MemMA: Coordinating the Memory Cycle through Multi-Agent Reasoning and In-Situ Self-Evolution

Anonymous ACL submission

Abstract

Memory-augmented LLM agents maintain external memory banks to support long-horizon interaction, yet most existing systems treat construction, retrieval, and utilization as isolated subroutines. This creates two coupled challenges: *strategic blindness* on the forward path of the memory cycle, where construction and retrieval are driven by local heuristics rather than explicit strategic reasoning, and *sparse, delayed supervision* on the backward path, where downstream failures rarely translate into direct repairs of the memory bank. To address these challenges, we propose MEMMA, a plug-and-play multi-agent framework that coordinates the memory cycle along both the forward and backward paths. On the forward path, a Meta-Thinker produces structured guidance that steers a Memory Manager during construction and directs a Query Reasoner during iterative retrieval. On the backward path, MEMMA introduces in-situ self-evolving memory construction, which synthesizes probe QA pairs, verifies the current memory, and converts failures into repair actions before the memory is finalized. Extensive experiments on LoCoMo show that MEMMA consistently outperforms existing baselines across multiple LLM backbones and improves three different storage backends in a plug-and-play manner. Our code is publicly available at `https://anonymous.4open.science/r/memma-588C`.

1 Introduction

Large language models (LLMs) (Radford et al., 2018, 2019; Touvron et al., 2023) are evolving from episodic chatbots into persistent *agentic* systems (Wang et al., 2024; Yao et al., 2022; Yang et al., 2024) that execute complex workflows over days or weeks. In such settings, agents receive a continuous stream of observations—user constraints, tool outputs, and environmental feedback—whose consequences unfold over long horizons. This shift makes *controllable, long-term memory* a first-class requirement: relying solely on ephemeral context windows is insufficient, as they are computationally expensive and prone to attention dilution. To maintain coherence over time, agents must actively manage an external memory bank (Packer et al., 2023; Hu et al., 2025), deciding what to retain and how to retrieve it under uncertainty.

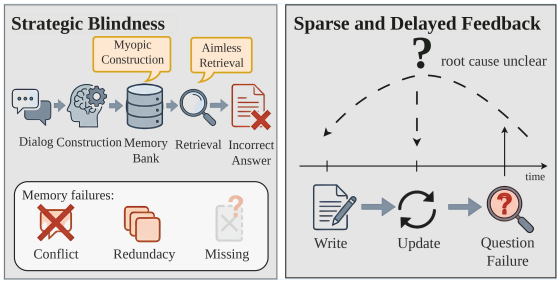


Figure 1: Two challenges in leveraging the memory cycle effect.

Effective memory, however, is not merely a storage utility; it is a closed-loop dynamic, conceptualized as the **memory cycle effect** (Zhang et al., 2025b). This cycle has three coupled phases: *construction*, *retrieval*, and *utilization*. Construction determines what information enters the memory bank and how it is organized; retrieval determines what stored information is surfaced as evidence; and utilization reveals whether the retrieved evidence is sufficient for downstream reasoning. This coupling implies that optimizing these stages in isolation is fundamentally suboptimal: a retrieval failure may stem from a much earlier construction error, while utilization outcomes should ideally feed back to improve future memory decisions. Despite this intrinsic dependency, most existing memory-augmented agents (Chhikara et al., 2025; Fang et al., 2025; Xu et al., 2025; Yan et al., 2025; Zhou et al., 2025; Shen et al., 2026) still treat memory operations as isolated, reactive subroutines, overlooking the coupling between stages. To leverage the memory cycle effect, two technical challenges must be addressed (Fig. 1).

First, on the *forward path* of the memory cycle, current systems often suffer from **strategic blindness**: they possess the mechanisms to edit memory and issue retrieval queries, yet lack explicit *meta-cognition to coordinate these actions toward downstream question answering*. As our preliminary analysis shows (Sec. 3.3), this manifests as two pathologies: (i) *Myopic Construction*, where the agent accumulates or overwrites conflicting facts without resolution; and (ii) *Aimless Retrieval*, where the agent performs shallow or repetitive searches without narrowing the true information gap. These failures suggest that *effective forward-path memory behavior requires explicit coordination between construction and retrieval, rather than isolated, short-sighted decisions*.

Second, on the *backward path* of the memory cycle, feedback from utilization to construction is typically *sparse and delayed*. Whether a memory-writing decision is useful may become clear only much later, when the agent fails a downstream question. This makes credit assignment difficult: when an answer is wrong, it is hard to identify which earlier construction decision caused the failure, allowing omissions and unresolved conflicts to persist in the memory bank and affect later updates. Although recent methods use reflection or experiential learning to improve agent behavior (Shinn et al., 2023; Zhao et al., 2024; Zhang et al., 2026), downstream failures are still rarely converted into direct signals for repairing the memory bank itself.

To address these challenges, we propose MEMMA (**Mem**ory Cycle **M**ulti-**A**gent Coordination), a plug-and-play multi-agent framework that coordinates the memory cycle along its forward and backward paths. Specifically, for the *forward path*, MEMMA separates strategic reasoning from low-level execution through a planner–worker architecture: a Meta-Thinker produces structured guidance that steers a Memory Manager during construction (what to retain, consolidate, or resolve), thereby mitigating *Myopic Construction*, and directs a Query Reasoner during retrieval by diagnosing missing evidence and how to retrieve it, replacing one-shot search with diagnosis-guided iterative refinement and thereby mitigating *Aimless Retrieval*. For the *backward path*, MEMMA introduces in-situ self-evolving memory construction: after each session, the system synthesizes probe QA pairs, verifies the memory against them, and converts failures into repair actions on the memory bank through evidence-grounded critique and semantic consolidation, before the memory is committed for future use. This directly addresses *sparse and delayed supervision* by turning downstream failures into immediate, localized repair signals for the current memory state, before flawed memories propagate into future memory updates.

Our contributions are: (i) **Analysis**. We identify two technical challenges in leveraging the memory cycle effect: *strategic blindness* on the forward path and *sparse, delayed feedback* on the backward path, and provide empirical evidence through a controlled preliminary study (Sec. 3.3). (ii) **Framework**. We propose MEMMA, a plug-and-play multi-agent framework that coordinates the memory cycle along both its forward and backward paths, combining reasoning-aware coordination for construction and iterative retrieval with in-situ self-evolving memory construction for backward repair. (iii) **Experiments**. MEMMA outperforms existing baselines on LoCoMo across multiple LLM backbones, and consistently improves three storage backends as a plug-and-play module.

2 Related Work

Memory-Augmented LLM Agents. External memory has become a core component of LLM agents that operate over long horizons. Prior work improves long-term memory from several directions, including memory architecture (Packer et al., 2023; Zhong et al., 2024), memory organization and consolidation (Xu et al., 2025; Fang et al., 2025), and memory retrieval (Du et al., 2025). These methods substantially improve individual stages of the memory pipeline, but they primarily optimize storage, organization, or retrieval in isolation. Our work is inherently different from existing work: MEMMA jointly coordinates memory construction and iterative retrieval, and converts utilization failures into direct repair signals for the memory bank. Full version is in Appendix A.

3 Preliminaries and Motivation

3.1 Problem Setting

Task Setup. We consider a long-horizon conversational setting in which an agent processes a stream of dialogue chunks $\mathcal{C} = \{c_1, \ldots, c_T\}$ over time. The stream is further organized into sessions $\mathcal{S} = \{s_1, \ldots, s_N\}$,where each session s_τ consists of one or more consecutive chunks corresponding to a coherent interaction episode. At each step t, the agent maintains an external memory bank M_t com-

posed of structured entries (e.g., text, timestamp, source, and speaker metadata), which is updated as new conversational information arrives. After processing the full stream $\mathcal{C}$, the agent is evaluated on a set of questions Q. For each query $q \in Q$, it retrieves evidence $E(q)$ from M_T and outputs an answer $\hat{y}(q)$. Our goal is to design an agent π that maximizes answer accuracy by jointly improving memory construction and retrieval.

Challenges. This setting is challenging because success depends on both memory construction and memory retrieval. During *construction*, the agent must decide what to write, update, merge, or discard when a new chunk arrives. During *retrieval and answering*, it must identify the right evidence from memory under ambiguity, temporal dependencies, and incomplete or underspecified initial queries. The challenge is therefore not merely to improve answer generation, but to maintain a useful memory bank and retrieve the right evidence under bounded memory and retrieval budgets.

3.2 Memory Cycle Effect as a Design Lens

The above challenges suggest that long-term memory should not be viewed as a linear pipeline of isolated modules. Instead, we adopt the **memory cycle effect** (Zhang et al., 2025b) as a design lens for analyzing long-term memory systems. Under this view, memory forms a closed loop with three tightly coupled phases: *construction*, *retrieval*, and *utilization*. Construction determines what information enters the memory bank and how it is organized; retrieval determines what stored information is surfaced as evidence; and utilization reveals whether the retrieved evidence is sufficient for downstream answering.

This perspective highlights two dependencies. First, there is a *forward dependency*: construction constrains retrieval, and retrieval in turn constrains utilization. A poorly constructed memory bank may omit important details, retain redundant entries, or leave conflicts unresolved, all of which degrade downstream retrieval quality. Second, there is a *backward dependency*: utilization outcomes expose deficiencies in upstream memory operations, since answering failures may stem from earlier storage omissions, unresolved contradictions, or poorly targeted retrieval. As a result, the utility of memory operations is often sparse and delayed, making isolated optimization of memory modules fundamentally suboptimal.

Together, these dependencies suggest that long-term memory should be studied as a coupled cycle rather than independent storage and retrieval components. This motivates the need for mechanisms that explicitly coordinate forward memory execution and propagate utilization feedback backward to improve future memory decisions.

Table 1: Preliminary analysis results (%) in LoCoMo dataset, GPT-4o-mini is the backbone LLM.

Method	F1	B1	ACC
Static Baseline	22.64	17.24	52.60
Unguided Active	23.49	**18.36**	54.60
Strategic Active	**24.78**	17.73	**59.21**

3.3 Motivating Analysis: Strategic Blindness

The analysis above motivates coordination across the memory cycle, but do existing active memory agents achieve this in practice? Recent agents (Fang et al., 2025; Xu et al., 2025) have moved beyond fully passive memory by introducing active updates or iterative retrieval. However, most still operate in a largely reactive manner: they trigger operations based on local context or immediate similarity signals rather than an explicit global strategy. We characterize this limitation as *strategic blindness*: the agent has the *hands* to edit memory and issue retrieval queries, but lacks the *brain* to coordinate these actions across the full memory cycle. This manifests as: (i) *Myopic Construction*: construction decisions are driven by local context rather than downstream utility. The agent indiscriminately appends, overwrites, or ignores information, leaving redundancy and conflicts unresolved. (ii) *Aimless Retrieval*: when the initial query is incomplete or semantically mismatched with stored memory, one-shot retrieval or shallow rewrites fail to surface the required evidence. Without strategic guidance, successive queries do not narrow the information gap.

Setup. To empirically validate this diagnosis, we conduct a preliminary study on a subset of LoCoMo (Maharana et al., 2024), focusing on reasoning-intensive queries by excluding adversarial samples. We compare three progressively stronger baselines using GPT-4o-mini (Hurst et al., 2024) as the backbone: (i) *Static*, which performs memory construction followed by one-shot top-30 retrieval; (ii) *Unguided Active*, which adds iterative query rewriting without strategic guidance; and (iii) *Strategic Active*, which introduces a plan-

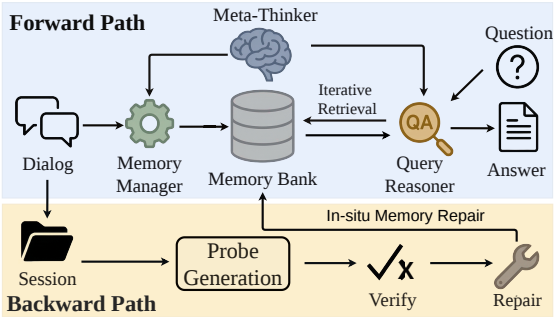


Figure 2: Overview of MEMMA.

ner to guide both construction and retrieval. We report token-level F1, BLEU-1 (B1), and LLM-as-a-Judge accuracy (ACC). More evaluation details are provided in Appendix B.1.

Empirical analysis. Table 1 reveals two findings: (i) *Refinement provides capability:* Unguided Active (54.6% Acc) outperforms Static (52.6%), confirming that one-shot retrieval often fails to surface the required evidence when the initial query is incomplete or mismatched with memory, which directly reflects *Aimless Retrieval*. (ii) *Reasoning provides control:* Strategic Active achieves a larger leap to 59.2% Acc. Since it shares the same active operators as Unguided Active, this gap reflects the value of explicit strategic guidance in addressing both *Aimless Retrieval* and *Myopic Construction*. Case studies in Appendix B.2 further illustrate both pathologies with concrete examples of redundant entries and retrieval drift. These findings suggest that active memory operations alone are insufficient: explicit strategic reasoning is needed to guide both construction and retrieval.

4 Methodology

Motivated by the memory cycle effect (Sec. 3.2) and strategic blindness (Sec. 3.3), we present MEMMA, a plug-and-play multi-agent framework that coordinates the memory cycle along its forward and backward paths (Fig. 2). Sec. 4.1 describes the forward path: a planner–worker architecture that separates strategic reasoning from low-level execution to address strategic blindness. Sec. 4.2 describes the backward path: an in-situ self-evolution mechanism that addresses sparse, delayed feedback by generating synthetic probe QA immediately after each session, providing dense, localized supervision for memory repair before the current memory is committed.

4.1 Reasoning-Aware Coordination over the Forward Path

MEMMA coordinates online construction, iterative retrieval, and answer-time utilization through specialized yet tightly coupled agents. Its key design principle is to separate strategic reasoning (what to store, what is missing, and when to stop) from low-level execution (memory editing, evidence retrieval, and answer generation).

Pipeline Overview. MEMMA uses a planner–worker architecture with four roles: (i) a *Meta-Thinker* π_p for high-level strategic reasoning, (ii) a *Memory Manager* π_s for memory editing, (iii) a *Query Reasoner* π_r for iterative query refinement, and (iv) an *Answer Agent* π_a for final response generation.

During *construction*, when a new dialogue chunk c_t arrives, π_p analyzes it against existing memory M_{t-1} and produces meta-guidance on what to retain, consolidate, or resolve. Conditioned on the guidance, π_s selects an atomic edit to update M_{t-1} to M_t. During *question answering*, given a query q, π_r retrieves candidate evidence from M_T and iteratively refines its search. At each step, π_p judges whether the current evidence is sufficient; if not, it identifies the most critical gap and directs π_r to refine the query toward complementary evidence. The loop ends when π_p deems the evidence sufficient or a budget H is reached. Then π_a generates the final answer. We detail each component below.

Meta-Thinker π_p. π_p is the planning layer of MEMMA, responsible for both construction and retrieval guidance. It produces phase-specific guidance conditioned on the current input and a bounded memory view:

$$\begin{aligned} g_t^S &\sim \pi_p(\cdot \mid c_t, \tilde{M}_{t-1}), \\ g_{q,h}^R &\sim \pi_p(\cdot \mid q, E_h, U_h, \tilde{M}_T), \end{aligned} \tag{1}$$

where g_t^S is construction guidance at step t and $g_{q,h}^R$ is retrieval guidance at refinement step h. Here, E_h denotes the evidence accumulated up to step h, $U_h = \{u_0, \ldots, u_h\}$ denotes the query history, and $\tilde{M}$ denotes a *bounded* view of the memory bank, e.g., top-k recent or semantically related entries.

Construction. g_t^S provides a set of *focus points* that flag information importance, redundancy with existing entries, and potential conflicts. These focus points steer π_s toward globally consistent memories rather than indiscriminate accumulation.

Retrieval. $g_{q,h}^R$ is a critique of the current evidence E_h. π_p evaluates coverage, consistency, and

specificity with respect to q. If the evidence is sufficient, it returns ANSWERABLE; otherwise, it returns NOT-ANSWERABLE together with a diagnosis of what is missing and how to retrieve it, e.g., a missing attribute or temporal scope. This encourages orthogonal evidence acquisition rather than near-duplicate searches. Full guidance templates and examples are in Appendix C.

Memory Manager π_s**.** π_s performs atomic memory edits based on the current chunk, bounded context, and guidance from π_p. Given c_t, $\tilde{M}_{t-1}$, and g_t^S, it selects an action $a_t^S \in \{\texttt{ADD}, \texttt{UPDATE}, \texttt{DELETE}, \texttt{NONE}\}$:

$$\begin{aligned} a_t^S &\sim \pi_s(\cdot \mid c_t, \tilde{M}_{t-1}, g_t^S), \\ M_t &= \textsc{Apply}(M_{t-1}, a_t^S), \end{aligned} \tag{2}$$

The guidance signal g_t^S helps π_s filter noise, consolidate redundancy, and resolve conflicts at the source rather than blindly appending. π_s is backend-agnostic and can wrap diverse memory implementations such as LightMem (Fang et al., 2025) and A-Mem (Xu et al., 2025).

Query Reasoner π_r**.** π_r implements the *active retrieval policy*. To overcome the *Aimless Retrieval* (Sec. 3.3), it replaces one-shot search with an iterative *Refine-and-Probe* loop. Let $u_0 = q$ be the initial query and $U_h = \{u_0, \ldots, u_h\}$ the query history. At step h, when π_p deems the current evidence E_h NOT-ANSWERABLE, it emits guidance $g_{q,h}^R$. π_r then proposes the next query and retrieves additional evidence:

$$\begin{aligned} u_{h+1} &\sim \pi_r(\cdot \mid U_h, E_h, g_{q,h}^R), \\ E_{h+1} &= E_h \cup \textsc{Search}(M_T, u_{h+1}). \end{aligned} \tag{3}$$

The loop terminates when π_p returns ANSWERABLE or the budget H is reached. Each refinement step targets the specific information gap diagnosed by π_p, so successive queries narrow the deficit rather than drifting across redundant rewrites. Full query rewrite prompt templates are in Appendix D.

Answer Agent π_a**.** Once the retrieval loop terminates, π_a generates the final answer $\hat{y}(q)$ based on the query and the final evidence set $E(q) = E_H$:

$$\hat{y}(q) = F_{\pi_a}(q, E(q)), \tag{4}$$

where F_{π_a} denotes a generation function (e.g., an LLM call). In our experiments, π_a is kept frozen to decouple answer-generation capacity from memory quality, so that gains can be attributed to coordination over the memory cycle rather than to the parametric knowledge of π_a.

4.2 In-Situ Self-Evolving Memory Construction

A major bottleneck in the memory cycle is that feedback for construction is typically sparse and delayed. The utility of a storage decision made in session τ may become observable only much later, when the agent fails a downstream question. Optimizing construction solely from final-task outcomes makes credit assignment difficult and lets early omissions propagate uncorrected. To address this, we introduce **in-situ self-evolving memory construction**, which provides dense intermediate feedback for the construction stage. Instead of waiting for a future user query to expose a memory failure, MEMMA synthesizes a set of probe QA pairs after each session and uses them to verify and repair the current memory before it is committed.

Probe Generation. Let s_τ denote the current session, and let $M_\tau^{(0)}$ denote the provisional memory state obtained after applying the construction policy of Sec. 4.1 to s_τ. To obtain intermediate supervision, we construct a probe set

$$\mathcal{Q}_\tau = \{(q_j, y_j)\}_{j=1}^{J}, \tag{5}$$

where each (q_j, y_j) is a synthetic question–answer pair grounded in s_τ and its relevant historical context $\tilde{M}_{\tau-1}$. The questions are designed to test whether the provisional memory faithfully captures and can retrieve information introduced in the current session, covering single-session factual recall, cross-session relational reasoning, and temporal inference (Shen et al., 2026). This turns a delayed end-task signal into J localized supervision signals immediately after construction. Design details are in Appendix E.1.

In-situ Verification. Given $\mathcal{Q}_\tau$, MEMMA verifies the provisional memory state $M_\tau^{(0)}$ immediately after the initial construction pass. For each probe q_j, we retrieve top-k evidence from $M_\tau^{(0)}$ and generate an answer with π_a:

$$E_j = \textsc{Search}(M_\tau^{(0)}, q_j), \quad \hat{y}_j = F_{\pi_a}(q_j, E_j). \tag{6}$$

A probe is considered failed if $\hat{y}_j$ is judged incorrect with respect to y_j. Such failures provide localized evidence that M_0 is insufficient for information introduced in or linked to s_τ.

Evidence-grounded Repair. For each failed probe, a reflection module converts the failure into a repair proposal. Conditioned on the question, gold answer, predicted answer, retrieved evidence, and

the provisional memory state $(q_j, y_j, \hat{y}_j, E_j, M_\tau^{(0)})$, it diagnoses whether the failure reflects missing information or memory content that is difficult to retrieve in its current form, and then proposes a candidate repair fact. Collecting all failed probes in the current batch yields a set of repair proposals

$$\mathcal{R}_\tau = \{r_j\}_{q_j \in \mathcal{Q}_\tau^{\text{fail}}}, \tag{7}$$

where $\mathcal{Q}_\tau^{\text{fail}} \subseteq \mathcal{Q}_\tau$ denotes the failed probes.

Semantic Consolidation. Applying all repairs in $\mathcal{R}_\tau$ directly would reintroduce redundancy or conflicts, e.g., when two probes request overlapping or inconsistent additions. We therefore consolidate the candidate repair facts against $M_\tau^{(0)}$. For each candidate fact, the consolidation step assigns one of three actions with respect to the existing memory: `SKIP` if it is redundant, `MERGE` if it complements an existing entry, or `INSERT` if it is novel. This resolves both conflicts with the existing memory and conflicts across repair proposals before any update is written back. The refined memory is obtained as

$$M_\tau^* = \textsc{Refine}(M_\tau^{(0)}, \mathcal{R}_\tau), \tag{8}$$

where REFINE denotes consolidation followed by write-back over $\mathcal{R}_\tau$. In this way, utilization failures are detected and repaired during construction before they can propagate into later memory updates, while keeping the evolving memory compact and internally consistent.

5 Experiments

This section presents the experimental results. We first compare MEMMA with existing baselines, then evaluate its flexibility across storage backends, and finally assess the contribution of each component and key design choices.

5.1 Experimental Setup

Datasets. We evaluate MEMMA on LoCoMo (Maharana et al., 2024), a benchmark for long-horizon conversational memory. Following prior work (Yan et al., 2025; Fang et al., 2025), we exclude the adversarial subset and focus on the reasoning-intensive QA setting. More dataset details are provided in Appendix F.1.

Baselines. We compare against two passive baselines: *Full Text* and *Naive RAG* (Gao et al., 2023), and four active memory systems: *LangMem* (LangChain, 2025), *Mem0* (Chhikara et al., 2025), *A-Mem* (Xu et al., 2025), and *LightMem* (Fang et al., 2025). Additional baseline details are in Appendix F.2.

Evaluation Protocol. Following prior work (Yan et al., 2025; Chhikara et al., 2025), we report three metrics: token-level F1 (F1), BLEU-1 (B1), and LLM-as-a-Judge accuracy (ACC). F1 and B1 measure lexical overlap with the reference answer; ACC measures semantic correctness via a judge model. GPT-4o-mini (Hurst et al., 2024) and Claude-Haiku-4.5 (Anthropic, 2025a) are used as the backbones for the Memory Manager, Meta-Thinker, and Query Reasoner. To isolate memory construction quality from answer-generation capacity, we fix GPT-4o-mini as both the Answer Agent and the LLM judge across all experiments. The retrieval budget is top-30 entries, the iterative refinement budget is H=3, and we generate J=5 probe QA pairs per session for self-evolution. Additional implementation details are in Appendix F.3.

5.2 Main Comparison with Baselines

To evaluate MEMMA, we compare it with baselines. We use LightMem as the storage backend of MEMMA, denoted by MEMMA_{LM}. GPT-4o-mini and Claude-Haiku-4.5 are the backbones. Other settings follow these in Sec. 5.1.

Table 2 reports the results. Three findings emerge: **(i)** MEMMA_{LM} *achieves the best overall performance under both backbones*. Under GPT-4o-mini, it reaches 49.40 F1, 38.28 B1, and 81.58 ACC, improving over LightMem by +4.82 F1, +1.62 B1, and +5.92 ACC. Under Claude-Haiku-4.5, it again achieves the best overall ACC, improving from 73.03 to 76.97 over LightMem. **(ii)** *The gains are strong at the category level.* Under GPT-4o-mini, MEMMA_{LM} improves most on Multi-Hop and Single-Hop, raising ACC from 65.62 to 78.12 and from 78.57 to 82.86, respectively. The Multi-Hop gains are consistent with diagnosis-guided iterative retrieval helping recover distributed evidence, while the Single-Hop gains suggest that construction guidance and self-evolution help preserve precise answer-bearing details. **(iii)** MEMMA_{LM} *improves an already strong baseline.* LightMem is already the strongest baseline, yet MEMMA_{LM} further improves it under both backbones, suggesting that the gain comes from memory-cycle coordination rather than a stronger storage backend.

Table 2: Results on LoCoMo across four question categories (multi-hop, temporal, open-domain, single-hop). We report F1, B1, and ACC (%). Best results are in **bold**. GPT-4o-mini and Claude-Haiku-4.5 are backbones; GPT-4o-mini is the answer agent. MEMMA$_{\text{LM}}$ uses LightMem as storage backend.

Model	Method	Multi-Hop			Temporal			Open-Domain			Single-Hop			Overall		
		F1	B1	ACC	F1	B1	ACC	F1	B1	ACC	F1	B1	ACC	F1	B1	ACC
GPT	Full Text	29.41	21.16	43.75	29.95	19.33	51.35	18.25	19.56	61.54	41.45	29.96	74.29	34.13	24.63	61.18
	Naive RAG	15.84	9.50	31.25	17.30	12.36	35.14	17.40	16.65	46.15	39.32	30.35	58.57	27.14	20.41	46.05
	LangMem	12.55	9.22	25.00	15.23	11.53	21.62	14.91	14.03	38.46	23.52	17.59	35.71	18.46	14.05	30.26
	A-Mem	15.56	10.88	31.25	55.01	42.40	51.35	18.18	15.27	53.85	42.72	32.43	62.86	37.90	28.85	52.63
	LightMem	33.74	29.33	65.62	**59.76**	**51.12**	78.38	**31.85**	**24.23**	**76.92**	43.88	34.68	78.57	44.58	36.66	75.66
	MEMMA$_{\text{LM}}$	**48.15**	**39.67**	**78.12**	57.21	41.94	**83.78**	24.58	22.44	**76.92**	**50.45**	**38.66**	**82.86**	**49.40**	**38.28**	**81.58**
Claude-Haiku	Full Text	29.41	21.16	43.75	29.95	19.33	51.35	18.25	19.56	61.54	41.45	29.96	74.29	34.13	24.63	61.18
	Naive RAG	15.84	9.50	31.25	17.30	12.36	35.14	17.40	16.65	46.15	39.32	30.35	58.57	27.14	20.41	46.05
	LangMem	20.05	14.85	34.38	34.72	26.33	37.84	20.01	20.85	69.23	22.65	16.19	48.57	24.81	18.78	44.74
	A-Mem	15.79	10.32	28.13	56.41	43.23	54.05	16.34	17.76	38.46	38.37	27.98	65.71	36.12	27.10	52.63
	LightMem	35.11	31.85	59.38	58.42	**49.85**	**89.19**	**32.60**	24.43	69.23	44.06	**36.56**	71.43	44.69	**37.77**	73.03
	MEMMA$_{\text{LM}}$	**35.38**	**32.48**	**65.62**	**59.25**	44.66	83.78	28.59	**26.86**	**84.62**	**45.31**	35.85	**77.14**	**45.10**	36.53	**76.97**

5.3 Flexibility across Storage Backends

To assess the flexibility of MEMMA across storage backends, we instantiate it on top of three memory systems: Single-Agent (Yan et al., 2025) (MEMMA$_{\text{SA}}$), A-Mem (MEMMA$_{\text{AM}}$), and LightMem (MEMMA$_{\text{LM}}$). All other components and settings are fixed as in Sec. 5.1.

Table 3 reports results on LoCoMo under GPT-4o-mini. Two observations emerge. **(i)** MEMMA *consistently improves all backends.* In terms of ACC, MEMMA improves the Single-Agent backend from 52.60 to 84.87, A-Mem from 52.63 to 78.29, and LightMem from 75.66 to 81.58. For A-Mem and LightMem, the gains are also consistent in F1 and B1. For the weaker Single-Agent backend, B1 decreases even though Acc rises sharply, suggesting that MEMMA improves semantic correctness more than lexical overlap in this setting. These results indicate that MEMMA improves long-horizon memory across diverse storage implementations. **(ii)** *The gains of* MEMMA *complement storage quality rather than replace it.* Among the enhanced variants, MEMMA$_{\text{LM}}$ achieves the strongest overall performance, which is consistent with LightMem being the strongest standalone backend. This pattern suggests that MEMMA improves how memory is coordinated, rather than relying on a particular storage design.

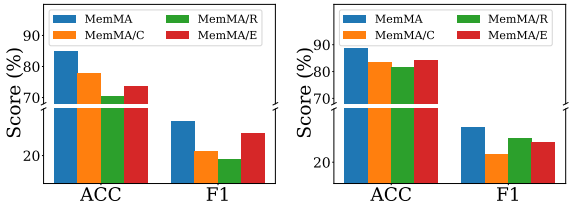
5.4 In-depth Dissection of MemMA

Ablation Studies. To understand the contributions of key components in MEMMA, we implement three ablated variants on the Single-Agent backend: **(i)** MEMMA/C removes Meta-Thinker guidance during construction and directly uses the Memory Manager for memory writing; **(ii)** MEMMA/R removes iterative retrieval, reverting to one-shot retrieval based on semantic similarity; and **(iii)** MEMMA/E removes the probe-and-repair loop of in-situ self-evolving memory construction and directly commits $M_\tau^{(0)}$ to the memory bank.

Table 3: Flexibility across backends on LoCoMo under GPT-4o-mini. Best results per backend are in **bold**.

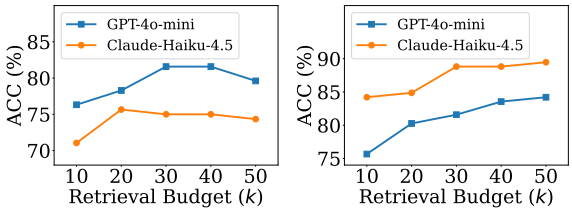
Method	F1	B1	ACC
Single-Agent	22.64	**17.24**	52.60
MEMMA$_{\text{SA}}$	**23.64**	12.94	**84.87**
A-Mem	37.90	28.85	52.63
MEMMA$_{\text{AM}}$	**46.23**	**35.13**	**78.29**
LightMem	44.58	36.66	75.66
MEMMA$_{\text{LM}}$	**49.40**	**38.28**	**81.58**

Fig. 3 reports the results under GPT-4o-mini and Claude-Haiku-4.5. The full MEMMA$_{\text{SA}}$ achieves the strongest overall performance, while the variants reveal complementary weaknesses. Specifically: (i) *Iterative retrieval is the most critical forward-path component.* MEMMA$_{\text{SA}}$/R causes the largest drop under both backbones, reducing ACC from 84.87 to 70.39 with GPT-4o-mini and from 88.82 to 81.58 with Claude-Haiku-4.5. This confirms that one-shot retrieval remains a major bottleneck and that diagnosis-guided refinement is essential for narrowing the information gap. (ii) *Self-evolution repairs construction omissions.* MEMMA$_{\text{SA}}$/E causes the second-largest degradation (ACC: 84.87 $\rightarrow$ 73.68 with GPT-4o-mini). The large ACC drop with only moderate F1 change suggests that self-evolution mainly improves semantic correctness by repairing missing information during construction. (iii) *Construction guidance reduces upstream noise.* MEMMA$_{\text{SA}}$/C reduces ACC from 88.82 to 83.55 with Claude-Haiku-4.5. This shows that construction decisions benefit from explicit strategic guidance rather than



(a) GPT-4o-mini (b) Claude-Haiku-4.5

Figure 3: Ablation studies of MEMMA$_{\text{SA}}$ under GPT-4o-mini and Claude-Haiku-4.5 on LoCoMo.



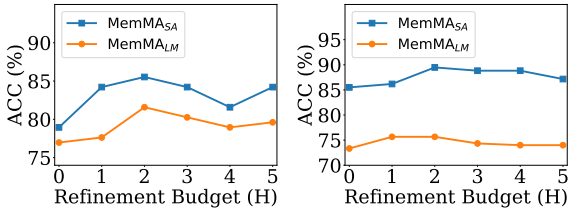
(a) MEMMA$_{\text{LM}}$ (b) MEMMA$_{\text{SA}}$

Figure 4: Impact of retrieval budget k of MEMMA under both GPT-4o-mini and Claude-Haiku-4.5.

local heuristics alone, as the Meta-Thinker helps determine what should be retained, consolidated, or resolved before information enters the memory bank. These ablations confirm that MEMMA's gains come from complementary improvements on both paths of the memory cycle.

Impact of retrieval budget k. We vary $k \in \{10, 20, 30, 40, 50\}$ on both Single-Agent and LightMem backends and report results in Fig. 4. We observe that the optimal k depends on storage quality. For MEMMA$_{\text{LM}}$, ACC peaks at k=30–40 (81.58) and declines at k=50 (79.61), indicating a sweet spot beyond which additional retrieval introduces noise. For MEMMA$_{\text{SA}}$, ACC increases steadily from 75.66 at k=10 to 84.21 at k=50, without saturation. We attribute this contrast to storage quality: stronger backends produce higher-quality, less redundant entries, so a moderate k suffices and excess retrieval dilutes the evidence; weaker backends need a larger k to retrieve enough evidence from sparser memory banks.

Impact of retrieval refinement budget H. We vary the refinement budget $H \in \{0, 1, 2, 3, 4, 5\}$ under both GPT-4o-mini and Claude-Haiku-4.5. The results of MEMMA$_{\text{SA}}$ and MEMMA$_{\text{LM}}$ are reported in Fig. 5. We observe that ACC improves sharply from one-shot retrieval (H=0) to a small H and then declines. For example, MEMMA$_{\text{SA}}$'s ACC rises from 78.95 at H=0 to 85.53 at H=2, then drops back to 81.58 at H=4. This shows that diagnosis-guided refinement converges quickly: one or two additional retrieval rounds suffice to close most information gaps, while further iterations risk retrieval drift. This validates the effectiveness of the Meta-Thinker's answerability diagnosis, which directs each refinement step toward the specific missing evidence rather than redundant searches. More analysis of the impact of probe generation model are in Appendix G.



(a) GPT-4o-mini (b) Claude-Haiku-4.5

Figure 5: Impact of refinement budget H of MEMMA.

5.5 Case Studies

We conduct a case study to better understand why MEMMA improves long-horizon QA. Our findings indicate that: **(i)** on the forward path, construction-time Meta-Thinker guidance determines whether answer-bearing details survive in memory, while diagnosis-guided iterative retrieval determines whether missing evidence is surfaced before the system commits to an answer. Importantly, iterative retrieval cannot compensate for details that were never preserved during construction. The cases also show that the retrieval controller and the storage backend play distinct roles: the Meta-Thinker and Query Reasoner identify the information gap, while the backend determines whether the required evidence can actually be recovered; **(ii)** on the backward path, in-situ self-evolution converts local probe failures into targeted memory repairs that transfer to downstream QA, for example by inserting missing named entities, sharpening vague event descriptions, and completing partial evidence clusters. Detailed examples are in Appendix H.

6 Conclusion

We introduce MEMMA, a plug-and-play multi-agent framework that coordinates the memory cycle along its forward and backward paths. On the forward path, a Meta-Thinker separates strategic reasoning from low-level execution, addressing strategic blindness in construction and retrieval. On the backward path, in-situ self-evolution converts probe QA failures into direct memory repair before the memory is committed. Experiments on LoCoMo show that MEMMA outperforms all baselines across multiple backbones and consistently improves three different storage backends.

7 Limitations

Our evaluation focuses on a dialogue-centric long-horizon memory benchmark. While LoCoMo covers diverse question types, including single-hop, multi-hop, temporal, and open-domain reasoning, it does not capture all settings in which persistent memory may be needed.

In addition, the backward path assumes that interaction streams can be organized into sessions and that synthetic probe QA can provide useful localized supervision. These assumptions are natural for the benchmark studied here, but may require adaptation in settings with less clear session boundaries or more open-ended interaction structure.

8 Ethics Statement

This work studies long-horizon memory management for LLM agents. All experiments are conducted on the publicly available benchmark, which consists of synthetic conversations and does not contain real user data. No personally identifiable information is collected, stored, or processed in this work. We note that improving memory quality in agent systems may raise broader considerations for real-world deployment, including user privacy, informed consent for data retention, controllability over stored memories, and the risk of persisting incorrect information through automated repair. While these concerns are beyond the scope of the present study, we believe they should be treated as first-class design requirements in any production deployment of memory-augmented agents.

References

Anthropic. 2025a. Claude haiku 4.5 system card.

Anthropic. 2025b. Claude opus 4.5 system card.

Anthropic. 2025c. Claude sonnet 4.5 system card.

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*.

Xingbo Du, Loka Li, Duzhen Zhang, and Le Song. 2025. Memr^3: Memory retrieval via reflective reasoning for llm agents. *arXiv preprint arXiv:2512.20237*.

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

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, Haofen Wang, et al. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1):32.

Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, et al. 2026. Evermemos: A self-organizing memory operating system for structured long-horizon reasoning. *arXiv preprint arXiv:2601.02163*.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. 2025. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

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

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*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

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

OpenAI. 2024. New embedding models and api updates.

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.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: a temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.

Samarth Sarin, Lovepreet Singh, Bhaskarjit Sarmah, and Dhagash Mehta. 2025. Memoria: A scalable agentic memory framework for personalized conversational ai. In *2025 5th International Conference on AI-ML-Systems (AIMLSystems)*, pages 32–39. IEEE.

Zhiyu Shen, Ziming Wu, Fuming Lai, Shaobing Lian, and Yanghui Rao. 2026. Membuilder: Reinforcing llms for long-term memory construction via attributed dense rewards. *Preprint*, arXiv:2601.05488.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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*.

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

Piaohong Wang, Motong Tian, Jiaxian Li, Yuan Liang, Yuqing Wang, Qianben Chen, Tiannan Wang, Zhicong Lu, Jiawei Ma, Yuchen Eleanor Jiang, et al. 2025a. O-mem: Omni memory system for personalized, long horizon, self-evolving agents. *arXiv preprint arXiv:2511.13593*.

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

Yaxiong Wu, Yongyue Zhang, Sheng Liang, and Yong Liu. 2025. Sgmem: Sentence graph memory for long-term conversational agents. *arXiv preprint arXiv:2509.21212*.

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*.

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

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, et al. 2026. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*.

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

Zeyu Zhang, Quanyu Dai, Rui Li, Xiaohe Bo, Xu Chen, and Zhenhua Dong. 2025b. Learn to memorize: Optimizing llm-based agents with adaptive memory framework. *arXiv preprint arXiv:2508.16629*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

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.

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

A Full Details of Related Works

A.1 Memory-Augmented LLM Agents

External memory (Hu et al., 2025; Zhang et al., 2025a) has become a core component of LLM agents that operate over long horizons. Existing work can be broadly organized along three dimensions.

At the *architecture level*, early systems explore how to structure the memory bank. Generative Agents (Park et al., 2023) maintains a chronological memory stream with reflection-based retrieval. MemGPT (Packer et al., 2023) introduces a hierarchical design that treats the context window as virtual memory managed by the LLM itself. MemoryBank (Zhong et al., 2024) adds temporal dynamics through forgetting-curve-based decay. More recent work moves toward richer structure: SGMem (Wu et al., 2025) represents dialogue as sentence-level graphs to capture cross-turn associations, and Memoria (Sarin et al., 2025) provides a scalable framework for personalized conversational memory.

At the *organization level*, systems shift focus from how memory is structured to what is stored and how it is consolidated. Mem0 (Chhikara et al., 2025) extracts and consolidates salient facts from multi-session conversations, reducing redundancy at the source. A-Mem (Xu et al., 2025) goes further by dynamically organizing memories into interconnected notes following the Zettelkasten method, allowing entries to evolve as new information arrives. LightMem (Fang et al., 2025) takes a different angle, designing a lightweight multi-stage pipeline inspired by the Atkinson–Shiffrin model to balance memory quality with computational cost. SimpleMem (Liu et al., 2026) pushes efficiency further through semantic lossless compression and recursive consolidation, while EverMemOS (Hu et al., 2026) introduces a self-organizing memory operating system for structured long-horizon reasoning.

At the *retrieval level*, the focus shifts to how stored information is surfaced. Zep (Rasmussen et al., 2025) organizes memory as a temporal knowledge graph for time-aware retrieval, enabling queries that require temporal reasoning. MemR^3 (Du et al., 2025) introduces a closed-loop retrieval controller with a router and an explicit evidence-gap tracker, moving retrieval from a one-shot operation to an iterative decision process. LangMem (LangChain, 2025) provides a practical SDK for memory extraction and retrieval in agent frameworks.

These methods substantially improve individual stages of the memory pipeline, but they primarily optimize storage, organization, or retrieval in isolation. By contrast, MEMMA addresses a broader scope: it coordinates both construction and retrieval along the forward path of the memory cycle, and further converts utilization failures into direct repair signals for the memory bank along the backward path.

A.2 Self-Evolution and Reflection for LLM Agents

A growing body of work improves agent behavior through self-feedback, but existing approaches operate at progressively deeper levels of abstraction from the memory bank itself.

At the *output level*, the simplest form of self-improvement operates on model responses directly. Self-Refine (Madaan et al., 2023) iteratively critiques and revises model outputs within a single generation episode, while Reflexion (Shinn et al., 2023) extends this across episodes by storing verbal self-critiques in an episodic buffer to guide future attempts. These methods improve response quality but do not modify the underlying memory bank.

At the *experience level*, systems move beyond per-episode feedback to accumulate reusable knowledge. ExpeL (Zhao et al., 2024) extracts natural-language insights from task trajectories and recalls them at inference time. Voyager (Wang et al., 2023) builds an ever-growing skill library from environment feedback, enabling lifelong learning in open-ended settings. O-Mem (Wang et al., 2025a) combines multiple memory types with a self-evolving mechanism for personalized agents. These systems accumulate knowledge in separate stores (experience buffers, skill libraries) but do not repair entries in the primary memory bank.

At the *policy level*, the most recent line of work directly optimizes memory management through training. MemBuilder (Shen et al., 2026) uses synthetic QA pairs as attributed dense rewards, providing finer-grained supervision than end-task accuracy alone. Memory-R1 (Yan et al., 2025) trains a memory manager to learn structured operations (ADD, UPDATE, DELETE) from end-task QA supervision with sparse rewards. Mem-α (Wang et al., 2025b) extends this to multi-component memory systems (core, episodic, semantic), training agents to manage complex architectures through interac-

tion and feedback. MemRL (Zhang et al., 2026) improves episodic memory through runtime reinforcement learning, and MEM1 (Zhou et al., 2025) jointly optimizes memory consolidation and reasoning in an end-to-end framework. These approaches improve the *policy* for using memory through training or reward optimization, but do not directly perform in-situ repair of the memory bank during construction.

In contrast, MEMMA operates at a different level: it directly repairs the *memory bank itself* during construction. By synthesizing probe QA pairs, verifying the current memory against them, and converting failures into construction-level repair actions through evidence-grounded critique and semantic consolidation, MEMMA provides dense, localized supervision before the memory is committed, without gradient-based training or separate experience stores.

B Motivating Analysis Details

B.1 Evaluation Details

We provide additional details for the preliminary study in Sec. 3.3.

Baseline Details. The three baselines are implemented by progressively enabling components of the same pipeline:

- *Static*: Uses a single-agent memory pipeline that processes each dialogue chunk sequentially, performs atomic memory edits (ADD, UPDATE, DELETE, NONE), and answers queries via one-shot top-30 retrieval based on cosine similarity. No query rewriting or strategic guidance is used.
- *Unguided Active*: Extends Static by enabling a query rewriting module that iteratively refines the retrieval query based on the retrieved evidence alone, without diagnosing what specific information is missing.
- *Strategic Active*: Further extends Unguided Active by enabling a planner that provides explicit guidance for both construction and retrieval. During construction, the planner identifies what should be retained, consolidated, or resolved. During retrieval, it diagnoses whether the current evidence is sufficient and, if not, specifies the missing information to guide the next query rewrite.

Implementation details. All three baselines use GPT-4o-mini (Hurst et al., 2024) as the backbone LLM. The retrieval budget is top-30 entries. For the two active baselines, the maximum number of query rewriting iterations is 5. All retrieval uses text-embedding-3-small (OpenAI, 2024) for embedding. We use GPT-4o-mini as the LLM judge model for calculating ACC. The full judge prompt is shown in Table 4.

Table 4: The prompt template used for LLM-as-a-Judge evaluation.

LLM-as-a-Judge Prompt

Task: Label an answer to a question as `CORRECT` or `WRONG`.

Inputs:

- **Question:** {question}
- **Gold answer:** {gold_answer}
- **Generated answer:** {generated_answer}

Instructions:

- The gold answer is usually concise. The generated answer might be longer, but be generous—as long as it touches on the same topic, count it as `CORRECT`.
- For time-related questions, be generous with format differences (e.g., "May 7th" vs "7 May").
- Provide a short explanation, then finish with `CORRECT` or `WRONG`.
- Return the label in JSON: `{"label": "CORRECT"}` or `{"label": "WRONG"}`.

B.2 Case Studies

This section provides representative examples for the preliminary study in Sec. 3.3. We organize the cases around the two pathologies of *strategic blindness*. Cases 1 and 2 illustrate *Aimless Retrieval*: active rewriting alone is not sufficient if the system cannot identify what evidence is missing. Case 3 illustrates *Myopic Construction*: local memory writing may over-store low-value details or fragment one coherent episode into multiple overlapping entries. Case 4 is a counterexample showing that *Strategic Active* is not always better, because planner guidance in the current implementation is

advisory rather than binding.

Case 1: Lexical paraphrase loop in unguided retrieval. The question is: *"When did Melanie go to the museum?"* (gold answer: *5 July 2023*). *Static* misses the evidence entirely and answers *"Not mentioned."* *Unguided Active* runs five rewrite rounds, but the queries stay close to the original wording: *"When did Melanie visit the museum?"*, *"Melanie museum trip date"*, *"Melanie's museum visit history."* None of these rewrites diagnose *what* is missing; they only rephrase *how* to ask. The retrieved set drifts toward park, beach, and camping memories—semantically adjacent but wrong. *Strategic Active* instead identifies the gap as a missing date, notes that the evidence already contains the answer, and stops rewriting. The first retrieved entry is the museum memory with the correct date.

Insight: More rewrite rounds do not help if each round is a lexical paraphrase of the last. The bottleneck is not the number of retrieval attempts but whether the system can diagnose the specific missing attribute.

Case 2: Event ambiguity requires disambiguation, not broader search. The question is: *"When is Caroline going to the transgender conference?"* (gold answer: *July 2023*). *Unguided Active* rewrites toward increasingly generic queries: *"Caroline transgender conference date"*, *"Caroline upcoming events schedule"*, *"Caroline future LGBTQ events."* The retrieved evidence mixes past LGBTQ events (e.g., a conference attended on 10 July 2023) with unrelated future activities, without resolving which conference the question refers to. *Strategic Active* narrows the gap to two specific issues: (1) the question asks about a *future* conference, not a past one, and (2) *transgender conference* and *LGBTQ conference* may refer to different events. One guided rewrite surfaces the relevant memory: Caroline is going to a transgender conference in July 2023.

Insight: When the memory bank contains multiple semantically similar events, the retrieval problem is not recall but disambiguation. Unguided rewriting broadens the search when it should narrow it.

Case 3: Local memory writing creates filler and fragmentation. During construction of the early support-group conversation, *Static* stores a greeting (*"Caroline greeted Mel"*) as a standalone entry, then repeatedly appends details about the support-group episode to a single over-packed memory. The result is a memory bank that mixes low-value filler with dense event summaries. *Strategic Active* partially addresses this: its planner flags information importance, temporal context, and redundancy, and even suggests consolidating similar sentiments. However, the final memory bank still distributes the same support-group episode across several overlapping entries—attendance, emotional reaction, and self-acceptance—because the planner's guidance is only advisory and the Memory Manager still makes atomic edits one utterance at a time.

Insight: Myopic Construction is not just about missing a planner. Even with planning, local utterance-level editing tends to produce either filler or fragmentation, because the Memory Manager cannot perform global reorganization within a single edit step.

Case 4: Planner guidance is advisory, not binding. The question is: *"What activities does Melanie partake in?"* (gold answer: *pottery, camping, painting, swimming*). Here, *Unguided Active* answers correctly, while *Strategic Active* fails. The planner guidance is reasonable: it suggests covering multiple activity types rather than focusing on one category. However, the Query Reasoner judges the evidence as ANSWERABLE and stops early. The Answer Agent then selects a partial subset of the retrieved activities (running, reading, violin, clarinet), missing the gold-answer items entirely.

Insight: Planner guidance in *Strategic Active* is a suggestion, not a constraint. When the downstream components ignore the guidance—by stopping retrieval too early or selecting from a biased subset of evidence—the system can still fail despite correct high-level reasoning. This motivates the tighter coordination mechanisms in MEMMA.

Takeaway. *Static* fails because one-shot retrieval often misses the evidence. *Unguided Active* adds active operators but still suffers from aimless rewriting and myopic construction. *Strategic Active* improves by diagnosing what is missing, but its guidance remains advisory: downstream components can still stop too early or select from partial evidence. These observations motivate the design of MEMMA, which introduces tighter coordination between the Meta-Thinker, Memory Manager, and Query Reasoner along both the forward and backward paths of the memory cycle.

C Meta-thinker Details

The Meta-Thinker π_p produces two types of guidance: construction guidance g_t^S (Sec. 4.1) and re-

trieval guidance $g^R_{q,h}$. The prompt for construction guidance is shown in Table 9, and the prompt for answerability checking (which produces $g^R_{q,h}$) is shown in Table 10.

D Query Reasoner π_r Details

The Query Reasoner π_r generates the next query u_{h+1} based on the Meta-Thinker's retrieval guidance $g^R_{q,h}$, as described in Sec. 4.1. The prompt is shown in Table 12.

E In-situ Self-Evolving Memory Construction Details

E.1 Synthetic QA Details

After each session s_τ, the system synthesizes a probe set $\mathcal{Q}_\tau = \{(q_j, y_j)\}_{j=1}^{J}$ to verify the provisional memory $M_\tau^{(0)}$. We group the synthetic probes into three types, each targeting a different failure mode in the memory cycle. Table 5 summarizes the taxonomy and provides one representative question–answer pair drawn from the generated probe data.

- *Single-hop Factoid:* Tests whether explicit facts stated in the current session s_τ are correctly stored, such as entities, attributes, or event details.
- *Multi-session Reasoning:* Tests whether the system can connect information in the current session s_τ with previously stored memory $M_{\tau-1}$, requiring cross-session integration rather than isolated fact retrieval.
- *Temporal Reasoning:* Tests whether the memory bank preserves chronological information, including relative time expressions, absolute dates, and event ordering.

These synthetic probes are designed to expose common failure modes in the memory cycle, including missing entities, incomplete event details, weak cross-session linking, and temporal inconsistency.

E.2 Prompt Details

We provide the prompt templates used in the evidence-grounded repair and semantic consolidation stages of in-situ self-evolving memory construction (Sec. 4.2). The probe generation stage follows the QA generation approach of MemBuilder (Shen et al., 2026), adapted to our memory structure; the probe types are described in Appendix E.1.

Evidence-Grounded Repair. For each failed probe, a reflection module diagnoses whether the failure reflects missing information or content that is difficult to retrieve, and proposes a candidate repair fact r_j. The prompt is shown in Tables 13 and 14.

Semantic Consolidation. Before writing repairs back to memory, each candidate fact is checked against existing entries and assigned one of three actions: `SKIP`, `MERGE`, or `INSERT`. The prompt is shown in Table 15.

F Experimental Details

F.1 Dataset Details

We evaluate on LoCoMo (Maharana et al., 2024), a benchmark for very long-term conversational memory. LoCoMo contains 10 conversation instances, each spanning roughly 600 dialogue turns and 16K tokens on average, with up to 32 sessions. The full benchmark includes 272 sessions, 5,882 dialogue turns, and 1,986 QA pairs across the 10 conversations.

Following prior work (Yan et al., 2025; Fang et al., 2025), we exclude the adversarial subset and focus on the reasoning-intensive QA setting. We use the first conversation sample (`conv-26`) as our evaluation subset. This subset contains 19 sessions and 419 dialogue turns. After excluding adversarial questions, 152 QA pairs remain, spanning four categories: single-hop (70), multi-hop (32), temporal (37), and open-domain (13). Using a fixed single-conversation subset ensures that all experiments and ablations are performed on exactly the same conversation and evaluation set.

F.2 Baseline Details

We compare MEMMA against both passive and active baselines:

- **Full Text**: concatenates the entire dialogue history into the context window and answers directly without memory construction or retrieval.
- **Naive RAG** (Gao et al., 2023): splits the dialogue into fixed-size chunks, embeds them, and retrieves the top-k chunks by cosine similarity at query time.

Table 5: Synthetic QA probe types used during probe generation in LoCoMo, with representative examples from the generated probe data.

Type	Example Question	Example Answer
Single-hop	What type of support group did I tell Melanie I attended recently?	An LGBTQ support group
Multi-hop	What is Melanie's hobby for creative expression and relaxation, and when did she create the specific piece she showed me?	Melanie paints as her hobby for creative expression and relaxation. She painted a lake sunrise last year that she showed me.
Temporal	On what date and time did I have the conversation with Melanie about attending the LGBTQ support group and my career interests in counseling?	At 1:56 pm on May 8, 2023

- **LangMem** (LangChain, 2025) provides a practical SDK for memory extraction and retrieval in agent frameworks, storing memories as structured key-value entries.

- **A-Mem** (Xu et al., 2025) dynamically organizes memories into interconnected notes following the Zettelkasten method, allowing entries to evolve as new information arrives through activation-based retrieval.

- **LightMem** (Fang et al., 2025) designs a lightweight multi-stage pipeline inspired by the Atkinson–Shiffrin model, organizing memory into sensory, short-term, and long-term stores to balance quality with computational cost.

F.3 Implementation Details

GPT-4o-mini (Hurst et al., 2024) and Claude-Haiku-4.5 (Anthropic, 2025a) are used as the default backbone for the Memory Manager, Meta-Thinker, and Query Reasoner. The iterative query refinement budget is H=3. To isolate memory construction quality from answer-generation capacity, we fix GPT-4o-mini as both the Answer Agent and the LLM judge across all experiments. For in-situ self-evolution, we generate J=5 probe QA pairs per session using Claude-Opus-4.5 (Anthropic, 2025b), retrieve top-30 entries for verification. All retrieval uses text-embedding-3-small (OpenAI, 2024).

G Impact of Probe Generation Model

G.1 Empirical Analysis.

To understand how probe quality affects in-situ self-evolving memory construction (Sec. 4.2), we vary the probe generation model among Claude-Haiku-4.5, Claude-Sonnet-4.5 (Anthropic, 2025c), and Claude-Opus-4.5. MEMMA$_{\text{LM}}$ with Claude-Haiku-4.5 as the construction backbone is used. All other settings follow Sec. 5.1.

Table 6: Impact of probe generation model on MEMMA$_{\text{LM}}$ with Claude-Haiku-4.5 as the construction backbone. Best results are in **bold**.

Probe Model	F1	B1	ACC
Claude-Haiku-4.5	44.98	**35.69**	74.34
Claude-Sonnet-4.5	43.30	32.74	74.34
Claude-Opus-4.5	**45.10**	35.66	**76.97**

Table 6 reports the results. We observe that: **(i)** *Opus achieves the best overall repair quality.* It reaches 76.97 ACC and 45.10 F1, outperforming both Haiku (74.34 ACC, 44.98 F1) and Sonnet (74.34 ACC, 43.30 F1). **(ii)** *Haiku and Sonnet match in ACC but diverge in lexical metrics.* Despite identical ACC, Haiku outperforms Sonnet in F1 (44.98 vs. 43.30) and B1 (35.69 vs. 32.74), indicating that Haiku's probes lead to higher-quality memory repairs at the token level.

We attribute this gap to differences in probe style. Sonnet tends to produce shorter, more extractive QA pairs (average answer length 11.12 words, with 136 out of 380 answers containing $\leq$ 3 words), while Haiku generates longer probes (average answer length 19.43 words) with more multi-session and temporal-reasoning questions. Opus produces probes of moderate length (average answer length 21.48 words) with the highest proportion of cross-session relational questions. Overly short probes test only surface-level keyword recall rather than cross-session consistency, so they provide weaker signals for diagnosing and repairing construction omissions.

G.2 Qualitative Examples.

To better understand the performance gap, we analyze the probe statistics and show representative examples in Table 7 and Table 8.

Two patterns stand out. First, Sonnet generates significantly more short answers: 11 one-word and 33 answers with ≤ 3 words, compared to 4 / 15 for Haiku and 4 / 8 for Opus. Sonnet's probes tend to compress answers into factoid-style keywords (e.g., "Accepted"), which tests keyword presence but not whether the memory bank can support multi-attribute reasoning. The issue is not that Sonnet hallucinates, but that it loses information by over-compressing, resulting in weaker supervision for memory repair.

Second, Sonnet's probes are dominated by single-hop questions (64 out of 95), while Haiku and Opus allocate more probes to multi-hop reasoning (25 and 26, respectively). Since single-hop probes only verify whether individual facts were stored, they are less likely to expose consolidation failures where information from different sessions was not properly linked. The higher proportion of multi-hop probes in Haiku and Opus explains their stronger repair quality.

Sonnet's single-word answer ("Accepted") only checks whether the memory bank contains a specific keyword. Haiku and Opus instead require the memory to support reasoning over multiple attributes (personal development, self-acceptance, courage), which is more likely to reveal gaps in cross-session consolidation. This explains why Sonnet, despite matching Haiku in ACC, falls behind in lexical metrics: its probes trigger fewer and shallower repairs.

H Full Details of Case Studies of MemMA

In this section, we expand the details of case studies in Sec. 5.5. We organize the cases by the two paths of the memory cycle. For the forward path, we separately examine construction-time Meta-Thinker guidance (Sec. 4.1) and iterative query refinement. For the backward path, we examine how in-situ self-evolving memory construction (Sec. 4.2) repairs the memory bank with facts that later improve downstream benchmark QA.

H.1 Forward Path: Construction-Time Meta-Thinker Guidance

To isolate the effect of construction-time meta guidance, we compare MEMMA$_{\text{SA}}$ against the ablated variant MEMMA$_{\text{SA}}$/C using Claude-Haiku-4.5 as the construction backbone. Both variants share the same query-time components, including answerability diagnosis and iterative query refinement; the only difference is whether the Meta-Thinker provides construction guidance g_t^S to the Memory Manager.

Case 1: Preserving answer-bearing visual detail. Consider the question: *"What did Caroline find in her neighborhood during her walk?"* MEMMA$_{\text{SA}}$ answers *"Caroline came across a rainbow sidewalk ..."*, whereas MEMMA$_{\text{SA}}$/C produces a vague answer about *"cool stuff"* and even confuses the walking event with a biking outing.

According to the construction trajectory, with guidance enabled, the Meta-Thinker's construction guidance g_t^S explicitly lists the answer-bearing visual object *rainbow sidewalk*, together with its supporting attributes such as *Pride Month* and *cool / vibrant / welcoming*. The Memory Manager then stores a clean entry containing the exact answer-bearing detail. Without guidance, this object detail is absent from the memory bank, so later retrieval can only recover semantically adjacent but insufficient context. This case shows that construction-time guidance preserves concrete object-level details that iterative query refinement cannot recover once they are lost.

Case 2: Preventing destructive merges. The question *"What instruments does Melanie play?"* reveals a different failure mode. MEMMA$_{\text{SA}}$ correctly answers *"the clarinet and the violin,"* whereas MEMMA$_{\text{SA}}$/C answers *"the clarinet"* and even incorrectly claims that Melanie does not play the violin.

The key difference lies in the constructed memory. With guidance, the Memory Manager stores the clarinet and violin facts as distinct entries, preserving them as parallel details. Without guidance, the Memory Manager incorrectly merges them into a conflicting entry, effectively overwriting one fact with another. This case shows that construction-time guidance also prevents harmful consolidation that would later produce factually incorrect retrieval results.

Takeaway. These cases show that the Meta-Thinker's construction guidance g_t^S improves the memory bank before retrieval begins. In particular, it preserves exact answer-bearing details, keeps semantically adjacent facts disentangled, and avoids destructive merges that would otherwise create retrieval drift or contradictions. Additional examples,

Table 7: Probe statistics across generation models. "One-word / $\leq$3" counts one-word and short ($\leq$ 3 words) answers out of 95 total per model. Question type counts follow the taxonomy in Appendix E.1.

Probe Model	Avg. Q Len.	Avg. A Len.	One-word / $\leq$3	Single-hop	Multi-hop	Temporal
Haiku	18.48	19.44	4 / 15	55	25	15
Sonnet	15.42	11.13	11 / 33	64	16	15
Opus	17.38	21.55	4 / 8	58	26	11

Table 8: Representative probe QA pairs from the same dialogue session. Sonnet's single-word answer tests only keyword presence, while Haiku and Opus require multi-attribute recall.

Model	Question	Answer
Haiku	What has the support group I attended done for my personal development and self-acceptance?	The support group has made me feel accepted and given me courage to embrace myself.
Sonnet	What did the LGBTQ support group help me feel that gave me courage to embrace myself?	Accepted
Opus	How has attending the LGBTQ support group influenced my personal growth and willingness to be open about my identity?	The support group has been a safe space that made me feel accepted, giving me the courage to embrace myself and be more open about my identity in other areas of life.

including quoted textual details (*"trans lives matter"*) and topic disentanglement (*adoption research* vs. *counseling research*), follow the same pattern.

H.2 Forward Path: Iterative Query Refinement

The second part of the forward path is Meta-Thinker-guided iterative retrieval. Here, retrieval operates over a fixed memory bank; the Meta-Thinker first judges whether the current evidence is sufficient (ANSWERABLE vs. NOT-ANSWERABLE), and the Query Reasoner then refines the query to retrieve the missing evidence.

Case 1: Recovering a temporal anchor. Consider the question: *"When did Caroline go to the LGBTQ conference?"* The Single-Agent baseline answers *"Not mentioned in the conversation,"* treating the information gap as an absence of information. By contrast, MEMMA$_{\text{SA}}$ first judges the current evidence as NOT-ANSWERABLE, noting that the problem is not the absence of all related memories, but the lack of an exact date and the ambiguity between *LGBTQ conference* and *transgender conference*. The Query Reasoner then issues increasingly targeted queries, such as asking for the specific date in July 2023 and explicitly disambiguating the two event names. The final answer becomes *"July 10, 2023."*

This case shows that the forward path does not improve performance by making better guesses; it improves performance by delaying commitment until the temporal anchor is retrieved.

Case 2: Filling a missing entity. A second example concerns the question: *"Where did Caroline move from 4 years ago?"* The LightMem baseline answers *"Her home country,"* which is directionally correct but incomplete because the benchmark expects the country name. MEMMA$_{\text{LM}}$ judges the evidence as NOT-ANSWERABLE: the relation is already known but the specific entity is missing. The Query Reasoner then rewrites the query around this information gap, first asking about Caroline's home country before she moved four years ago and then asking more explicitly for the country name. The final answer becomes *"Her home country, Sweden."*

This case is informative because the same diagnostic pattern also appears with the weaker Single-Agent backend. There, the Meta-Thinker correctly identifies the same information gap, but the backend does not contain the relevant entry. Thus, the Meta-Thinker and Query Reasoner can accurately locate the gap regardless of backend, but the final answer depends on whether the memory bank contains the answer-bearing entry.

Case 3: Recovering a missing event detail. For the question *"What did Melanie and her family see during their camping trip last year?"*, the baseline answers *"They saw amazing views,"* which is too generic to be judged correct. MEMMA$_{\text{LM}}$ instead judges the evidence as NOT-ANSWERABLE,

performs one additional refinement round, and recovers the specific answer *"Perseid meteor shower."* The key point here is that the answer already exists in the memory bank; the initial top-k retrieval simply failed to surface the decisive detail. Iterative refinement fixes this by turning a vague event description into a concrete answer.

Takeaway. Across these cases, the Meta-Thinker first identifies the information gap—a temporal anchor, a missing entity, or a specific event detail—and the Query Reasoner translates that gap into a more targeted retrieval query. The forward-path gain therefore comes not from stronger answer generation, but from refusing to answer too early and iteratively retrieving until the information gap is closed.

H.3 Backward Path: In-Situ Self-Evolving Memory Construction

To isolate the effect of in-situ self-evolution, we compare the full MEMMA$_{\text{SA}}$ against the ablated variant MEMMA$_{\text{SA}}$/E using GPT-4o-mini as the construction backbone. Both variants share the same construction-time Meta-Thinker guidance and query-time components; the only difference is whether the probe-and-repair loop (Sec. 4.2) is applied after each session. The following cases show that self-evolution improves performance not only by improving probe QA accuracy, but by writing back repair facts that later change downstream benchmark answers from incorrect to correct.

Case 1: Named-entity insertion for concert-related QA. During self-evolution of session τ=10, the probe *"What is the name of the artist who performed at Melanie's daughter's birthday concert?"* fails. Before self-evolution, the system answers that the artist is not mentioned in memory; after self-evolution, it answers *"Matt Patterson."* The repair trace shows that self-evolution inserts the following candidate repair fact:

> `ADD_FACT`: "The artist who performed at Melanie's daughter's birthday concert is Matt Patterson."

A related repair later adds another musical entity, *Summer Sounds*.

These inserted facts directly transfer to the downstream benchmark question *"What musical artists/bands has Melanie seen?"* Without self-evolution, the system answers only that *"a band performed at a show"* but cannot name it. With self-evolution, the answer becomes *"Summer Sounds"* and *"Matt Patterson."* Probe failures expose that the memory bank contains event descriptions but not the exact entity names required for downstream QA.

Case 2: Restoring a distinctive event detail. During self-evolution, the probe *"What was Melanie's most memorable camping experience with her family?"* fails. The system produces a generic answer about roasting marshmallows and telling stories, missing the distinctive detail. Self-evolution repairs this by inserting a new event fact centered on the Perseid meteor shower.

This repair transfers to the downstream benchmark question *"What did Melanie and her family see during their camping trip last year?"* Without self-evolution, the downstream answer remains generic and mentions only ordinary camping activities. With self-evolution, the system retrieves and outputs the specific event detail *"Perseid meteor shower."* This case shows that self-evolution sharpens vague event memories into distinctive, answerable ones.

Case 3: Completing a partial evidence cluster. During self-evolution, the probe *"What new pottery project did Melanie recently finish, and what was her earlier pottery creation?"* fails. The system can only answer part of the question and leaves the pottery objects underspecified. Self-evolution repairs this by writing back the missing facts about a *colorful bowl* and an earlier *black and white bowl*.

These repairs transfer to downstream benchmark questions such as *"What types of pottery have Melanie and her kids made?"* and *"What kind of pot did Mel and her kids make with clay?"* Without self-evolution, the model answers only with generic descriptions such as *"pots"* or *"various pottery projects."* With self-evolution, the final answer becomes object-level and complete: bowls, a cup with a dog face, a colorful bowl, and a black-and-white bowl. This case illustrates that self-evolution does not only insert isolated facts; it can also complete a sparse local evidence cluster so that the whole topic becomes answerable.

Takeaway. Across these cases, in-situ self-evolution improves performance by turning vague, generic, or partially correct memory regions into retrieval-friendly, answerable memory units. More specifically, it works through three recurring repair mechanisms: (i) named-entity insertion, (ii) distinctive event-detail sharpening, and (iii) partial evidence completion. The key point is that probe failures do not remain local. Instead, they are con-

verted into evidence-grounded repair actions that transfer directly to downstream benchmark performance.

I Information about AI Assistants

We used an OpenAI LLM (GPT-5.4) as a writing and formatting assistant. In particular, it helped refine grammar and phrasing, improve clarity, and suggest edits to figure/table captions and layout (e.g., column alignment, caption length, placement). The LLM did not contribute to research ideation, experimental design, implementation, data analysis, or technical content beyond surface-level edits. All outputs were reviewed and edited by the authors, who take full responsibility for the final text and visuals.

Table 9: The prompt template used for Meta-Thinker construction guidance.

Meta-Thinker Construction Guidance Prompt

Role: You are a quality-control checker for a memory construction system. Given one conversation utterance, list every distinct factual statement it contains. Each fact must be an atomic, self-contained statement that could answer a WHO/WHAT/WHEN/WHERE/HOW MANY question.

Rules:

- Extract EVERY fact—do not skip anything. Err on the side of over-extraction.
- Use the speaker's exact words for names, objects, dates, places, and quantities.
- One fact per line. Do NOT merge multiple facts into one line.
- Prefix each fact with the correct speaker name.
- Do NOT interpret emotions, themes, values, or symbolism.
- Do NOT paraphrase—preserve the original phrasing.

Output format:

```
FACTS:
- [Speaker] fact 1
- [Speaker] fact 2
- ...
```

Table 10: The prompt template used for Meta-Thinker answerability checking (Part 1).

Meta-Thinker Answerability Checking Prompt (Part 1)

Role: You are a Meta-Thinker agent for answerability checking in a memory-augmented QA system.

Inputs:

- Question
- Retrieved memories grouped by speaker (memory_id, timestamp, snippet)
- Previous queries

Goal: Minimize false `NOT_ANSWERABLE` while staying evidence-grounded.

Blocking-gap test: Return `NOT_ANSWERABLE` only if a missing fact or unresolved contradiction would CHANGE the final short answer. If a best-supported answer is already stable, return `ANSWERABLE`.

Granularity policy:

- Time questions: require exact day/date only if the question explicitly asks for it; otherwise accept the best unambiguous granularity.
- Who/what/which: one clearly supported entity is enough unless the question explicitly requests exhaustive output.
- Contradictions: only contradictions that change the final answer are blocking.

Table 11: The prompt template used for Meta-Thinker answerability checking (Part 2).

Meta-Thinker Answerability Checking Prompt (Part 2)

Anti-stall: If $\geq$3 previous queries were attempted and the same non-blocking gap repeats, prefer `ANSWERABLE` at best-supported granularity.

Output format:

```
<decision>ANSWERABLE|NOT
ANSWERABLE</decision>
<reason>1-3 sentences about the asked
slot only.</reason> <key-gaps>Ranked
bullets if NOT-ANSWERABLE; NONE
otherwise.</key-gaps>
<missing-speaker> speaker-1
| speaker-2 | both |
unknown</missing-speaker>
<time-need>Required granularity and
missing anchor, or NONE.</time-need>
<retrieval-guidance>Goal, suggested
queries, keywords, constraints,
avoid terms.</retrieval-guidance>
```

Table 12: The prompt template used for Query Reasoner π_r to generate orthogonal query u_{h+1}.

Orthogonal Query Generation Prompt

Role: You are an expert Query Rewriter for conversation memory retrieval.

Inputs:

- The question and Meta-Thinker diagnosis (top gap, missing speaker, time need, constraints, avoid terms)
- Previous queries and retrieval trace (query $\rightarrow$ retrieved memory IDs)

Task: Generate EXACTLY ONE new retrieval query that targets the top gap and is maximally likely to retrieve new evidence.

Hard rules:

- Do NOT repeat any previous query verbatim or near-verbatim.
- MUST target the top gap only (do not broaden to multiple gaps).
- MUST include all constraints (entity + time/version) exactly as provided.
- If time need is provided, include both the relative phrase (e.g., "last year") and the computed absolute time (e.g., "2021").
- MUST avoid exhausted terms.
- Prefer disambiguation queries if contradiction exists.
- If missing speaker is specified, phrase the query to target that speaker's perspective.

Output format (JSON):
`{"rewritten_query": "...", "strategy": "...", "target_speaker": "..."}`

Table 13: The prompt template used for evidence-grounded repair in self-evolution (Part 1).

Evidence-Grounded Repair Prompt (Part 1)

Role: You are a memory-repair assistant for a two-speaker conversation memory system.

Inputs:

- Question, Gold Answer, Model Answer
- Retrieved evidence snippets (from current memory; may be irrelevant if the info is missing)

Task: Decide whether to add one fact to memory so the system can answer correctly next time.

Decision rules (priority order):

- If Gold Answer is unanswerable, output `NOOP`.
- If Gold Answer is answerable and Model Answer is wrong or incomplete, output `ADD_FACT`. The fact should capture the key information from the Gold Answer.
- If Gold Answer and Model Answer are essentially equivalent, output `NOOP`.

Quality rules:

- Fact must be concrete, specific, and retrieval-friendly (include names, dates, details).
- Preserve relative date expressions verbatim; do NOT convert to absolute dates.
- Assign `target_speaker` based on who the fact is about.

Table 14: The prompt template used for evidence-grounded repair in self-evolution (Part 2: output format and examples).

Evidence-Grounded Repair Prompt (Part 2)

Output format (JSON):
`{"op": "ADD_FACT | NOOP", "target_speaker": "speaker_a | speaker_b", "fact": "...", "evidence_span": "...", "confidence": 0.0, "reason": "..."}`

Example — information missing from memory:
`Question:` What does the necklace from Caroline's grandma symbolize?
`Gold:` Love, faith, and strength.
`Model:` The memories do not contain information about a necklace.
`Output:` `{"op": "ADD_FACT", "fact": "Caroline's grandma gave her a necklace from Sweden that symbolizes love, faith, and strength", "evidence_span": "", "confidence": 0.88}`

Example — unanswerable gold answer:
`Question:` What is Melanie's passport number?
`Gold:` Not mentioned in the conversation.
`Output:` `{"op": "NOOP"}`

Table 15: The prompt template used for semantic consolidation (deduplication) in self-evolution.

Semantic Consolidation Prompt

Role: You are a memory deduplication assistant.

Inputs:

- A new proposed fact to be added to memory
- One or more existing memory entries that are semantically similar (with similarity scores)

Decision:

- `SKIP`: An existing entry already fully covers the new fact.
- `MERGE`: The new fact describes the same event/attribute as an existing entry and adds a missing detail. Combine into one entry.
- `INSERT`: The new fact is about a different topic, event, or time period.

Critical rule: Different dates or different occurrences of the same activity = `INSERT`, never `MERGE`. Only merge when both texts refer to the exact same single event at the same time.

Output format (JSON):
`{"action": "SKIP | MERGE | INSERT", "merge_target_index": -1, "merged_fact": "", "reason": "..."}`