories require different memory behaviors, yet it is still unclear how agents should adapt their memory strategies to task demands and environmental

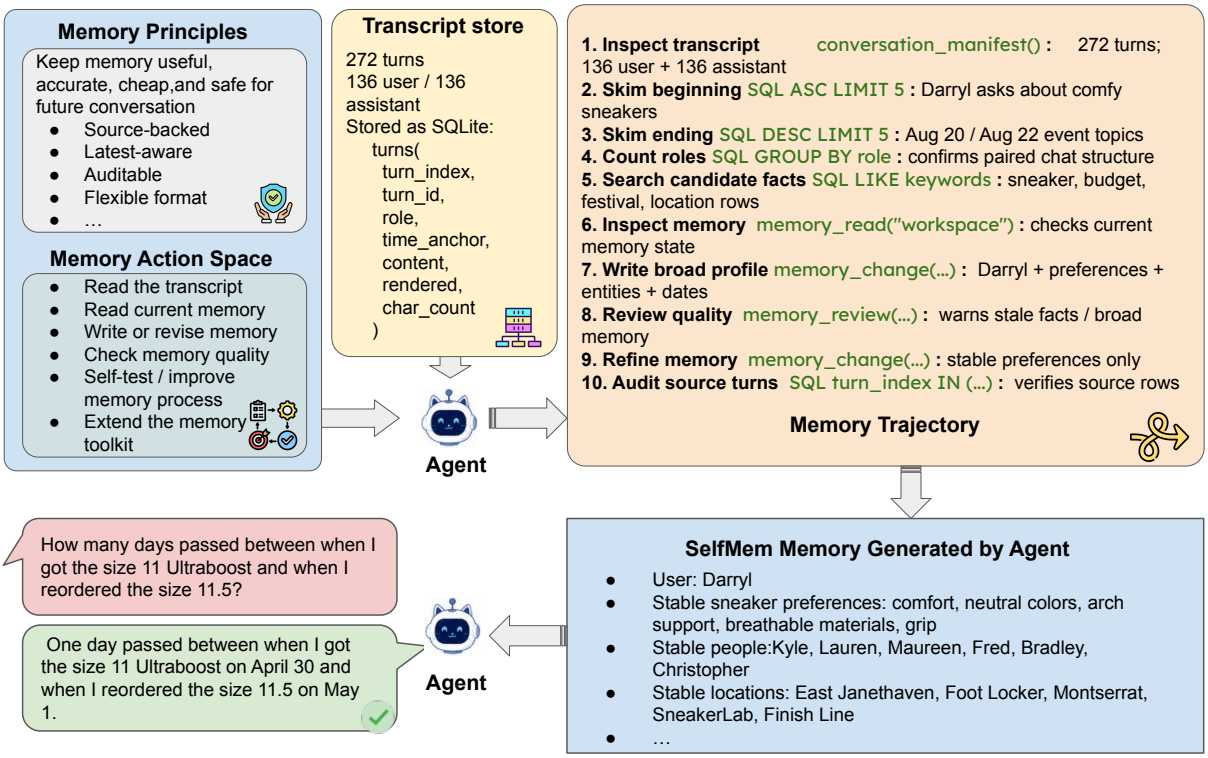

Figure 2: Overview of SelfMem. The raw transcript is kept as an immutable source of truth, while the agent is given a memory action space for inspecting the transcript, writing and revising memory, and reviewing memory quality.

feedback. As illustrated in Figure 1, this motivates our method, SelfMem: instead of giving agents a fixed memory mechanism, we teach them how to manage memory adaptively by exposing memory-management decisions to the agents themselves.

As shown in Figure 2, SelfMem turns memory management into an agent-controlled optimization process. Rather than prescribing a fixed storage, update, or retrieval rule, SelfMem exposes memory tools together with feedback signals such as response performance, token counts, cost, and cache reuse. Using these signals, the agent decides what to keep in memory, what to compress, what to update, and what to leave for transcript retrieval. Empirically, our method yields a stronger quality–cost trade-off than fixed memory pipelines.

Our results show that SelfMem delivers robust memory performance on BEAM across long conversation histories, diverse question types, and all evaluated token scales. SelfMem achieves the highest official score and $\text{Pass}_{0.5}$ at 100K, 500K, and 1M tokens. Compared with the strongest non-SelfMem baseline, it improves the official score by 0.165, 0.141, and 0.134, respectively, and increases $\text{Pass}_{0.5}$ by 17.0, 14.9, and 14.3 percentage points. Beyond aggregate performance, question-type analysis shows that SelfMem remains broadly robust across different memory demands, achieving the best score on 9/10 question types at 100K, 8/10 at 500K, and 7/10 at 1M. Our optimization and generalization analysis further shows that model-guided strategy refinement can also improve memory behavior, suggesting that SelfMem exposes a useful optimization surface for long-horizon memory management.

2 Related Work

Agent Memory Recent work on agent memory equips LLM agents with external memory states for long-running interaction and personalization. Generative Agents introduced a memory-stream architecture in which agents record observations, retrieve relevant experiences, and synthesize higher-level reflections for future planning (Park et al., 2023). MemoryBank extends this direction to long-term dialogue by maintaining persistent memories and user portraits that can be recalled, updated, and reinforced over repeated interactions (Zhong et al., 2024). MemGPT formulates long-context interaction as virtual context management, where an agent moves information between core/working memory and archival memory through tool-mediated memory operations (Packer et al., 2023). ReadAgent

studies agentic reading over very long inputs by constructing gist memories and selectively looking up original passages when detailed evidence is needed (Lee et al., 2024). A parallel line of study examines how to evaluate and structure long-term conversational memory. LoCoMo introduces very long-term multi-session dialogues and evaluates agents on question answering, event summarization, and multimodal dialogue generation over extended histories (Maharana et al., 2024). LongMemEval further decomposes long-term memory evaluation into abilities such as information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention (Wu et al., 2025). More recent memory systems move toward structured and production-oriented memory construction. Mem0 extracts, consolidates, and retrieves salient conversational facts, with an enhanced graph-based variant for relational memory (Chhikara et al., 2025). A-Mem takes a more agentic step by dynamically writing, indexing, linking, and evolving memories following Zettelkasten-style organization (Xu et al., 2026). As summarized in Appendix A, Table 2, these methods differ in their memory units, retrieval mechanisms, and update behaviors, but they still largely instantiate a predefined memory architecture or a fixed set of memory operations.

Self-improving Agent A complementary line of work shows that language models and agents can improve their behavior through feedback, reflection, and search, often without updating the underlying model parameters. Self-Refine uses the same model to generate feedback and iteratively refine its own outputs (Madaan et al., 2023). Reflexion converts task feedback into verbal reflections stored in episodic memory, allowing agents to improve across future trials through language rather than weight updates (Shinn et al., 2023). Voyager demonstrates that an embodied agent can use environmental feedback, execution errors, and self-verification to build an expanding library of reusable skills (Wang et al., 2023). LATS further combines reasoning, acting, planning, and environmental feedback through tree search, enabling agents to explore and evaluate alternative action trajectories before committing to a decision (Zhou et al., 2024). Self-RAG extends this idea to retrieval-augmented generation by learning when to retrieve, generate, and critique evidence through self-reflection (Asai et al., 2024). Beyond improving a single output or action trajectory, recent work on natural-language optimization suggests that models can also search over strategies themselves. OPRO treats an LLM as a black-box optimizer that proposes new candidate instructions from previous solutions and their scores (Yang et al., 2024). Pryzant et al. (2023) uses natural-language gradients, beam search, and bandit selection to automatically improve prompts. Promptbreeder evolves both task prompts and mutation prompts through fitness feedback, showing that hand-crafted prompt strategies can be improved through iterative self-referential search (Fernando et al., 2024). These studies motivate our hypothesis that memory behavior can also be optimized in language: given appropriate feedback signals and an action space, the agent may discover memory strategies that are better adapted to the current task than manually specified rules.

3 SelfMem: A Self-Optimizing Memory Framework

3.1 Overview

As shown in Figure 2, SelfMem turns memory management into a model-controlled optimization process. The framework keeps the original logged conversation history (raw transcript) as an immutable source of truth, while exposing a memory workspace and a set of memory-management tools to the agent. Instead of asking the agent to fill a fixed profile schema or follow a predefined summarization rule, SelfMem allows the agent to inspect transcript evidence, decide what memory is worth writing, organize memory in a task-sensitive format, review its own memory quality, and revise memory when diagnostics indicate risk. The key design goal is to separate *framework constraints* from *memory strategy*. The framework provides human-designed procedural principles, the available memory tools, feedback channels, and audit constraints. The agent then determines the concrete memory behavior: which transcript turns or retrieved snippets to inspect, what information to store, what to compress or update, what to preserve exactly, and what to leave recoverable from the original transcript.

3.2 Problem Definition

We consider memory management for agents that operate over a fixed interaction history. Let

$$H = (t_1, t_2, \ldots, t_T)$$

denote the chronological record of previous user–agent turns, where each turn contains the speaker role, time anchor, content, and rendering metadata. Given a future query q, the agent must produce an answer y by using information from H. A simple solution is to place the entire history into the model context. However, this becomes expensive or infeasible when the interaction exceeds the context window. Another common solution is to construct a fixed summary or memory schema, but such representations may omit task-critical details, preserve stale facts, or fail to match the needs of different tasks. We therefore formulate memory management as the construction of a compact memory workspace $M^\star$ from the original interaction history, while keeping the transcript available for exact factual grounding:

$$\begin{aligned} M^\star &= \Pi_{\text{agent}}(H; \mathcal{A}, \mathcal{F}), \\ E_q &= \text{Retrieve}(q, M^\star, H), \\ y &= \text{LLM}(q, M^\star, E_q). \end{aligned}$$

Here, $\mathcal{A}$ is the memory action space available to the agent, $\mathcal{F}$ is the feedback returned by the environment, and E_q denotes the transcript snippets or turns retrieved to support the query. Existing memory systems usually instantiate Π with hand-designed storage formats, retrieval rules, summarization policies, or update operations. In contrast, SelfMem exposes this memory-management procedure to the agent itself: the agent decides which parts of the transcript to inspect, what information to store, how to organize it, and when to revise it based on tool feedback and task demands.

3.3 Transcript Store

The transcript store is the source of truth in SelfMem. Rather than exposing hidden summaries or pre-extracted facts, SelfMem stores the logged interaction history as a read-only database. In our implementation, each transcript is represented as a SQLite table:

```
turns(
  turn_index, turn_id, role, time_anchor,
  content, rendered, char_count
)
```

The memory agent can inspect this transcript only through read-only tools:

$$o_i^H = \text{Read}_H(a_i),$$

where a_i is a model-selected read action and o_i^H is the returned transcript observation. The transcript interface includes `conversation_manifest`, which reports schema and transcript statistics, and `conversation_sql_query`, which allows the agent to issue read-only SQL queries over the turns table. This design lets the agent decide which parts of the history are worth inspecting, rather than forcing a fixed chunking or summarization policy over the entire conversation.

3.4 Memory Workspace and Action Space

We provide a model-managed memory workspace M_i. The workspace can contain any structure the agent judges useful, such as a user profile, preference list, project-state note, timeline, or even compact policy. The agent interacts with the workspace through a model-selectable action space:

$$\mathcal{A} = \mathcal{A}_{\text{read}} \cup \mathcal{A}_{\text{write}} \cup \mathcal{A}_{\text{review}} \cup \mathcal{A}_{\text{probe}}.$$

At each step, the agent selects both a tool and its arguments:

$$a_i = \text{Select}_{\text{LLM}}(P_{\text{mem}}, I, M_{i-1}, o_{<i}, d_{<i}),$$

where P_{mem} is the memory-management prompt, I denotes the procedural memory principles, M_{i-1} is the current workspace, $o_{<i}$ are previous tool observations, and $d_{<i}$ are previous diagnostic signals.

The main tool families are:

Transcript-read tools. These tools inspect the immutable transcript. The agent can obtain the transcript manifest, run read-only SQL queries, search for entities or events, check recent turns, and audit source turns before or after writing memory. *Memory-read tools.* The agent can inspect the current workspace using them. This allows it to avoid duplicate writes, detect stale memory, and decide whether a new update should add, replace, merge, demote, or delete existing content. *Memory-write tools.* The agent modifies memory through these tools. Each change contains an operation type, target, content, rationale, and metadata. Supported operation styles include adding memory, replacing memory, merging duplicates, refining broad summaries, archiving stale information, recording exact facts, creating timelines, and adding retrieval hints. *Memory-review tools.* The agent can call the tools to diagnose an existing or proposed memory without mutating it. Review feedback can flag missing source support, stale exact facts, contradictions, overly broad summaries, and retrievability risk. *Self-test tools.* Optionally, the agent can create internal memory probes from the observed transcript and workspace. These probes test whether

the memory organization supports plausible future questions, without using hidden benchmark answers, rubrics, or per-question judge labels.

3.5 Feedback Signals

The feedback $\mathcal{F}$ returned by SelfMem is tool and environment feedback rather than a hidden supervision signal. At memory-construction time, the agent does not receive gold answers, benchmark rubrics, held-out labels, or per-question judge decisions. Instead, feedback comes from observable interactions with the transcript, memory workspace, and diagnostics:

$$\mathcal{F}_i = \{o_i^H, o_i^M, d_i^{\text{write}}, d_i^{\text{review}}, u_i\},$$

where o_i^H is transcript-read feedback, o_i^M is memory-read feedback, d_i^{write} is write-time diagnostic feedback, d_i^{review} is review feedback, and u_i denotes optional operational signals such as token usage, latency, cache reuse, request count, and estimated cost. These signals are deliberately not collapsed into a single scalar reward. Memory management involves trade-offs, such as storing more information can improve recall but increase context cost; compressing memory can reduce latency but lose exact details. SelfMem lets the agent reason about these trade-offs in language and revise its memory behavior accordingly.

3.6 Memory Construction and Review

Memory construction proceeds as a model-controlled tool-use loop. The workspace is initialized as

$$M_0 = \emptyset.$$

At step i, the agent observes the current workspace and previous tool outputs, then selects an action a_i. Read and review actions return observations or diagnostics, while write actions produce an update ΔM_i that is applied to the workspace:

$$M_i = \begin{cases} \text{Apply}(M_{i-1}, \Delta M_i), & a_i \in \mathcal{A}_{\text{write}}, \\ M_{i-1}, & \text{otherwise.} \end{cases}$$

Each write operation records its target, content, rationale, and metadata, making the memory process auditable. Since memory writing is not treated as one-shot summarization, the agent can review a draft or existing memory:

$$d_i = \text{Review}(M_i, f_i),$$

where f_i specifies the review focus, such as source support, exact facts, stale values, contradictions, or retrievability. The agent can then revise the workspace through another memory update. This creates an inspect–write–review–revise loop in which the agent improves the memory organization itself, rather than simply producing a final answer.

3.7 Memory-Conditioned Answering

After construction, the final workspace $M^\star$ is used as a compact guide for future answering. Given a query q, SelfMem retrieves relevant transcript turns or snippets E_q and provides them together with the memory workspace to the answer model:

$$\begin{aligned} E_q &= \text{Retrieve}(q, M^\star, H), \\ y &= \text{LLM}(q, M^\star, E_q). \end{aligned}$$

In SelfMem, the memory workspace is not populated by a hand-written rule that specifies what must be saved. Instead, the agent decides what information is worth storing based on the transcript, available memory tools, and feedback signals. It may keep durable information such as user preferences, recurring entities, and project context, while leaving exact or frequently changing details, such as dates, counts, prices, and updated decisions, to be retrieved from relevant transcript turns when needed.

Overall, SelfMem shifts memory from a fixed storage pipeline to a model-guided process: the framework provides the transcript store, tools, feedback channels, and constraints, while the agent decides the concrete memory actions and organization.

4 Experiments

4.1 Experimental Setup

Benchmark and evaluation protocol. We evaluate on the original BEAM benchmark (Tavakoli et al., 2025), which provides *long conversation histories* at 100K, 500K, and 1M token scales, together with 100 conversations and 2,000 probing questions that test memory over extended interactions, covering information extraction, temporal reasoning, multi-session reasoning, and event ordering. Questions are asked sequentially, with previous model-generated question–answer turns appended to later prompts. This simulates a multi-turn probing session, enables prompt-cache reuse, and is applied consistently to all compared methods.

Baselines. Following the methods reviewed in Section 2, we compare SelfMem with representative

memory and context-management baselines. These include RAG, which chunks, embeds, and retrieves conversation history (Lewis et al., 2020); full context, which directly places the raw conversation in the prompt when feasible; and compression, which compacts history when it approaches the context limit. We also include LoCoMo, MemoryBank, ReadAgent, MemGPT, A-Mem, and Mem0, covering persistent conversational memory, agentic reading, hierarchical memory, linked note-style memory, and production-oriented memory extraction (Maharana et al., 2024; Zhong et al., 2024; Lee et al., 2024; Packer et al., 2023; Xu et al., 2026; Chhikara et al., 2025). All baselines use the same conversations, questions, answer model, and official judge; they differ only in how the conversation history is transformed into the context given to the answer model. Appendix A provides implementation details.

Model and decoding configuration. All LLM-generation calls in memory construction, and answer generation use GPT-5.4-nano, and for BEAM score we use GPT-5.4-mini as the judge model. We use deterministic decoding with temperature 0 for answer generation and LLM-based judging. Full-context baseline prompting is evaluated only when the raw conversation fits within the model and API limits; it is marked unsupported at larger scales when the original history exceeds those limits.

Evaluation metrics. Following the BEAM evaluation protocol, *Score* is the mean official question score across all evaluated probing questions: nugget-based questions use the official LLM judge score, while event-ordering questions use normalized Kendall correlation, $\tau_{\text{norm}} = (\tau_b + 1)/2$. $\text{Pass}_{0.5}$, is a thresholded pass rate: the percentage of questions whose official score is at least 0.5. *Cost* is token-computed API cost, calculated from recorded request-level token usage and the model pricing table. *Cache hit* is the ratio of cached input tokens to total input tokens. *LLM Req.* and *Embed Req.* count LLM and embedding requests, respectively.

4.2 Results

SelfMem consistently achieves the best memory performance. Table 1 reports the main BEAM results across 100K, 500K, and 1M token scales. SelfMem achieves the best Score and $\text{Pass}_{0.5}$ at all scales, indicating that its advantage is not limited to a particular context length. Compared with fixed retrieval, compression, and structured memory baselines, SelfMem maintains stronger performance as the conversation history grows longer. This suggests that agent-controlled memory construction is better suited to long-horizon settings, where relevant information is sparse, distributed across many turns, and often mixed with outdated or less useful details. The strongest baseline is consistently RAG, which shows that simple retrieval remains a competitive strategy for long conversations. However, RAG only retrieves chunks at query time and does not build an adaptive memory representation. In contrast, SelfMem allows the agent to decide what should be stored, revised, compressed, or left for later transcript lookup. This leads to large and stable gains over the best non-SelfMem baseline, with Score improvements of 0.165, 0.141, and 0.134 at 100K, 500K, and 1M, respectively. The persistence of these gains at 1M tokens shows that SelfMem scales more robustly than manually designed memory pipelines.

Better accuracy does not require expensive memory pipelines. Table 1 also shows that higher accuracy is not simply the result of using heavier memory infrastructure. Several baselines introduce substantial additional cost through repeated LLM calls, embedding-based ingestion, or complex memory updates, yet still underperform SelfMem. For example, at 1M tokens, Mem0 costs \$18.830 and uses 25,779 embedding requests, but reaches only 0.292 Score; MemGPT and A-Mem also incur much higher costs than SelfMem while achieving lower accuracy. In contrast, SelfMem obtains the best 1M Score of 0.454 with a cost of only \$2.004 and zero embedding requests. Although RAG remains the cheapest method, its lower scores across all scales indicate that inexpensive retrieval alone is insufficient for robust long-horizon memory. These results suggest that SelfMem improves the quality–cost trade-off by letting the agent selectively construct and revise useful memory, rather than relying on exhaustive retrieval, ingestion, or fixed memory pipelines.

SelfMem is more robust across question types. Figure 3 provides a finer-grained breakdown by BEAM question type. SelfMem performs well across diverse memory demands rather than improving only one favorable category. It achieves the best score on most question types at every scale, including 9/10 types at 100K, 8/10 at 500K, and 7/10 at 1M. The strongest gains appear in information extraction, knowledge update, multi-session reasoning, preference following, summarization,

Scale	Method	$\text{Pass}_{0.5}$, ↑	Score ↑	Cost ↓	Cache	LLM Req.	Embed Req.
100K	RAG	42.00	0.339	**$0.551**	1.7%	1,548	5,656
	Full context	25.80	0.205	$2.427	87.7%	1,559	0
	Compression	22.50	0.170	$2.830	85.0%	1,726	0
	LoCoMo	36.50	0.306	$2.904	1.1%	1,587	580
	ReadAgent	37.00	0.314	$2.940	1.0%	1,600	490
	MemoryBank	34.50	0.284	$0.609	2.1%	1,602	6,212
	MemGPT	36.20	0.291	$6.434	1.0%	1,582	450
	A-Mem	35.20	0.302	$3.116	0.7%	1,606	580
	Mem0	35.80	0.282	$1.697	34.1%	2,160	2,912
	SelfMem (Ours)	**59.00**	**0.504**	$0.984	36.2%	1,932	0
500K	RAG	42.10	0.346	**$1.258**	0.6%	2,821	37,011
	Full context	–	–	–	–	–	–
	Compression	25.30	0.195	$8.144	71.4%	4,066	0
	LoCoMo	36.90	0.302	$8.156	0.6%	2,812	1,400
	ReadAgent	35.90	0.286	$7.926	0.8%	2,799	1,050
	MemoryBank	33.90	0.276	$1.335	0.5%	2,799	39,897
	MemGPT	30.70	0.249	$11.917	1.2%	2,799	980
	A-Mem	34.90	0.281	$6.500	0.6%	2,812	1,400
	Mem0	36.40	0.303	$9.366	36.4%	6,846	13,898
	SelfMem (Ours)	**57.00**	**0.487**	$1.828	33.8%	3,559	0
1M	RAG	38.30	0.320	**$1.843**	0.4%	3,468	74,153
	Full context	–	–	–	–	–	–
	Compression	25.70	0.210	$15.033	57.3%	6,376	0
	LoCoMo	29.30	0.261	$12.275	0.9%	3,662	1,421
	ReadAgent	30.70	0.268	$12.232	0.6%	3,618	1,061
	MemoryBank	31.40	0.266	$1.934	0.4%	3,499	76,903
	MemGPT	25.70	0.221	$12.264	0.8%	3,609	989
	A-Mem	31.10	0.265	$8.480	0.7%	3,588	1,421
	Mem0	34.90	0.292	$18.830	35.9%	11,356	25,779
	SelfMem (Ours)	**52.57**	**0.454**	$2.004	31.1%	4,040	0

Table 1: Main BEAM results across all conversations. Bold and underlined values mark the best and second-best results within each scale for $\text{Pass}_{0.5}$, Score, and Cost. Cache hit and request counts are reported for efficiency but not highlighted. Cost includes answer-generation and judge calls; token-count requests are excluded.

and temporal reasoning, where answers require selecting and organizing sparse information from long conversation histories. This pattern supports the core design of SelfMem: allowing the agent to construct and revise memory helps it adapt to different needs.

5 Optimization and Generalization Analysis

We next evaluate whether SelfMem can further improve its memory behavior through iterative strategy refinement. Starting from the default setting, the agent repeatedly proposes revised memory-strategy notes using only aggregate feedback from training conversations, including training scores and memory/tool diagnostics. Candidate strategies are generated from the training split conversation 0-8 and evaluated on a separate held-out test split conversation 9-19 in BEAM (Tavakoli et al., 2025) dataset. Iteration 0 corresponds to the default SelfMem setting without refinement, while later iterations use model-produced strategy notes. Held-out test scores are used only for final analysis and are never provided to the agent during strategy generation. Further details about the split construction, refinement procedure, and prompts are provided in Appendix B and Appendix C.

Results. Figure 4 shows how to read the refinement process from two complementary views. The upper panel groups results by refinement round, showing whether repeated strategy revision improves memory behavior over time. The lower panel groups results by the number of training conversations, showing whether using more training feedback leads to better held-out performance. In both panels, the gray band indicates the observed

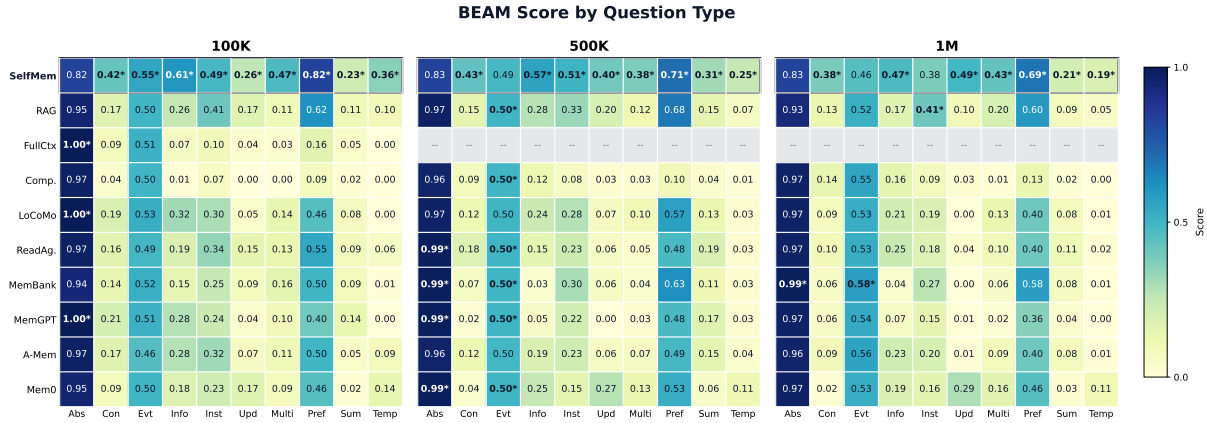

Figure 3: Question-type analysis on BEAM. Each cell reports the official BEAM score averaged over evaluated conversations for a method, context scale, and question type. Asterisks mark the best method within each scale and question type.

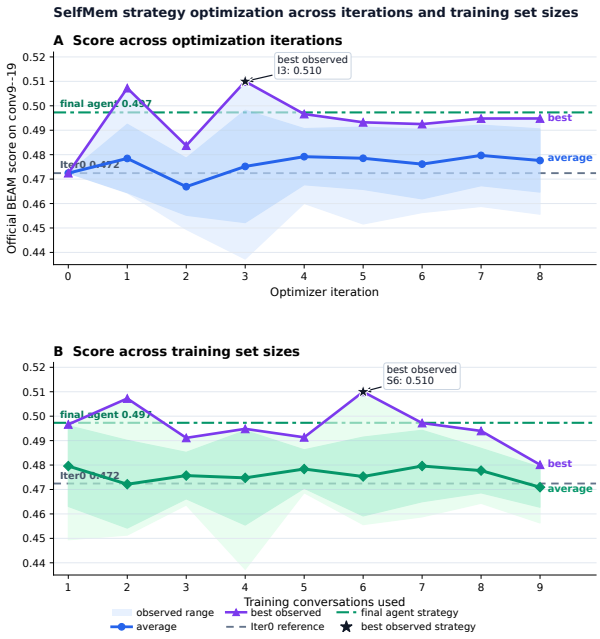

Figure 4: SelfMem strategy optimization across refinement iterations and training set sizes. The upper panel reports held-out BEAM scores across optimizer iterations, while the lower panel groups refinement outcomes by the number of training conversations used.

performance range, the average line shows the typical strategy quality, and the best-strategy line highlights the strongest memory policy discovered during the search. This layout separates average stability from peak improvement, making it clear that refinement can discover better strategies even when the average trend is not strictly increasing. Starting from the default no-refinement strategy, which achieves a held-out score of 0.472, the final agent-synthesized strategy improves to 0.497, and the best strategy discovered during refinement reaches 0.510. The strongest result appears at an intermediate refinement stage and with an intermediate amount of training feedback, rather than at the largest iteration or largest training set. This suggests that the gain does not come from simply running more refinement rounds or adding more examples. Instead, SelfMem benefits when the agent discovers a more robust procedural strategy for deciding what to store, revise, compress, and retrieve. Overall, Figure 4 shows that SelfMem exposes a useful refinement space for memory management, where better memory behavior can be learned from aggregate feedback and memory-tool diagnostics.

6 Conclusion

We introduced SelfMem, a self-optimizing memory framework for AI agents. Rather than placing the agent inside a fixed retrieval, compression, or memory-update pipeline, SelfMem exposes memory tools and feedback signals that allow the agent to decide what to store, revise, compress, and retrieve. SelfMem achieves the best performance in BEAM at 100K, 500K, and 1M token scales, improving over the strongest non-SelfMem baseline by 0.165, 0.141, and 0.134 Score points, respectively. These gains are achieved with substantially lower cost than heavier memory frameworks such as MemGPT, A-Mem, and Mem0. Also, our strategy refinement analysis further shows that memory behavior can be improved through iterative strategy updates. Together, these results indicate that providing the model with an environment for learning and exploring memory strategies is more effective than directly prescribing a fixed memory pattern.

Limitations

This work focuses on evaluating the core hypothesis of SelfMem: providing agents with an environment to explore and refine memory strategies can be more effective than prescribing a fixed memory pattern. Our experiments are conducted on BEAM with a fixed answer and judging protocol, so future work should validate the approach on additional long-horizon benchmarks, model families, and deployment settings. The iterative strategy-refinement study is currently performed at the 100K scale; extending the same refinement procedure to 500K and 1M histories would further test its scalability. In addition, our baselines are implemented in a shared harness to ensure fair comparison, but highly specialized production implementations may show different efficiency profiles. These limitations mainly concern evaluation breadth and engineering scope. They do not change the main finding: under the same benchmark, model, and judging protocol, SelfMem consistently improves long-horizon memory performance while avoiding heavier fixed memory pipelines.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. LongBench: A bilingual, multi-task benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand. Association for Computational Linguistics.

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

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 13481–13544. PMLR.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. 2024. A human-inspired reading agent with gist memory of very long contexts. In *International Conference on Machine Learning*, pages 26396–26415. PMLR.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. 2025. Agent-oriented planning in multi-agent systems. In *The Thirteenth International Conference on Learning Representations*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024b. Agentbench: Evaluating LLMs as agents. In *The Twelfth International Conference on Learning Representations*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, Bangkok, Thailand. Association for Computational Linguistics.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, and 65 others. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: towards llms as operating systems.

Joon Sung Park, Joseph O'Brien, Carrie Jun 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*, pages 1–22.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

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*, volume 36.

Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J Ross Mitchell. 2025. Beyond a million tokens: Benchmarking and enhancing long-term memory in llms. *arXiv preprint arXiv:2510.27246*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. LongMemEval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

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

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 19724–19731.

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 *Proceedings of the 41st International Conference on Machine Learning*, pages 62138–62160.

A Baseline implementation details

This section describes how we instantiate each memory and context-management baseline in our shared evaluation harness. All methods receive the same BEAM conversations and probing questions, and differ only in how they transform the conversation history into the context provided to the answer model.

Shared evaluation protocol. All baselines are evaluated in the same harness using identical conversations, probing questions, answer model, and judge. For each scale–method pair, the harness records model outputs, intermediate checkpoints, and request-level usage traces. All model calls are routed through the same official endpoint of OpenAI, ensuring that API usage, prompt caching are measured consistently across methods.

Full context and RAG. The full-context baseline sends the entire rendered conversation history to the answer model, followed by the target question. We report this baseline only when the raw history fits the model and API input limits; larger settings are marked unsupported rather than truncated, so that the baseline reflects true full-context access. The RAG baseline follows the standard retrieval-augmented generation abstraction (Lewis et al., 2020). We first flatten the conversation into retrieval chunks. In the default setting, each chunk corresponds to one rendered conversational turn, formatted as `speaker: utterance`; the chunk metadata stores the original turn index and speaker. We do not merge distant turns or maintain a learned memory state. We embed all chunks using `text-embedding-3-small` for embeddings, and build an in-memory FAISS vector index. At query time, we embed the question with the same model and retrieve the top-$k = 5$ chunks by cosine similarity. Since all embeddings are normalized, FAISS nearest-neighbor search is equivalent to cosine ranking. The retrieved chunks are concatenated in ranked order and prepended to the answer prompt as non-parametric context.

LoCoMo-style memory. The LoCoMo baseline is inspired by the benchmark's dual-memory view of long conversational history (Maharana et al., 2024). We convert each session into multiple memory records: a session summary, speaker observations when available, event records, and a fallback session-text memory. At retrieval time, the question embedding ranks all records, but the final context is structured rather than a flat nearest-neighbor list: it contains the latest session summary, relevant session summaries, relevant observations grouped by speaker, and relevant events. This implementation preserves the idea that long-term conversational memory should separate episodic summaries, speaker-specific facts, and event-level memories.

ReadAgent. ReadAgent treats long input as a document that should be paginated, gisted, and selectively revisited (Lee et al., 2024). We approximate this in the conversational setting by treating each natural session as a page. For each page, we store a short gist and the original page text. The question retrieves relevant gists, then the answer prompt receives both a pagination summary and the selected full pages as an "interactive lookup" context. This realization emphasizes ReadAgent's two-stage behavior: lightweight memory first, then targeted access to the underlying page.

MemoryBank. MemoryBank models long-term memory with profile information and a forgetting-curve-style retrieval mechanism (Zhong et al., 2024). We instantiate this by storing session summaries, raw turn memories, and speaker-specific observation memories. Each item has a strength value and a temporal distance from the most recent session. Retrieval combines semantic similarity with a retention factor that decays with distance and is moderated by strength. Retrieved memories receive a small strength update. The answer context also includes an overall history summary and personality/profile summary, reflecting MemoryBank's emphasis on persistent user-level memory.

MemGPT. MemGPT frames context management as virtual memory, with a small active context and a larger archival store (Packer et al., 2023). Our realization follows this hierarchy. The most recent sessions are placed into working memory together with a compact core-memory block summarizing user/persona information. Older sessions are converted into archival memory records containing a session summary and the session text. At question time, archival memories are retrieved by semantic similarity and paged into the answer context beside the working memory. Thus, the model sees a MemGPT-style split between always-present core/working memory and selectively retrieved archival memory.

A-Mem. A-Mem represents memory as linked agentic notes rather than independent chunks (Xu

Method	Memory unit	Memory control	Retrieval / use	Update behavior	Relation to SelfMem
Full context	Raw conversation history	No explicit memory mechanism	Directly places the history in context	No memory update; limited by context length and cost	A strong baseline when the full history fits, but it does not learn what should be remembered or compressed.
RAG	Text chunks or document passages	Designer-defined chunking, indexing, and top-k retrieval	Retrieves external passages and prepends them to the prompt	Usually static after indexing; updates require reindexing	Provides useful non-parametric access, but memory behavior is fixed by the retrieval pipeline.
ReadAgent	Gist memories and original passages	Prompted reading policy decides when to summarize and look up details	Uses gist memories first, then revisits original text when needed	Constructs task-time gists; no learned long-term memory strategy	Shows the value of action-based reading and lookup, but does not optimize memory policy from feedback.
MemoryBank	Long-term memories and user profiles	Framework-defined storage, retrieval, and memory-strength rules	Retrieves relevant memories using similarity and memory strength	Updates and strengthens memories over repeated interactions	Supports persistent personalization, but its memory operations are largely predefined.
MemGPT	Core, recall, and archival memory	Agent operates within an OS-inspired fixed memory hierarchy	Page information between active context and archival memory	Edits core memory and searches archival memory through tool calls	More adaptive than simple RAG, but still commits to a predefined memory architecture.
Mem0	Extracted and consolidated facts, optionally graph-structured	Production memory pipeline controls extraction, consolidation, and retrieval	Searches stored memories for personalized context	Continuously consolidates and updates salient facts	A strong engineered memory layer, but extraction and consolidation remain framework choices rather than learned strategy choices.
A-Mem	Structured notes with tags, links, and contextual attributes	Agentic note construction following Zettelkasten-style organization	Retrieves notes together with linked neighboring memories	Creates, links, and evolves memory notes over time	Closest to SelfMem in spirit; however, it optimizes within a note-graph design, while SelfMem lets the agent revise the memory strategy itself.
Compression	Compressed summaries or compacted state	Fixed summarization or compaction policy	Carries compressed state forward across long histories	Recompresses windows or states as history grows	Improves efficiency but can lose task-critical details; SelfMem treats compression as one memory trade-off to be decided.
SelfMem	Agent-written memory workspace plus raw feedback signals	**Agent-controlled memory strategy**	Conditionally writes, retrieves, revises, compresses, or skips memory actions	**Revises, merges, deletes, pins, and reorganizes memory based on feedback**	**Turns storage, retrieval, update, and compression into editable memory behavior optimized from quality, cost, latency, token, and cache signals.**

Table 2: Positioning SelfMem against representative long-context and agent-memory methods. Existing approaches usually instantiate a fixed memory mechanism, such as retrieval, profile memory, hierarchical paging, note graphs, fact extraction, or compression. SelfMem instead exposes these memory behaviors as agent-editable decisions and guides the agent to optimize them using task feedback and efficiency signals.

et al., 2026). We create notes from session summaries and speaker observations; when such structure is unavailable, the full session text becomes a note. Each note stores content, local context, extracted keywords, tags, and links to semantically similar previous notes. Retrieval first selects primary notes by semantic similarity to the question, then expands to linked neighbors. The prompt therefore includes both primary notes and related notes.

Mem0. The Mem0 baseline uses the public Mem0 memory SDK as an off-the-shelf long-term memory system. For each source conversation, we create an isolated memory instance backed by a local Qdrant vector store, with a separate directory and collection namespace for that conversation. This prevents memories from different conversations from leaking into one another. Conversation turns are ingested in chronological order in small batches. Each ingested message is rendered with its speaker name and turn index, so that Mem0 can extract memories from the local conversational context rather than from isolated utterances. We use the SDK's standard pipeline: an LLM-based memory extraction step converts raw dialogue into memory candidates, an embedding model encodes the extracted memories, and Qdrant stores the resulting vectors. At evaluation time, each probing question is submitted as a Mem0 search query against the corresponding conversation-specific memory store. We retrieve the top-ranked memories returned by Mem0 and render them as bullet-point context for the answer model. The answer model does not see the full conversation history unless that content has been selected by Mem0's memory extraction and retrieval pipeline. This makes the baseline stronger than simple dense retrieval in that it can store compressed, LLM-extracted facts, but also introduces additional failure modes: relevant information may be omitted during memory extraction, merged incorrectly, or not retrieved at query time. Because Mem0 performs LLM-based extraction, embedding, and vector-store writes during ingestion, it has substantially higher setup overhead than lightweight RAG; we therefore treat ingestion as part of the memory-construction cost and keep each conversation's memory store isolated and reusable

across its probing questions.

Native compression. The compression baseline uses OpenAI Responses native compaction. For histories that exceed the safe active window, the harness rolls over 200K-token windows and carries the compacted state forward with `previous_response_id`. Compression is performed once per conversation, and the compacted state is reused across sequential questions. We record OpenAI-reported cached input tokens from each response.

B SelfMem Strategy Iteration Details

This appendix records the SelfMem strategy-search protocol in a reproducible, prompt-level form. The strategy optimizer used only training-side summaries: aggregate official scores, $\text{Pass}_{0.5}$ counts, token-computed costs, LLM/tool request counts, cache statistics, and memory-operation diagnostics. It did not receive held-out questions, held-out answers, rubrics, failed nuggets, source identifiers, or per-question pass/fail labels.

Optimization protocol. SelfMem treats memory behavior as a strategy that can be revised by the model itself. At each optimization iteration, the model receives the current procedural strategy together with aggregate training feedback from prior runs. The optimizer is asked to revise only the memory procedure: what to write, what to ignore, how to merge duplicates, when to replace old memory, and how to keep the memory compact enough to be useful under a fixed retrieval budget. The optimizer is explicitly prohibited from writing benchmark facts, answer templates, or conversation-specific content into the strategy. Thus, the search process optimizes the memory-management policy rather than memorizing evaluation answers.

Iteration mechanics. Iteration 0 is the first model-generated strategy under the same optimizer constraints, before any subsequent feedback refinement. Later iterations receive only aggregate training summaries from earlier iterations and are asked to produce a revised strategy note. Each candidate strategy is then held fixed while SelfMem processes the training conversations. After the run, only aggregate outcomes are returned to the optimizer. This creates a closed-loop procedure:

1. propose a procedural memory strategy;
2. run SelfMem with that fixed strategy on training conversations;
3. summarize the run using aggregate score, cost, and memory/tool diagnostics;
4. revise the strategy without seeing per-question supervision;
5. repeat for the next optimization iteration.

The held-out conversations are never used to choose an iteration or training-set size. They are used only for final evaluation.

Selected strategy. The selected strategy is procedural rather than factual: it changes how the model writes and maintains memory, not what facts it should answer with. The optimizer converged toward a compact-memory policy. The main observation was that larger or more verbose memory was not reliably better; duplicated preference blocks, long project-style summaries, and transient implementation details often consumed context without improving transfer. The selected policy therefore emphasizes precision, stability, and low churn:

1. store only stable, high-signal user constraints, preferences, and reusable procedures;
2. reject domain-specific facts, names, quantities, and scenario-local details;
3. keep one compact canonical memory object whenever possible;
4. use short procedure signatures to identify duplicate or near-duplicate memories;
5. prefer replacement and merging over appending;
6. keep at most one or two retained memory items unless there are clearly distinct reusable categories;
7. treat cost as a secondary efficiency signal, never as a reason to drop necessary evidence.

Write and update policy. The final strategy uses a strict write gate. A memory is written only when the transcript contains multiple structural signals, such as explicit constraints, precedence rules, required output formats, conflict-resolution behavior, or verification instructions. Weak, transient, or mostly factual candidates are ignored. When

a new candidate overlaps with an existing memory, SelfMem replaces or merges the older item instead of appending another copy. This prevents memory growth from diluting the small amount of information that is actually retrieved later.

Read-time implication. Because answer-time retrieval is fixed, the strategy focuses on making the stored memory compact and structurally useful. The memory should contain reusable decision procedures, such as how to resolve conflicts or prioritize constraints, rather than long factual summaries. This makes the retrieved memory more likely to influence the model's behavior when only a small memory context is available.

C Prompt Templates

This appendix provides the prompt-level specification of SelfMem. Run-specific values, long JSON payloads, and conversation-specific fields are replaced with placeholders in braces, and line breaks are lightly normalized for readability. We include these prompts because SelfMem optimizes memory behavior through prompt-mediated strategy revision rather than through parameter updates. Table 3 gives the core memory-construction instruction, and Table 4 gives the SQLite transcript-mode addendum used when the source conversation is exposed as a read-only chronological table. Table 5 reports the fixed-evidence answering prompt. Tables 6 and 7 present the local and global strategy-optimization prompts, respectively. Finally, Table 8 summarizes the internal action space available during strategy construction and memory maintenance.

D The Use of Large Language Models

For this paper, we leveraged GPT-5.4[1] and Codex[2] to support grammar refinement, LaTeX formatting, and the preparation of figure generation code. All technical ideas, experimental designs, analyses, conclusions, and writing were developed and carried out entirely by the authors. The authors have full responsibility for the final text.

[1] https://openai.com/
[2] https://openai.com/codex/

Prompt 1: Memory-construction instruction.

Maintain a model-managed memory workspace for a long-running daily chat. Available tools include `memory_read`, `rag_search`, `meta_log_read`, `memory_change`, `memory_review`, and model-created declarative memory tools. Goal: keep memory useful, accurate, source-backed, and cheap for future conversation. Use tools when useful. Always check latest or contradictory exact facts before relying on or saving dates, counts, versions, deadlines, current status, changed values, or user preferences. Prefer atomic exact facts with source turn references when the chat contains dates, counts, versions, deadlines, changed/latest values, or user preferences. Keep broad project profiles separate from exact fact memory. Do not invent facts that are not supported by the chat or tool results.

Table 3: Prompt template for SelfMem memory construction. The prompt defines the model-managed memory workspace, the available memory tools, and the source-grounding rules used to write accurate, compact, and low-cost future-facing memory.

Prompt 2: SQLite transcript-mode addendum.

The transcript is materialized as one chronological SQLite `turns` table. Use `conversation_manifest` and read-only `conversation_sql_query` to decide how to inspect it. The harness does not split the transcript into pages and does not secretly scan the transcript to build missing-fact lists. Keep memory changes auditable by citing inspected turn ranges. Do not claim coverage for rows you have not actually inspected.

Table 4: SQLite transcript-mode addendum. The addendum specifies how the model should inspect a chronological transcript table and requires memory edits to remain auditable through cited turn ranges.

Prompt 3: Answering with fixed evidence.

Answer the benchmark question using the memory workspace and retrieved evidence. The memory workspace is a hint/index; retrieved evidence is the authority. Prefer retrieved evidence when it conflicts with memory, and use latest exact values from retrieved evidence for dates, counts, versions, deadlines, and changed facts. For exact-fact questions, compare older and later values in the retrieved evidence; do not stop at the first matching number/date/version. For current/latest/how many/average wording, look for the last matching mention by turn order and prefer the latest confirmed value over older or planned values. Pay attention to wording: if the user asks for first/original values, answer the original value and mention later updates only if useful. If the answer is not supported, say that the available memory/evidence is insufficient. Be concise but include exact dates, versions, names, and constraints when relevant. Inputs: Memory workspace: {JSON memory view}; Previous Q/A in this evaluation session: {last five Q/A pairs}; Retrieved evidence: {top retrieved chunks}; Question: {question}.

Table 5: Fixed-evidence answering prompt. The prompt treats retrieved evidence as authoritative over memory and instructs the model to resolve exact facts, updates, changed values, and insufficient-evidence cases using the retrieved conversation evidence.

Prompt 4: Local repair.

Repair the memory and retrieval strategy for training conversation {conversation}, attempt {attempt}. Target train accuracy: {target}%. The next run will rerun the same training conversation from scratch with your repaired strategy. Learn only from the final official score below; no per-question labels or rubric items are provided. Leakage guardrail: do not request, infer, or mention held-out conversation {heldout}, its questions, answers, rubrics, or hidden evaluation signals. Training references may guide strategy repair only; do not store them as future held-out facts. Previous strategy: {previous strategy}. Training score feedback: {aggregate official score JSON}. Observed memory/tool artifact from the scored attempt: {artifact summary JSON}. Write a concise repaired strategy for the next rerun. Improve the general memory and retrieval process based on the score and artifact summary only. Official score/accuracy is the primary objective; `money_cost_usd` and token counts are secondary efficiency signals and tie-breakers. Do not learn a blanket no-retrieval policy just to reduce cost: learn when evidence is needed and prefer targeted RAG/SQL/semantic search over broad dumps. You may reason about when the model should use memory, RAG, SQL, semantic search, or no retrieval, but do not prescribe a fixed tool sequence or hard-code database queries. Keep under 4,000 characters. Return only the repaired strategy note.

Table 6: Local strategy-repair prompt. The prompt revises the memory and retrieval strategy for a training conversation using only aggregate official score feedback and memory/tool artifact summaries, while enforcing held-out leakage constraints.

Prompt 5: Global refinement.

Refine SelfMem's general memory and retrieval strategy from TRAIN score feedback only. Iteration just evaluated: {iteration}. Train conversations: {train conversations}. Held-out conversation: {heldout conversation}. Held-out question count: {heldout question count}. Leakage rules: you may inspect train score summaries and train memory/tool diagnostics below; do not request, infer, or mention held-out questions, held-out answers, rubrics, or hidden evaluation signals; do not copy train answers/references as facts to remember for held-out conversations; convert the final scores into a transferable process for unseen conversations. Previous strategy: {previous strategy}. Train score feedback: {score-only feedback JSON}. Train memory/retrieval/tool diagnostics: {artifact summaries JSON}. Write the next candidate strategy as a concise memory-and-retrieval process note. Hard limit: keep it under 4,500 characters. Improve general memory construction and model-chosen retrieval behavior based on the final scores and artifact summaries only. Official score/accuracy is the primary objective; `money_cost_usd` and token counts are secondary efficiency signals and tie-breakers. Do not learn a blanket no-retrieval policy just to reduce cost: learn when evidence is needed and prefer targeted RAG/SQL/semantic search over broad dumps. You may reason about when the model should use memory, RAG, SQL, semantic search, or no retrieval, but do not prescribe a fixed tool sequence, hard-code database queries, or include benchmark answer keys.

Table 7: Global strategy-refinement prompt. The prompt updates the general SelfMem strategy from training-side aggregate score and diagnostic feedback, without access to held-out questions, answers, rubrics, failed nuggets, or benchmark answer keys.

Tool type	Tool	Description
Workspace inspection	`memory_read (target)`	Reads the current model-managed memory workspace, or a specified memory target, before the model decides whether to merge, replace, or leave memory unchanged.
Raw-chat retrieval	`rag_search (query, top_k)`	Searches the raw conversation log for supporting context during memory construction. It is used to verify what should be written or checked, and does not expose benchmark answers, rubrics, or held-out labels.
Run diagnostics	`meta_log_read (scope)`	Reads aggregate operational metadata, including token counts, cost, latency, cache statistics, and tool-call counts. These signals help diagnose inefficient memory behavior, while quality remains the primary objective.
Memory mutation	`memory_change (operation, target, content, rationale, metadata)`	Applies a memory operation such as writing, merging, replacing, compacting, indexing, or deleting memory. Each change includes a rationale and optional metadata for auditability.
Model-created tool definition	`memory_tool_create (name, description, parameters, procedure, rationale, metadata)`	Defines a reusable declarative memory-management tool when primitive operations are insufficient. This supports strategy-level abstraction without hard-coding a fixed memory policy.
Model-created tool use	`memory_tool_use (name, arguments, target, result_content, rationale, metadata)`	Applies a previously defined memory-management tool and optionally writes its result into the workspace. This enables the reuse of procedures such as deduplication, compaction, and conflict checking.
Self-test creation	`memory_probe_create (question, rationale, expected_memory_targets, metadata)`	Creates an internal, non-benchmark self-test question to check whether the current memory would support future conversations. The probe is synthetic and does not use held-out evaluation questions or answers.
Self-test execution	`memory_probe_run (probe_id, question, query, top_k, rationale, metadata)`	Runs a self-test against the memory workspace and raw-chat retrieval. The returned evidence is used only to diagnose memory coverage or redundancy, not to tune on benchmark labels.
System optimization record	`memory_system_optimize (goal, diagnosis, change, validation, rationale, metadata)`	Records a general memory-system improvement or no-op diagnosis based on observed self-test and training-run evidence. This is the main mechanism by which the model revises its memory strategy across iterations.

Table 8: SelfMem internal tools for strategy construction and memory maintenance. The tools allow the model to inspect memory state, retrieve raw-chat evidence, read aggregate diagnostics, mutate memory, define reusable memory procedures, run synthetic self-tests, and record strategy-level optimizations. They do not expose held-out questions, answers, rubrics, failed nuggets, or per-question labels.