MemPro: Agentic Memory Systems as Evolvable Programs

Anonymous ACL submission

Abstract

Long-horizon autonomous agents require memory systems to retain historical information, track evolving states, and reuse relevant knowledge beyond finite context windows. Existing agentic memory systems typically follow a memory construction–retrieval (MCR) pipeline, but often adapt mainly the memory bank while keeping the surrounding pipeline fixed after deployment. This fixed-pipeline design struggles to handle heterogeneous task-specific failure modes and can become misaligned with memory banks that evolve in scale and structure over time. To address these limitations, we propose **MemPro**, a system-level evolution framework that treats the entire MCR pipeline as an evolvable program rather than adapting only the memory bank or prompt text. MemPro maintains a version tree of runnable memory-system implementations, where an *Evolving Agent* iteratively selects promising versions, diagnoses recurring failures, and creates improved child versions through failure-mode-guided edit–debug refinement. Experiments on LongMemEval, LoCoMo, HotpotQA, and NarrativeQA show that MemPro consistently outperforms strong static and prompt-level evolving baselines within a few iterations, continues to improve with evolution, and achieves a favorable performance–cost trade-off. Code is available at `https://anonymous.4open.science/r/MemPro-64D8`.

1 Introduction

Large language models (LLMs) (Brown et al., 2020; OpenAI, 2023; Grattafiori et al., 2024; Yang et al., 2025) increasingly serve as the foundation for autonomous agents (Wang et al., 2024a), yet long-horizon tasks and sustained interactions require them to retain and reuse historical information over time. Simply extending the context window with more history is costly, noisy, and insufficient for maintaining structured long-term state (Packer et al., 2023). Memory systems therefore play a central role, maintaining and retrieving task- or user-relevant information beyond the context window (Zhang et al., 2025; Xu et al., 2025).

Recent agentic memory systems typically follow a memory construction–retrieval (MCR) pipeline, where construction builds or updates a structured memory bank from interaction histories or task inputs, and retrieval selects and uses relevant memories to answer downstream queries (Yan et al., 2025a; Chhikara et al., 2025). Prior work has improved this pipeline by designing more structured and compact memory banks, including hierarchical organizations (Packer et al., 2023; Kang et al., 2025), graph-based or dynamically linked memories (Xu et al., 2025; Chhikara et al., 2025), and summarization or compression pipelines (Zhong et al., 2023; Fang et al., 2025a). However, existing agentic memory systems typically treat the memory bank as the primary adaptive component, while the surrounding MCR pipeline is manually designed and kept fixed after deployment (Zhang et al., 2025). This fixed-pipeline assumption leads to two key limitations. First, it struggles with *task heterogeneity*: different long-term memory tasks exhibit different failure modes and therefore require different memory-use strategies. For example, temporal reasoning may require tracking event order or identifying the latest state, whereas multi-session reasoning may require linking scattered evidence across sessions—yet a fixed pipeline must apply the same strategy to all of them. Second, it creates *memory–pipeline misalignment*: as the memory bank evolves in scale and structure over time, a fixed pipeline may no longer match how memories are organized and used. Together, these limitations can lead to incomplete retrieval, noisy evidence, or ineffective use of retrieved memories. Although prompt-level optimization methods (Agrawal et al., 2026; Khattab et al., 2024) can adapt the textual components of such systems, they cannot change the executable pipeline logic and are

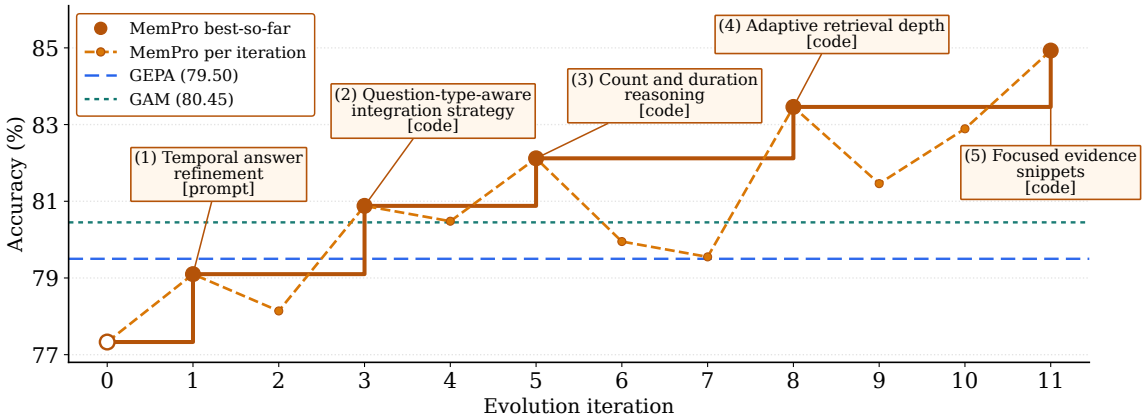

Figure 1: Evolution dynamics of MemPro on the LoCoMo evaluation set, showing the performance of each evolved version, the best-so-far performance, and the main improvements of performance-enhancing versions.

therefore insufficient to address the above limitations. This motivates a broader view of memory self-evolution: an agentic memory system should not only update the memory bank or prompt text, but also evolve the MCR pipeline system-wide.

To address these limitations, we propose **MemPro** (Agentic **Mem**ory Systems as Evolvable **Pro**grams), a system-level evolution framework that treats the MCR pipeline as an evolvable program. MemPro evolves runnable memory-system versions containing both prompts and executable code for constructing and maintaining memory banks, as well as using retrieved memories to solve downstream queries. It maintains a version tree of MCR pipeline implementations, where each node corresponds to a runnable pipeline version and its evaluation log. Starting from an initial pipeline, an *Evolving Agent* iteratively selects promising versions, diagnoses recurring failure modes, and creates improved child versions through failure-mode-guided edit–debug refinement. This tree structure lets MemPro branch from strong historical versions and explore alternative directions rather than following a single linear trajectory. Figure 1 illustrates the evolution dynamics of MemPro.

We evaluate MemPro on two long-term memory benchmarks, LongMemEval and LoCoMo, and two long-context QA benchmarks, HotpotQA and NarrativeQA. Across both memory-centric and QA settings, MemPro consistently outperforms strong static and prompt-level evolving baselines within a few iterations, and keeps improving as the version tree expands—suggesting that evolving the executable MCR pipeline yields benefits beyond adapting the memory bank or prompts alone. Our contributions are as follows:

- We identify two limitations of fixed-pipeline agentic memory systems: task heterogeneity and memory–pipeline misalignment. We argue that memory self-evolution should operate at the system level rather than only on stored memories or prompt text.
- We propose **MemPro**, a system-level evolution framework that treats the MCR pipeline as an evolvable program. MemPro maintains a version tree of runnable pipeline implementations and uses failure-mode-guided edit–debug refinement to evolve both prompts and executable pipeline code.
- We conduct extensive experiments across four long-term memory and long-context QA benchmarks, demonstrating MemPro's consistent gains over strong baselines, continued improvement with evolution, and favorable performance–cost trade-off.

2 Related Work

2.1 Agentic Memory Systems

Agentic memory systems extend LLM agents beyond finite context windows and typically follow a memory construction–retrieval pipeline, where a memory bank is built or updated from historical inputs and later retrieved for downstream tasks. Prior work has improved this pipeline through persistent memory management (Park et al., 2023; Zhong et al., 2023; Packer et al., 2023), structured or hierarchical memory organization (Kang et al., 2025; Li et al., 2026), graph-based or dynamically linked memories (Chhikara et al., 2025; Xu et al., 2025), lightweight summarization and compression (Fang et al., 2025a; Liu et al., 2026; Yan et al., 2025a),

learned or heuristic memory writing and retrieval strategies (Yan et al., 2025b; Wang et al., 2025; Yu et al., 2026), and experience-based or procedural memory reuse (Wang et al., 2024b; Fang et al., 2025b; Cao et al., 2025; Ouyang et al., 2025).

Despite these advances, most systems adapt mainly the memory bank while keeping the surrounding pipeline fixed after deployment, so it struggles with heterogeneous, task-specific failure modes and may become misaligned with evolving memory banks. In contrast, MemPro treats the whole pipeline as an evolvable program and optimizes runnable memory-system implementations.

2.2 Prompt-Level Evolution

Prompt-level evolution methods improve LLM systems by refining textual instructions without updating model weights. Representative methods optimize prompts through instruction search, feedback, or evolutionary refinement (Zhou et al., 2022; Yang et al., 2024; Pryzant et al., 2023; Yuksekgonul et al., 2025; Fernando et al., 2023; Khattab et al., 2024). GEPA further uses trajectory-level reflection to diagnose failures and evolve prompts (Agrawal et al., 2026). Closely related to agentic memory, MetaMem optimizes a self-evolving meta-memory that provides textual guidance for using memorized knowledge (Xin et al., 2026).

When applied to memory systems, prompt-level evolution can refine prompts and improve over static systems, but cannot modify the executable logic that constructs memory banks or uses retrieved memories to answer queries. MemPro instead evolves runnable pipeline implementations—both prompts and executable code—enabling system-level self-evolution.

3 Preliminaries

We introduce the Memory Construction–Retrieval (MCR) pipeline, which captures a common structure in recent agentic memory systems. The MCR pipeline consists of two stages: (1) memory bank construction and (2) memory retrieval and usage. Figure 2 (left) illustrates the MCR pipeline.

Memory Bank Construction. At time step t, the raw data D_t are first segmented into structured segments $\mathcal{S}_t = \{s_i^t\}_{i=1}^{N_t}$, where s_i^t denotes the i-th segment, such as a dialogue session or document chunk, and N_t is the number of segments. Given $\mathcal{S}_t$, the previous memory bank $\mathcal{M}_{t-1}$, and the memory construction prompt I_{mem} in Figure 6, a *Memory Agent* $\mathcal{A}_{\text{mem}}$ produces memory updates $\Delta\mathcal{M}_t = \mathcal{A}_{\text{mem}}(I_{\text{mem}}, \mathcal{S}_t, \mathcal{M}_{t-1})$, which are incorporated into the memory bank to obtain $\mathcal{M}_t = \textsc{Update}(\mathcal{M}_{t-1}, \Delta\mathcal{M}_t)$.

Memory Retrieval and Usage. Given a query q and the memory bank $\mathcal{M}$, a *Research Agent* $\mathcal{A}_{\text{res}}$ iteratively retrieves and uses memory information to solve the query. It maintains a research state z_k, which denotes the accumulated key information relevant to answering q, with $z_0 = \emptyset$. For each iteration $k = 1, \ldots, K$, the agent performs three steps under different system prompts. **(1) Retrieval.** Under the retrieval prompt I_{ret} in Figure 7, the agent examines q and the current research state z_{k-1} to determine what information is still needed, and generates a retrieval request $r_k = \mathcal{A}_{\text{res}}(I_{\text{ret}}, q, z_{k-1})$. The memory bank then returns relevant information $u_k = \textsc{Retrieve}(r_k, \mathcal{M})$, where u_k denotes the returned memory information. **(2) Integration.** Under the integration prompt I_{int} in Figure 8, the agent integrates the returned information with the query and the previous research state to update the research state, $z_k = \mathcal{A}_{\text{res}}(I_{\text{int}}, q, z_{k-1}, u_k)$. **(3) Reflection.** Under the reflection prompt I_{ref} in Figure 9, the agent judges whether the current research state z_k contains sufficient information to answer q, denoted by $b_k = \mathcal{A}_{\text{res}}(I_{\text{ref}}, q, z_k)$, where $b_k \in \{\textsc{Continue}, \textsc{Stop}\}$. If $b_k = \textsc{Stop}$ or the maximum number of retrieval steps K is reached, the agent generates the final answer $\hat{y} = \mathcal{A}_{\text{res}}(I_{\text{ans}}, q, z_k)$, where I_{ans} denotes the answer-extraction prompt; otherwise, it continues.

4 Methodology

Our objective is to enable system-level evolution by treating the entire MCR pipeline as the optimization target. To this end, we propose MemPro, which goes beyond updating the memory bank or modifying prompts and instead optimizes the MCR pipeline as an evolvable program through iterative failure-mode-guided refinement. Figure 2 gives an overview of MemPro.

4.1 MCR Version Tree

Overview. In MemPro, we construct and maintain an MCR version tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$, where each node $v \in \mathcal{V}$ represents a runnable implementation of an MCR pipeline F_v together with its evaluation log L_v. The evaluation log serves as the basis for subsequent evolution. Each non-root node v is de-

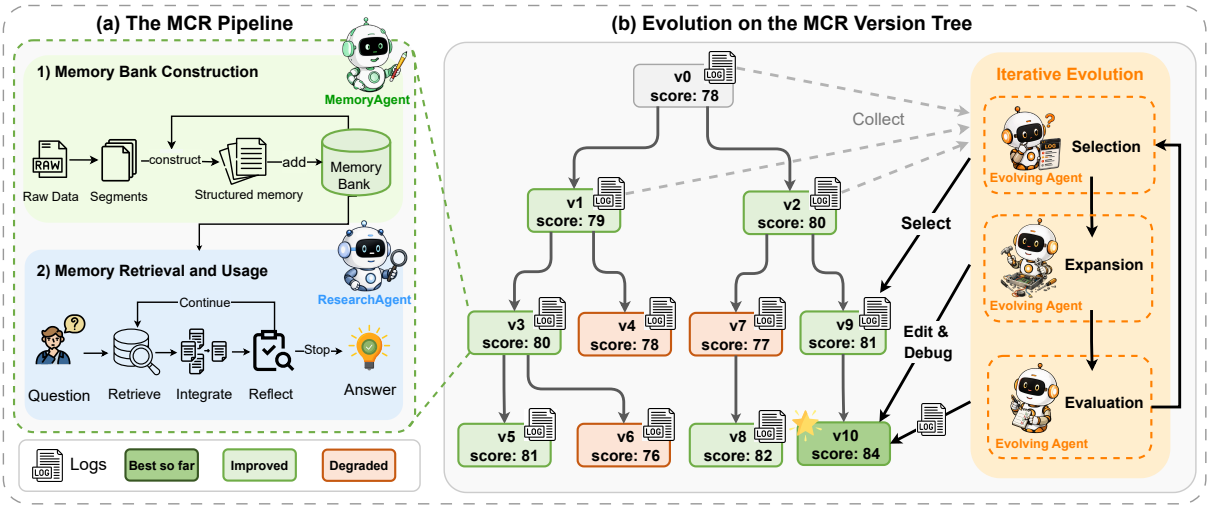

Figure 2: Overview of MemPro. (a) The MCR pipeline. (b) MemPro performs evolution over a version tree: the *Evolving Agent* selects a node based on logs, expands it into a new version, and generates its evaluation log.

rived from its parent $\mathrm{pa}(v)$, while the root v_0 stores the initial MCR pipeline F_{v_0} and log L_{v_0}.

Evaluation Log. We first split a small training set $\mathcal{D}_{\text{train}}$ from the evaluation dataset and use it to guide evolution; the detailed splitting strategy is described in 5.1. For an MCR pipeline version F_v, the evaluation log $L_v = (S_v, C_v, \mathrm{pa}(v), A_v)$ records its performance and diagnostic information on the training set. Detailed execution traces are stored in the version registry and used for diagnostic analysis. Here, S_v denotes the overall score, i.e., the average score on the training set, and serves as the primary metric for assessing the strength of version v. C_v denotes category-level scores, measuring the average score of v on each fine-grained category and indicating where the version is weak. $\mathrm{pa}(v)$ denotes the parent node from which v evolves, with the root node having no parent. A_v denotes the overall assessment, which summarizes the major failure modes of v and possible improvement directions. The root log L_{v_0} is obtained by evaluating the initial pipeline F_{v_0} on $\mathcal{D}_{\text{train}}$.

4.2 Evolution on the MCR Version Tree

Given the number of evolution iterations T, MemPro starts from the root node v_0 and performs iterative evolution on the MCR version tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ for $\ell = 1, \ldots, T$. The evolution process is controlled by an *Evolving Agent* $\mathcal{A}_{\text{evo}}$. Each iteration consists of three steps: (1) selection, which chooses the parent node for expansion, (2) expansion, which creates a new version from the selected node, and (3) evaluation, which evaluates the new pipeline version. We detail these steps below.

Selection. The *Evolving Agent* collects all evaluation logs in the version tree and compares them to select the most promising node for expansion. In addition to the selected node, it also produces a diagnostic analysis that summarizes the logs in the current tree and analyzes the failure modes and improvement directions of the selected node. Formally, given the selection prompt I_{sel} and all logs $\{L_v\}_{v\in\mathcal{V}}$, the agent outputs

$$(v_\ell^\star, R_\ell) = \mathcal{A}_{\text{evo}}(I_{\text{sel}}, \{L_v\}_{v\in\mathcal{V}}), \tag{1}$$

where $v_\ell^\star$ denotes the selected node at evolution iteration ℓ, and R_ℓ denotes the diagnostic analysis. The selection is guided by three criteria: *high overall score*, *generalizable improvement directions*, and *frequent failure modes*. More details are specified in the selection prompt I_{sel} in Figure 12.

Expansion. Given the selected node $v_\ell^\star$, its MCR pipeline version $F_{v_\ell^\star}$, the diagnostic analysis R_ℓ, and the expansion prompt I_{exp} shown in Figure 13, the *Evolving Agent* initializes the new version from the selected version, i.e., $F_{u_\ell}^{(0)} = F_{v_\ell^\star}$ and $R_\ell^{(0)} = R_\ell$. Here, $R_\ell^{(j)}$ denotes the accumulated diagnostic analysis at inner expansion step j. The expansion stage proceeds iteratively. At step j, the agent chooses one action $a_j \in \{\textsc{Edit}, \textsc{Debug}, \textsc{Terminate}\}$:

$$a_j = \mathcal{A}_{\text{evo}}\big(I_{\text{exp}}, F_{u_\ell}^{(j)}, R_\ell^{(j)}\big). \tag{2}$$

If $a_j = \textsc{Edit}$, the agent revises the current MCR pipeline according to the accumulated diagnostic analysis $R_\ell^{(j)}$, while keeping $R_\ell^{(j+1)} = R_\ell^{(j)}$:

$$F_{u_\ell}^{(j+1)} = \mathcal{A}_{\text{evo}}\Big(I_{\text{exp}}, F_{u_\ell}^{(j)}, R_\ell^{(j)}\Big). \tag{3}$$

If $a_j = \text{DEBUG}$, the agent selects a diagnostic example based on the failure analysis in $R_\ell^{(j)}$, avoiding full training-set re-evaluation:

$$x_j^{\text{diag}} = \mathcal{A}_{\text{evo}}\big(I_{\text{exp}}, R_\ell^{(j)}, \mathcal{D}_{\text{train}}\big). \quad (4)$$

The current version $F_{u_\ell}^{(j)}$ is executed on this example to obtain a full trace $\tau_j = F_{u_\ell}^{(j)}(x_j^{\text{diag}})$. The trace updates the diagnostic analysis while keeping the pipeline unchanged, $F_{u_\ell}^{(j+1)} = F_{u_\ell}^{(j)}$:

$$R_\ell^{(j+1)} = \mathcal{A}_{\text{evo}}\Big(I_{\text{exp}}, R_\ell^{(j)}, x_j^{\text{diag}}, \tau_j\Big). \quad (5)$$

This targeted debugging avoids re-evaluating the full $\mathcal{D}_{\text{train}}$ during each inner expansion step, improving efficiency. If $a_j = \text{TERMINATE}$, the expansion process stops and the current version is taken as the new pipeline version, $F_{u_\ell} = F_{u_\ell}^{(j)}$. The process repeats until the agent outputs TERMINATE or the maximum number of expansion steps J is reached; in the latter case, $F_{u_\ell} = F_{u_\ell}^{(J)}$.

Evaluation. After obtaining the new pipeline version F_{u_ℓ}, we execute it on the full training set $\mathcal{D}_{\text{train}}$ to collect execution traces:

$$\Gamma_{u_\ell} = \{\, F_{u_\ell}(x) \mid x \in \mathcal{D}_{\text{train}} \,\}. \quad (6)$$

The *Evolving Agent* then summarizes these traces into an evaluation log:

$$L_{u_\ell} = \mathcal{A}_{\text{evo}}(I_{\text{eval}}, F_{u_\ell}, \Gamma_{u_\ell}), \quad (7)$$

where I_{eval} denotes the evaluation prompt, which is provided in Figure 14. The resulting log $L_{u_\ell} = (S_{u_\ell}, C_{u_\ell}, \text{pa}(u_\ell), A_{u_\ell})$ records the training performance, category-level performance, parent node, and overall assessment of the new pipeline version. The pair (F_{u_ℓ}, L_{u_ℓ}) defines a new node u_ℓ with $\text{pa}(u_\ell) = v_\ell^\star$, which is added to $\mathcal{T}$.

5 Experiments

5.1 Experimental Setup

Datasets and splits. To comprehensively evaluate the effectiveness of MemPro, we conduct experiments on four datasets covering both agentic memory and multi-hop QA scenarios. The agentic memory datasets include LongMemEval (Wu et al., 2024) and LoCoMo (Maharana et al., 2024), while the multi-hop QA datasets include HotpotQA (Yang et al., 2018) and NarrativeQA (full-document) (Kočiský et al., 2018). For LongMemEval and LoCoMo, we randomly sample 10% of the examples from each dataset for MemPro training and use the remaining examples for testing. For HotpotQA, we consider three context-length settings—56K, 224K, and 448K—and randomly sample 30 questions (23%) for training under each setting, using the remaining questions for testing. For NarrativeQA, we randomly sample 40 questions (12%) for training and use the rest for testing. Detailed category breakdowns and additional dataset information are provided in Appendix A.3.

Metrics. For LongMemEval and LoCoMo, we use LLM-as-a-Judge for evaluation, with GPT-4o-mini (OpenAI, 2024) as the judge model across all experiments. The judge prompts for LongMemEval and LoCoMo follow GAM (Yan et al., 2025a) and LightMem (Fang et al., 2025a), respectively. For HotpotQA and NarrativeQA, we report word-level F1 (Yan et al., 2025a) between the predicted answer and the gold answer. The judge prompts are provided in Figure 10 and Figure 11.

Baselines. We compare MemPro against a series of representative baselines: (1) Direct-context methods: Full Text and Naive RAG, which represent baselines without an agentic framework. (2) Static agentic memory systems: LangMem (LangChain, 2025), Mem0 (Chhikara et al., 2025), A-Mem (Xu et al., 2025), MemoryOS (Kang et al., 2025), LightMem (Fang et al., 2025a), SimpleMem (Liu et al., 2026), and GAM (Yan et al., 2025a), which represent strong existing agentic memory baselines. (3) Prompt-level evolution methods: GEPA (Agrawal et al., 2026) and MetaMem (Xin et al., 2026), which implement prompt-level self-evolution on top of static memory systems. More details about the baselines are provided in Appendix A.4.

Implementation Details. During MemPro evolution, the *Memory Agent* and *Research Agent* in the MCR pipeline use gpt-4o-mini as the backbone, while the *Evolving Agent* is implemented with a Codex harness using gpt-5.4-medium. We set the maximum outer evolution and inner expansion iterations to 15 and 20, respectively. For Table 1, we report two evaluation-time backbone settings, `gpt-4o-mini` and `Qwen3-30B-A3B-Instruct-2507`; in each setting, the *Memory Agent* and *Research Agent* use the same backbone. Among the baselines, GEPA, SimpleMem, and GAM on LongMemEval are reproduced under settings aligned with MemPro, while

Method	LongMemEval							LoCoMo				
	Temp	Multi	Know	User	Asst.	Pref.	Avg.	Multi	Open	Single	Temp	Avg.
GPT-4o-mini												
Full Text	31.58	45.45	76.92	87.14	89.29	36.67	56.89	68.79	56.25	86.56	50.16	73.83
RAG	39.85	48.48	67.95	90.00	**98.21**	53.33	60.90	55.32	47.92	70.99	56.39	63.64
LangMem	15.79	20.30	66.67	60.00	46.43	60.00	37.20	52.10	41.65	62.80	43.25	55.45
Mem0	40.15	46.21	70.12	81.43	41.07	60.00	53.51	30.85	34.38	38.41	37.07	36.49
A-MEM	47.36	48.87	64.11	92.86	96.43	46.67	62.20	56.03	31.25	72.06	60.44	64.16
MemoryOS	32.33	31.06	48.72	80.00	64.29	30.00	44.66	56.74	45.83	67.06	40.19	58.25
LightMem	67.18	71.74	83.12	87.14	32.14	68.18	69.81	62.06	42.71	74.67	74.14	70.26
SimpleMem	69.17	60.90	78.21	85.71	75.00	73.33	71.60	64.50	44.90	76.80	74.50	72.08
GAM	60.15	70.68	78.21	75.71	94.64	70.00	72.40	79.07	57.29	86.08	73.83	80.45
GEPA	69.17	71.43	79.49	88.57	78.57	76.67	75.60	79.48	**64.28**	82.14	77.14	79.50
MetaMem	68.80	71.10	78.20	88.90	63.50	**91.70**	74.47			–		–
MemPro-5	73.86	72.41	80.93	94.22	54.68	82.11	75.77†	80.47	58.96	87.84	74.63	81.94†
MemPro-10	75.68	71.96	82.08	96.31	58.24	83.72	77.11	79.82	60.37	89.26	77.54	83.29
MemPro-15	**77.44**	**73.68**	**83.33**	**98.57**	60.71	86.67	**79.00**	**82.26**	62.50	**90.01**	**80.68**	**84.93**
Qwen3-30B-A3B-Instruct-2507												
Full Text	33.08	35.61	76.92	82.86	87.50	50.00	54.67	69.86	57.29	**87.40**	51.71	74.87
RAG	36.84	47.73	65.38	91.43	**98.21**	70.00	60.69	62.41	57.29	76.81	47.98	66.95
LangMem	37.60	38.35	67.95	78.57	42.86	70.00	50.80	53.22	50.01	63.54	30.13	53.84
Mem0	41.94	28.13	28.57	55.32	26.09	81.82	38.67	42.91	46.88	46.37	34.58	43.31
A-MEM	51.88	51.12	76.93	90.00	96.43	40.00	65.20	57.45	43.75	67.90	27.73	56.10
MemoryOS	28.57	36.84	61.54	72.86	92.86	33.33	49.60	52.48	40.62	61.59	26.48	51.30
LightMem	54.20	51.91	66.67	80.00	31.25	80.00	58.13	70.57	60.42	79.19	54.83	71.36
SimpleMem	65.41	59.40	75.64	84.29	69.64	76.67	69.20	72.00	63.00	80.10	58.90	73.13
GAM	67.67	67.29	**82.05**	92.14	66.07	63.33	72.80	71.98	**75.00**	80.61	60.43	74.46
GEPA	68.42	71.43	**82.05**	90.00	80.36	76.67	76.20	74.82	65.62	82.51	61.99	75.77
MetaMem	69.60	69.24	79.18	90.70	38.14	**94.16**	71.90			–		–
MemPro-5	68.73	73.82	79.64	90.87	82.59	77.46	76.96†	74.36	67.94	82.20	65.18	76.33†
MemPro-10	68.21	75.03	80.84	91.76	94.12	78.39	78.80	73.92	69.63	83.07	66.43	77.09
MemPro-15	**71.43**	**75.94**	**82.05**	**92.86**	**98.21**	80.00	**80.80**	**75.17**	70.83	83.47	**67.60**	**77.85**

Static agentic memory system | Prompt-level evolution | System-level evolution

Table 1: Results on LongMemEval and LoCoMo. All reported results are LLM-as-a-Judge scores. **Bold** and underline indicate the best and second-best results per backbone. Avg. denotes the mean across subcategories. MemPro-5/10/15 denote MemPro after 5/10/15 evolution iterations. † indicates that MemPro-5 already surpasses all non-MemPro baselines on the corresponding Avg. score.

the remaining results are taken from the corresponding papers (Yan et al., 2025a; Fang et al., 2025a; Xin et al., 2026). For Table 2, all baseline results are taken from the corresponding papers (Yan et al., 2025a). We replace the Qwen3-30B-A3B backbone with Qwen2.5-14B to match the baseline settings for fair comparison. Baselines missing from Table 2 are omitted because they do not support HotpotQA or NarrativeQA. For Figure 4, all baselines are reproduced under settings aligned with MemPro. Further implementation details are provided in Appendix A.5. Notably, all results from our runs, including MemPro and reproduced baselines, are averaged over three runs.

5.2 Performance on Memory Benchmarks

To evaluate MemPro and the baselines in long-term interaction scenarios, we conduct experiments on LongMemEval and LoCoMo. Table 1 shows the results, with key observations summarized below.

Naive Compression Hurts Performance, but Better Memory Design Can Overcome It. As shown in Table 1, Full Text and Naive RAG outperform several early memory systems (LangMem, Mem0, MemoryOS), suggesting that naive compression can trade accuracy for context capacity. Yet recent systems (LightMem, SimpleMem, GAM) also compress memory while surpassing Full Text and Naive RAG, showing that this loss can be overcome by better memory-system design.

Prompt-Level Evolution Improves Agentic Memory Systems. The results in Table 1 show that prompt-level evolving memory systems achieve stronger overall performance than static memory systems. This demonstrates that even prompt-level evolution alone can effectively improve performance, highlighting the necessity of evolving components beyond the memory bank in agentic memory systems.

Method	*GPT-4o-mini*					*Qwen2.5-14B*				
	HotpotQA				NarrativeQA	HotpotQA				NarrativeQA
	56K	**224K**	**448K**	**Avg.**		**56K**	**224K**	**448K**	**Avg.**	
Full Text	56.56	54.29	53.92	54.92	31.26	49.75	46.82	43.17	46.58	29.69
RAG	52.71	51.84	54.01	52.85	25.00	51.81	46.72	48.36	48.96	18.29
A-MEM	33.90	30.22	31.37	31.83	27.07	27.04	25.65	22.92	25.20	25.18
Mem0	32.58	31.74	27.41	30.58	29.16	30.12	32.44	26.55	29.70	27.80
MemoryOS	26.47	23.10	24.16	24.58	26.70	24.58	30.25	23.13	25.99	23.45
LightMem	40.93	35.28	30.02	35.41	17.51	37.30	27.72	28.25	31.09	16.57
GAM	63.22	61.46	59.81	61.50	36.86	64.07	55.99	57.87	59.31	30.17
MemPro-5	65.18†	61.72†	60.84†	62.58†	37.23†	64.36†	58.73†	57.91†	60.33†	30.34†
MemPro-10	68.05	63.94	62.36	64.78	37.84	65.41	60.54	58.76	61.57	30.78
MemPro-15	**70.32**	**65.81**	**64.02**	**66.72**	**38.12**	**66.98**	**62.28**	**59.80**	**63.02**	**31.23**

Table 2: Results on HotpotQA and NarrativeQA. All reported results are F1 scores. For HotpotQA, 56K/224K/448K denote context lengths, and Avg. denotes their mean. MemPro-5/10/15 denote MemPro after 5/10/15 evolution iterations. **Bold** and underline indicate the best and second-best results per backbone. † indicates that MemPro-5 already surpasses all non-MemPro baselines on the corresponding score.

MemPro Achieves SOTA with Only a Few Evolution Iterations. As shown in Table 1, with only 5 evolution iterations, MemPro-5 consistently outperforms the strongest baseline on each benchmark and backbone LLM, achieving state-of-the-art (SOTA) performance. This shows that MemPro can achieve strong performance with a low evolution cost, demonstrating that evolving components beyond the memory bank and prompts can further improve performance. These results highlight the promise of treating the entire memory system as the target of evolution. While MemPro leads on overall averages, it underperforms several baselines on the Single-Assistant (**Asst.**) category under the gpt-4o-mini backbone; we analyze this category- and backbone-specific gap in Appendix A.2.

MemPro Continues to Improve as Evolution Progresses After Reaching SOTA. Although MemPro-5 already outperforms the strongest baseline on each benchmark and backbone LLM, MemPro continues to improve steadily as evolution progresses. As shown in Table 1, using the Avg. columns of each benchmark and averaging them across the two backbones, MemPro improves by +1.59 on LongMemEval and +1.06 on LoCoMo from MemPro-5 to MemPro-10, and further improves by +1.95 on LongMemEval and +1.20 on LoCoMo from MemPro-10 to MemPro-15. By the 15th evolution iteration, MemPro achieves superior performance, demonstrating the effectiveness and high ceiling of its evolution process. Figure 1 visualizes this evolution trajectory, and a qualitative case study is provided in Appendix A.1.

5.3 Performance on QA Benchmarks

Beyond the memory benchmarks, we also evaluate MemPro and the baselines on HotpotQA and NarrativeQA to assess whether long-context memory capabilities transfer to knowledge-intensive QA scenarios. Table 2 reports the results, from which we draw observations broadly consistent with those on the memory benchmarks.

Direct Context Methods Beat Early Memory Systems but Trail Strong Ones. Consistent with the memory benchmarks, direct-context methods beat early memory systems but trail strong ones such as GAM, while MemPro leads.

MemPro Surpasses Baselines Within a Few Iterations and Keeps Improving. The results in Table 2 show that MemPro surpasses all non-MemPro baselines as early as the fifth evolution iteration across benchmark settings and backbone models. After reaching this strong level, MemPro continues to improve with evolution.
Experiments on the multi-hop QA benchmarks further corroborate our observations and demonstrate the robustness of MemPro's performance.

5.4 Ablation Study

We conduct ablations on LoCoMo for four key designs: (1) *w/o Code* removes code-level edits, reducing MemPro to prompt-level evolution; (2) *w/o Version Tree* degenerates tree evolution into chain evolution, expanding only from the latest version; (3) *w/o Evolution* keeps only one outer evolution iteration; and (4) *w/o Iterative Expansion* uses a sin-

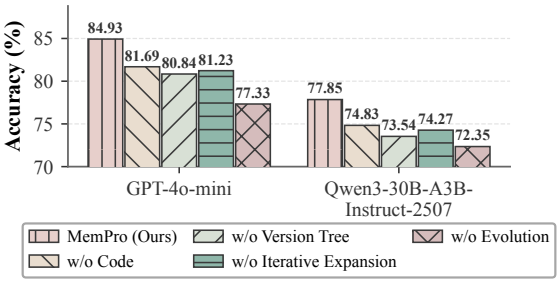

Figure 3: Ablation study on the LoCoMo.

gle edit during expansion without inner iterations. Figure 3 supports the following observations.

All components contribute to MemPro. Removing any key component leads to clear performance drops, showing that MemPro's gains come from system-level edits, tree evolution, multi-round evolution, and iterative expansion.

System-Level Evolution Matters. The *w/o Code* variant reduces MemPro to prompt-level evolution and clearly underperforms the full MemPro. This supports our central claim that self-evolution should go beyond prompt text: editing executable pipeline components expands the optimization space and strengthens system-level evolution.

Tree Evolution Outperforms Chain Evolution. The *w/o Version Tree* variant degenerates tree evolution into chain evolution and performs worse than MemPro. This shows that maintaining a version tree is more effective than following a single linear trajectory, as it allows MemPro to expand from strong historical versions rather than being constrained to the latest version.

Outer Evolution Enables Continuous Improvement. The *w/o Evolution* variant keeps only one outer evolution iteration and degrades performance. This demonstrates that multi-round evolution over the version tree is important for continuous pipeline improvement, allowing later iterations to build on evaluated versions and discover better versions.

Iterative Expansion Matters. The *w/o Iterative Expansion* variant performs only a single edit during expansion and also degrades performance. This indicates that expansion should not be treated as a one-shot edit; inner expansion iterations allow the *Evolving Agent* to refine the selected pipeline using accumulated diagnostic analysis before finalizing the new version.

We also provide a retrieval-tool ablation in Appendix A.6, showing that BM25, embedding, and PAGE-ID retrieval are complementary.

5.5 Efficiency Analysis

We conduct an efficiency analysis on LoCoMo to compare accuracy and token cost, with results shown in Figure 4. MemPro achieves the highest overall accuracy with a reasonable token cost. It substantially outperforms Full Text while using fewer tokens, and surpasses strong memory baselines such as GAM and GEPA with comparable token budgets. Although lightweight methods are cheaper, they show much lower accuracy. Overall, MemPro offers a favorable trade-off between memory performance and token efficiency.

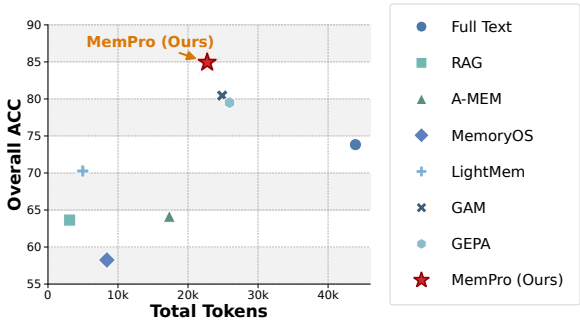

Figure 4: Efficiency analysis on LoCoMo.

6 Conclusion

We presented **MemPro**, a system-level evolution framework that treats the entire memory construction–retrieval pipeline as an evolvable program rather than adapting only the memory bank or prompt text. Motivated by the task heterogeneity and memory–pipeline misalignment of fixed-pipeline memory systems, MemPro maintains a version tree of runnable pipeline implementations and, through iterative selection, expansion, and evaluation, uses an *Evolving Agent* with failure-mode-guided edit–debug refinement to evolve both prompts and executable code. Across LongMemEval, LoCoMo, HotpotQA, and NarrativeQA, MemPro consistently surpasses strong static and prompt-level evolving baselines within a few iterations, keeps improving with evolution, and attains a favorable performance–cost trade-off, suggesting that agentic memory should be optimized as a runnable, self-evolving system.

7 Limitations

MemPro introduces an offline evolution stage in addition to task-time inference. This cost can be amortized when the evolved memory system is reused across many queries, and future work could further improve the efficiency of the evolution process. In addition, our implementation uses a capable *Evolving Agent* to edit and debug runnable MCR pipelines; future work could study more efficient or specialized evolving agents. Finally, MemPro currently uses a tree-structured evolution framework, where each new pipeline version is derived from a single parent version. We have not explored more general evolution topologies, such as graph-structured evolution that allows multiple versions to be merged or recombined. Future work could study whether these alternatives improve the diversity and efficiency of system-level evolution.

8 Ethical Considerations

MemPro targets agentic memory systems that retain and reuse historical information. In real-world applications, such memories may contain sensitive user preferences, interaction histories, personal information, or task-specific records. Responsible deployment should therefore include privacy and data-governance practices such as user consent, data minimization, access control, and mechanisms for users to inspect, correct, or delete stored memories. Our experiments use public benchmark datasets and do not collect private user data.

Because MemPro edits executable components of the memory construction–retrieval pipeline, evolved versions should be sandboxed, logged, and reviewed before deployment in user-facing or safety-critical settings. In our experiments, evolution is restricted to the MCR pipeline, while evaluation data, gold answers, judge prompts, and held-out examples are not editable or exposed to the *Evolving Agent*.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*. Oral.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. 2025. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *arXiv preprint arXiv:2512.10696*.

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*.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, and 1 others. 2025a. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025b. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Aaron Grattafiori and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25972–25981.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into self-improving pipelines.

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

LangChain. 2025. Langmem sdk for agent long-term memory.

Kai Li, Xuanqing Yu, Ziyi Ni, Yi Zeng, Yao Xu, Zheqing Zhang, Xin Li, Jitao Sang, Xiaogang Duan, Xuelei Wang, Chengbao Liu, and Jie Tan. 2026.

Timem: Temporal-hierarchical memory consolidation for long-horizon conversational agents. *Preprint*, arXiv:2601.02845.

Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. 2026. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*.

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)*.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2024. GPT-4o mini: Advancing cost-efficient intelligence.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T Le, Samira Daruki, Xiangru Tang, and 1 others. 2025. Reasoningbank: Scaling agent self-evolving with reasoning memory. *arXiv preprint arXiv:2509.25140*.

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

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.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

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

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*.

Haidong Xin, Xinze Li, Zhenghao Liu, Yukun Yan, Shuo Wang, Cheng Yang, Yu Gu, Ge Yu, and Maosong Sun. 2026. Metamem: Evolving meta-memory for knowledge utilization through self-reflective symbolic optimization. *arXiv preprint arXiv:2602.11182*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*.

BY Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. 2025a. General agentic memory via deep research. *arXiv preprint arXiv:2511.18423*.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, and 1 others. 2025b. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*.

An Yang and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pages 12028–12068.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. 2026. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639:609–616.

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

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2023. MemoryBank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. In *The eleventh international conference on learning representations*.

A Appendix

A.1 Case Study

Figure 1 shows how MemPro gradually improves the LoCoMo pipeline through the version tree. Starting from the initial framework, MemPro first improves ACC from 77.33 to 79.10 by refining temporal answers. This edit makes the model answer time-related questions with more direct and specific expressions, such as dates, months, or time spans, instead of producing broad explanations. The next major improvement comes from a question-type-aware integration strategy, which raises ACC to 80.88. This code-level change lets the system merge retrieved memories differently for different question types, so that temporal, counting, entity-centric, and multi-hop questions can preserve the information they need. Then, count and duration reasoning further improves ACC to 82.12 by explicitly handling questions that require counting repeated events or computing temporal spans. After that, adaptive retrieval depth increases ACC to 83.46 by searching more when the question needs broader evidence and avoiding unnecessary noisy retrieval for simpler questions. Finally, Focused evidence snippets improves ACC to 84.93 by placing salient evidence before integration, making key snippets easier for the model to use and reducing interference from long-context noise. Overall, MemPro improves ACC by 7.60 percentage points over the initial framework. The curve also shows several non-monotonic intermediate versions, which means that some edits introduce regressions. The version tree mitigates this problem by preserving strong previous versions and allowing later iterations to branch from them, rather than forcing evolution to follow a single linear path.

A.2 Per-Category Analysis on LongMemEval: Single-Assistant

Across most LongMemEval categories MemPro improves steadily, but on the Single-Assistant (**Asst.**) category under the gpt-4o-mini backbone it underperforms several baselines: MemPro-15 reaches 60.71, whereas direct-context methods and strong static systems such as RAG (98.21), A-MEM (96.43), and GAM (94.64) score much higher. We note three points that place this gap in context.

First, the gap is backbone-specific rather than fundamental. With the Qwen3-30B-A3B-Instruct-2507 backbone, the same evolved pipeline attains 98.21 on Single-Assistant, matching the best baseline. This suggests that the limitation arises from the interaction between this category and the weaker gpt-4o-mini backbone, not from system-level evolution itself.

Second, Single-Assistant questions ask about the assistant's own prior statements, whose answers often depend on the assistant's specific earlier wording. The MCR pipeline stores compressed, abstractive memories rather than raw turns, so assistant-side phrasing can be lost during memory construction—precisely the failure mode where uncompressed methods such as Full Text and RAG retain an advantage. Under a weaker backbone, the *Memory Agent*'s abstraction tends to be more lossy, amplifying this effect.

Third, the evolution signal for this category is sparse. Single-Assistant contains 56 questions in LongMemEval, of which only about six (10%) are sampled into the evolution set. The *Evolving Agent* therefore receives very few failure cases for this category, and because selection and evaluation are driven by the overall score, optimization naturally concentrates on more frequent failure modes.

Taken together, these observations indicate that the gap reflects a category- and backbone-specific trade-off rather than a defect of system-level evolution. A natural remedy is to make evolution category-aware—for example, adding a category-balanced objective or preserving raw assistant-side snippets alongside abstractive memories—which we leave to future work.

A.3 Dataset Details

LoCoMo (Maharana et al., 2024) is a long-term conversational memory benchmark built from multi-session user–assistant histories. It contains four question types: Single Hop, Multi Hop, Temporal, and Open Domain. LoCoMo contains 1,540 questions in total. We randomly sample 154 questions, corresponding to 10% of the full set, for framework evolution, and reserve the remaining 1,386 questions as a question-disjoint held-out evaluation set. We follow the GAM GitHub evaluation protocol and judge prompt, and use `gpt-4o-mini` as the judge model.

LongMemEval (Wu et al., 2024) evaluates long-term memory over multi-session interactions. It covers six categories: Temporal Reasoning, Multi Session, Knowledge Update, Single User, Single Assistant, and Single Preference. The benchmark contains 500 questions. We randomly sample 50

questions for framework evolution and evaluate on the remaining 450 question-disjoint held-out questions. We use an LLM-as-judge evaluator with `gpt-4o-mini` as the judge model; the judge prompt is provided in Figure 10 and Figure 11.

HotpotQA (Yang et al., 2018) is a multi-hop question-answering benchmark. We use the dataset-provided long-context settings with approximately 56K, 224K, and 448K tokens. These settings use the same underlying questions but provide different retrieved contexts. For each context length, we use the same 128 questions. The same 30 question IDs are used for evolution across all three context lengths, and the remaining 98 questions are used for held-out evaluation at each length. The 30 training question IDs across the three context lengths form 90 question-context instances, and a single HotpotQA framework is evolved on this mixed training set.

NarrativeQA (Kočiský et al., 2018) is a narrative reading-comprehension benchmark over long stories. We use the full-document setting rather than the summary setting: each document is chunked, converted into memory, retrieved at question time, and used for answer generation. We randomly sample 40 examples for framework evolution and report final results on a disjoint 300-example held-out evaluation set.

A.4 Baseline Details

We compare MemPro with direct-context methods, fixed memory-architecture systems, and optimization-based memory methods.

Full Text feeds the complete available history or document directly to the inference model. RAG retrieves query-relevant chunks from the available context and answers from the retrieved evidence.

LangMem, Mem0 (Chhikara et al., 2025), A-MEM (Xu et al., 2025), MemoryOS (Kang et al., 2025), LightMem (Fang et al., 2025a), and GAM (Yan et al., 2025a) are fixed memory architectures. These systems maintain fixed memory-writing and memory-reading procedures at evaluation time. On LoCoMo and LongMemEval, we reproduce GAM and evaluate it under the same held-out protocol as MemPro.

GEPA (Agrawal et al., 2026) is a prompt-only optimization baseline. We run GEPA with the same evolution subsets, optimizer model, and iteration budget as MemPro. GEPA can edit framework prompts but cannot modify framework code or control flow. Its final version is selected on the evolution subset and then evaluated on the same held-out examples as MemPro.

MemPro without evolution denotes our manually initialized framework before failure-driven evolution, corresponding to `v0000` in the version tree. MetaMem results are taken from the original paper where published results are available.

MemPro, MemPro without evolution, GEPA, and the reproduced GAM results are evaluated under our held-out protocol. Other previously published memory baselines are reported from LightMem or GAM under the matched benchmark and backbone setting, and MetaMem results are taken from the original paper.

A.5 Implementation Details

MemPro evolves runnable MCR framework versions rather than isolated prompt strings. Each version contains both the task-facing prompts and the executable code used by the *Memory Agent* and *Research Agent*, including memory construction, retrieval, evidence integration, context construction, and final-answer generation. The *Evolving Agent* is implemented with the OpenAI Codex coding harness using `gpt-5.4-medium`. It takes `AGENTS.md` as the task-level instruction file, which specifies the procedures for base-version selection, edit–debug iteration, and training-set performance analysis.

During evolution, all task-time MCR reasoning calls are performed with `gpt-4o-mini` using temperature 0.0 and top-p 1.0. The initial retrieval module uses three retrieval channels: BM25 keyword retrieval, `BAAI/bge-m3` semantic retrieval, and PAGE-ID retrieval. Each channel returns the top-5 candidates by default. For each question, the *Research Agent* performs at most five retrieval–integration iterations. The maximum evolution budget is capped at 15 iterations.

GEPA is given the same training set and iteration budget as MemPro, and uses the same optimizer model `gpt-5.4-medium`. However, GEPA is restricted to prompt edits and cannot modify executable framework code or control flow. For model-transfer experiments, we reuse the framework version evolved under `gpt-4o-mini` and replace only the task-time inference model with `Qwen3-30B-A3B-Instruct-2507`, without any additional evolution. `Qwen3-30B-A3B-Instruct-2507` is served with SGLang on eight H800 GPUs using bfloat16 precision, and decoded with temperature 0.7 and top-p 0.9. All LLM-as-a-Judge evaluations use

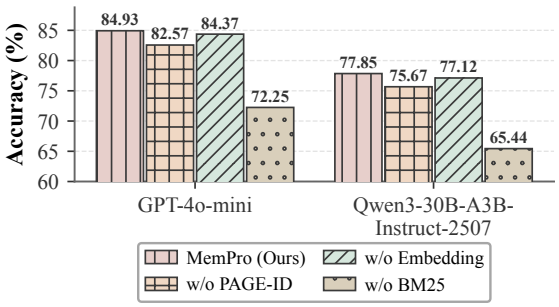

Figure 5: Retrieval ablation study on the LoCoMo. Results are reported with `gpt-4o-mini` and `Qwen3-30B-A3B-Instruct-2507`.

`gpt-4o-mini` with temperature 0. Token cost is computed over task-time inference only and excludes offline evolution calls.

A.6 Retrieval Ablation Study

We analyze the contribution of different retrieval tools in MemPro on the LoCoMo held-out set. As shown in Figure 5, removing any retrieval tool decreases performance under both backbone models, showing that keyword-based, semantic, and structural retrieval signals are complementary. BM25 is the most important retrieval channel: removing it drops accuracy from 84.93 to 72.25 with `gpt-4o-mini` and from 77.85 to 65.44 with `Qwen3-30B-A3B-Instruct-2507`. This suggests that exact keyword matching is crucial for long-term conversational memory, where answers often depend on specific names, events, dates, or surface-form cues. Embedding-based retrieval also contributes consistently: removing it drops accuracy from 84.93 to 82.57 with `gpt-4o-mini` and from 77.85 to 75.67 with `Qwen3-30B-A3B-Instruct-2507`. This indicates that semantic retrieval helps recover relevant memories when the query and stored memories use different wording. PAGE-ID retrieval has a smaller but still positive effect: removing it drops accuracy from 84.93 to 84.37 with `gpt-4o-mini` and from 77.85 to 77.12 with `Qwen3-30B-A3B-Instruct-2507`, suggesting that structural lookup provides useful auxiliary grounding. Overall, the best performance is achieved when MemPro coordinates all three retrieval tools, combining BM25 for exact keyword matching, embedding retrieval for semantic matching, and PAGE-ID retrieval for structural localization.

Memory Construction Prompt

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, such as 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.

Figure 6: Memory construction prompt used by the *Memory Agent* in the MCR pipeline to generate memory updates from structured segments and the previous memory bank.

Retrieval Prompt

You are the Retrieval Agent. Make a retrieval plan for the question using the memory below.

QUESTION:

{request}

MEMORY:

{memory}

Return one JSON object with exactly these keys:

- "info_needs": array of missing facts or sub-questions
- "tools": array using only ["keyword", "vector", "page_index"]
- "keyword_collection": array of short keyword queries
- "vector_queries": array of short natural-language queries
- "page_index": array of integers, or []

If a field is not needed, use an empty array.

Return only JSON.

Figure 7: Retrieval prompt for the *Research Agent* in the MCR pipeline.

Integration Prompt

You are the Integration Agent. Merge the current result with the new evidence into one factual summary.

QUESTION:

{question}

EVIDENCE_CONTEXT:

{evidence_context}

RESULT:

{result}

Return one JSON object with exactly these keys:

- "content": a factual summary
- "sources": page ids that support the content

If there is no useful information, set "content" to "" and "sources" to [].

Return only JSON.

Figure 8: Integration prompt for the *Research Agent* in the MCR pipeline.

Reflection Prompt

You are the Reflection Agent. Decide whether the current result is enough to answer the question.

QUESTION:

{question}

RESULT:

{result}

Return one JSON object with exactly these keys:

- "enough": true or false

- "next_question": if "enough" is false, generate the next question needed to obtain the missing information; otherwise return null

Return only JSON.

Figure 9: Reflection prompt for the *Research Agent* in the MCR pipeline.

LoCoMo Judge Prompt

Your task is to label an answer to a question as "CORRECT" or "WRONG". You will be given the following data: (1) a question (posed by one user to another user), (2) a 'gold' (ground truth) answer, (3) a generated answer which you will score as CORRECT/WRONG.

The point of the question is to ask about something one user should know about the other user based on their prior conversations. The gold answer will usually be a concise and short answer that includes the referenced topic, for example: Question: Do you remember what I got the last time I went to Hawaii? Gold answer: A shell necklace The generated answer might be much longer, but you should be generous with your grading - as long as it touches on the same topic as the gold answer, it should be counted as CORRECT.

For time related questions, the gold answer will be a specific date, month, year, etc. The generated answer might be much longer or use relative time references (like 'last Tuesday' or 'next month'), but you should be generous with your grading - as long as it refers to the same date or time period as the gold answer, it should be counted as CORRECT. Even if the format differs (e.g., 'May 7th' vs '7 May'), consider it CORRECT if it's the same date.

Now it's time for the real question:

Question:

{question}

Gold answer:

{gold_answer}

Generated answer:

{generated_answer}

Return the label CORRECT or WRONG in a json format with the key as "label". Do NOT include both CORRECT and WRONG in your response, or it will break the evaluation script.

Figure 10: Judge prompt used for LoCoMo evaluation.

LongMemEval Judge Prompt

I will give you a task type, a question, a correct answer, and a response from a model. Please answer yes if the response contains the correct answer. Otherwise, answer no.

If the response is equivalent to the correct answer or contains all the intermediate steps to get the correct answer, you should also answer yes. If the response only contains a subset of the information required by the answer, answer no.

For task types `single-session-user`, `single-session-assistant`, and `multi-session`, use the general rule above.

For task type `temporal-reasoning`, in addition to the general rule, do not penalize off-by-one errors for the number of days. If the question asks for the number of days/weeks/months, etc., and the model makes off-by-one errors (e.g., predicting 19 days when the answer is 18), the model's response is still correct.

For task type `knowledge-update`, if the response contains some previous information along with an updated answer, the response should be considered as correct as long as the updated answer is the required answer.

For task type `single-session-preference`, I will give you a question, a rubric for desired personalized response, and a response from a model. Please answer yes if the response satisfies the desired response. Otherwise, answer no. The model does not need to reflect all the points in the rubric. The response is correct as long as it recalls and utilizes the user's personal information correctly.

Task type:

{task}

Question:

{question}

Correct Answer / Rubric:

{answer}

Model Response:

{response}

Is the model response correct? Answer yes or no only.

Figure 11: Judge prompt used for LongMemEval evaluation.

Selection Prompt

You are an autonomous evolution agent for a memory-augmented question answering system.

Your task is to analyze previous evolution logs, select the most promising base version, and create a new editable child version from it.

AVAILABLE TOOLS:

- `{Tool LogReader: read version records, metrics, parent-child relations, branch history, and analysis notes.}`
- `{Tool TraceInspector: inspect representative weak or regressed cases, including retrieved evidence, memory outputs, predictions, and traces.}`
- `{Tool BaseSelector: compare candidate base versions using historical performance, stability, and branch context.}`
- `{Tool VersionCreator: create a new editable child version from the selected base version.}`

SELECTION CRITERIA:

- Prefer versions with strong overall training performance and high success rate.
- Prefer versions that improve over their parent while avoiding broad regressions.
- Prefer versions that improve recurring weak cases rather than isolated examples.
- Prefer versions with stable behavior across examples, categories, or question types.
- When scores are similar, prefer clearer failure analysis, lower regression risk, and better compatibility with the next improvement direction.

OUTPUT FORMAT:

Write a concise selection note covering the selected base version, its parent and branch position, the main reason for selection, key risks, recommended next improvement direction, whether memory artifacts can be reused, and the newly created child version for the edit stage.

Do not use held-out evaluation information, gold-answer shortcuts, sample-specific rules, or manual fixes tied to individual examples.

Figure 12: Selection-stage prompt used by the *Evolving Agent* to choose a promising base version from the MCR version tree based on node evaluation logs.

Expansion Prompt

You are an autonomous evolution agent improving a memory-augmented question answering system.

Your task is to edit the newly created child version, apply one coherent change direction, and refine it through targeted debugging before full training-set evaluation.

AVAILABLE TOOLS:

- `{Tool CodeEditor: modify prompts or executable framework logic inside the current child version.}`
- `{Tool DebugRunner: run targeted diagnostic examples and return predictions, memory states, retrieved evidence, assembled context, and traces.}`
- `{Tool TraceInspector: compare the intended behavior with the actual trace and identify recurring failure patterns.}`
- `{Tool MemoryManager: decide whether previous memory artifacts can be reused or must be rebuilt.}`

EDIT–DEBUG CRITERIA:

- Use exactly one coherent edit direction in each child version.
- The edit may target memory construction, retrieval planning, evidence integration, reflection, context assembly, or final-answer formatting.
- Debug on representative weak or regressed cases to identify recurring failure patterns.
- Revise the child version only when traces reveal a generalizable problem.
- Avoid fixes based on sample IDs, gold answers, or individual examples.

MEMORY REUSE:

Reuse memory artifacts only if the edit does not affect memory construction. Rebuild memory when the edit changes memory prompts, memory-writing logic, chunking, page construction, or any component that can alter stored memory content.

OUTPUT FORMAT:

Report the child version, base version, parent relationship, edit direction, actual changes made, memory reuse decision, targeted failure pattern, observed improvements and regressions in debugging, and whether the version is ready for full training-set evaluation.

Figure 13: Expansion-stage prompt used by the *Evolving Agent* to expand the selected MCR version into a new version through edit–debug refinement.

Evaluation Prompt

You are an autonomous evolution agent analyzing a candidate version of a memory-augmented QA system.

Your task is to evaluate the candidate version against its selected base version after full training-set execution and decide how it should be used in the version tree.

AVAILABLE TOOLS:

- `{Tool BenchmarkRunner: execute the candidate version on the training split and collect aggregate results.}`
- `{Tool ResultReader: read score summaries, success rates, token usage, and per-example outcomes.}`
- `{Tool TraceInspector: inspect weak or regressed cases through predictions, retrieved evidence, memory outputs, and execution traces.}`
- `{Tool RegistryWriter: record the version's change summary, performance analysis, and next-step recommendation.}`

ANALYSIS CRITERIA:

- Use the primary metric as the main decision signal.
- Compare the candidate against its selected base version under the same training setting.
- Consider success rate, failure count, regression severity, and improvements on previously weak examples.
- Inspect weak and regressed cases only to identify broad remaining failure modes.
- Consider token usage only when performance is tied or nearly tied.

DECISION AND OUTPUT:

Accept the version as a new strong base if it improves performance with acceptable regression risk. Keep it as a side branch if it shows a useful direction but is not clearly better overall. Recommend another edit–debug round if the change is promising but incomplete. Discard it if it weakens performance, increases instability, or improves only diagnostic traces without improving the primary metric.

Write a concise analysis note covering the candidate version, parent/base relationship, what changed, whether memory was reused or rebuilt, performance comparison, major improvements and regressions, remaining failure patterns, final decision, and recommended next experiment. Update the version registry with the same conclusion.

Do not use held-out evaluation information or gold-answer shortcuts.

Figure 14: Evaluation-stage prompt used by the *Evolving Agent* to evaluate the newly expanded MCR version and generate its evaluation log.