RecMem: Recurrence-based Memory Consolidation for Efficient and Effective Long-Running LLM Agents

Zijie Dai[1] **Shiyuan Deng**[2*] **Sheng Guan**[3] **Yizhou Tian**[1]
Xin Yao[4] **Xiao Yan**[5] **James Cheng**[1]

[1]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[3]School of Computer Science, Beijing University of Posts and Telecommunications
[2]Huawei Cloud, [4]Huawei Theory Lab, [5]Institute for Math and AI, Wuhan University
caiusdai@link.cuhk.edu.hk dengshiyuan@huawei.com

Abstract

Memory systems often organize user-agent interactions as retrievable external memory and are crucial for long-running agents by overcoming the limited context windows of LLMs. However, existing memory systems invoke LLMs to process every incoming interaction for memory extraction, and such an *eager memory consolidation* scheme leads to substantial token consumption. To tackle this problem, we propose *RecMem* by rethinking when memory consolidation should be conducted. RecMem stores incoming interactions in a subconscious memory layer and encode them using lightweight embedding models for retrieval. LLMs are only invoked to extract episodic and semantic memory when sustained recurrence are observed for semantically similar interactions. Such *recurrence-based consolidation* works because these interactions correspond to a semantic cluster with rich information and thus are worth extraction and summarization. To improve accuracy, RecMem also incorporates a semantic refinement mechanism that recovers the fine-grained facts omitted by memory extraction. Experiments show that RecMem reduces the memory construction token cost of three SOTA memory systems by up to 87% while exceeding their accuracy.

1 Introduction

Large Language Models (LLMs) have demonstrated strong capabilities across a wide range of tasks (Guo et al., 2024; Shao et al., 2024). However, enabling LLMs to function as long-running agents requires accumulating experience over extended user-agent interactions (Jiang et al., 2025). In practice, this is hindered by two critical limitations: current LLMs cannot retain information beyond their limited context windows (Liu et al., 2025), and they often under-utilize relevant evidences even if they are present in long inputs due to the lost-in-the-middle effect (Liu et al., 2023).

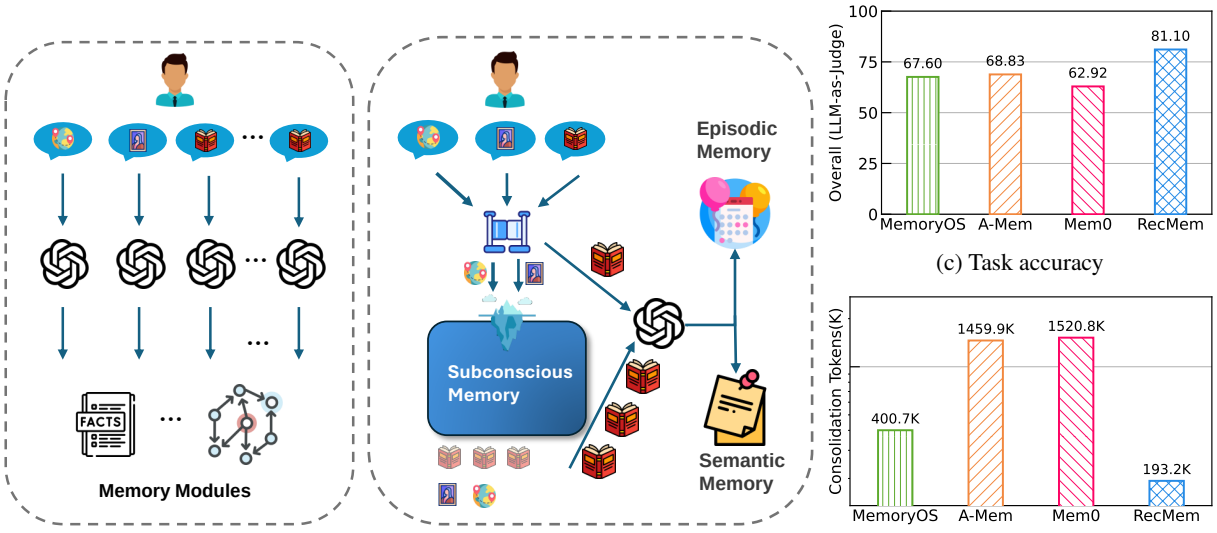
To address these limitations, memory systems emerge as an essential component for building long-running LLM agents (Jiang et al., 2025; Zhang et al., 2024), and many solutions have been proposed with different memory structures and memory extraction methods (Xu et al., 2025b; Chhikara et al., 2025; Rezazadeh et al., 2025; Packer et al., 2024; Maharana et al., 2024). For example, Zep (Rasmussen et al., 2025) constructs temporal knowledge graphs by abstracting relational triplets from interactions; Mem0 (Chhikara et al., 2025) extracts atomic facts from interactions for similarity-based retrieval; A-Mem (Xu et al., 2025b) organizes interactions as connected notes, and a note can update the contents of its neighbors.

Despite the differences in existing memory systems, we observe that they all adopt an *eager memory consolidation* scheme. In particular, for every incoming user-item interaction, they invoke LLMs to extract facts and merge these facts with existing memory contents. This scheme avoids missing information in the interactions but incurs substantial token cost for memory construction, as shown in Figure 1(d), which makes it expensive to utilize these memory systems in practice. We argue that running LLM-based memory consolidation for every interaction is an overkill. For instance, some interactions may convey little information or contain noise, while some interactions are not related to existing ones and can be queried directly without consolidation. Hence, it is possible to reduce the memory construction cost by choosing when to conduct memory consolidation more judiciously.

Similar insights emerge from cognitive science. The multi-store theory (Atkinson and Shiffrin, 1968) and the Complementary Learning Systems framework (Kumaran et al., 2016; O'Reilly et al., 2014; McClelland et al., 1995) both converge on a common principle: isolated experiences remain

* Dr. Shiyuan Deng is the corresponding author.

Findings of the Association for Computational Linguistics: ACL 2026, pages 32353–32376



(a) Eager memory consolidation (b) Recurrence-based consolidation (ours) (c) Task accuracy (d) Memory construction cost

Figure 1: Comparing RecMem with existing memory systems. (a) Existing systems conduct eager memory consolidation for every incoming interaction; (b) our RecMem conducts recurrence-based consolidation selectively from a subconscious memory; (c)-(d) task accuracy and memory construction cost on the LoCoMo benchmark.

in transient or rapidly-encoded stores, and only repeated or recurring patterns drive consolidation into stable long-term memory. This principle directly motivates RecMem's recurrence-driven consolidation scheme.

Motivated by these insights, we propose RecMem, an efficient memory system for long-running agents that conducts fewer LLM-based memory consolidations in a *recurrence-driven* manner. In particular, RecMem introduces a subconscious memory layer that buffers the raw user-agent interactions via lightweight embeddings, enabling cost-effective retrieval without invoking LLMs. Memory consolidation is only conducted when an incoming interaction can find a sufficient number of semantically similar or related interactions in the subconscious memory, and LLMs are utilized to extract episodic summaries and semantic facts from these interactions. This works because these interactions form a semantic cluster with rich information that is worth memory consolidation and resembles generating long-term memory from transient memory in cognitive science.

RecMem also incorporates a *semantic refinement* mechanism to improve accuracy. Specifically, LLM-based extraction, especially event-level episodic summarization, may omit fine-grained but query-critical details, leading to lossy long-term memory. Our semantic refinement revisits the raw interactions associated with each episodic memory, extracts the missing and persistent facts that are not captured by the episodic memory, and distills them into a semantic memory to avoid information loss.

We empirically evaluate RecMem on two challenging long-term memory benchmarks (i.e., LoCoMo (Maharana et al., 2024) and LongMemEval-S (Wu et al., 2025)) and compare with three SOTA memory systems (i.e., Mem0 (Chhikara et al., 2025), A-Mem (Xu et al., 2025b), and MemoryOS (Kang et al., 2025)). The results show that RecMem yields higher question answering accuracy than all baselines on both datasets while drastically reducing the token cost for memory construction. In particular, on the LoCoMo benchmark in Figure 1(d), RecMem reduces the token consumption by up to 7.8x over the baselines. Moreover, RecMem's query-time token cost remains comparable to existing memory systems, so the construction-time savings translate into lower end-to-end cost over long interaction histories.

Our contributions are summarized as follows:

- We identify a fundamental inefficiency in existing LLM memory systems, i.e., *eager memory consolidation* for every interaction leads to a high memory construction token cost.
- Inspired by cognitive science, we propose *recurrence-based consolidation* to save token cost by conducting memory consolidation only when an incoming interaction can find a sufficient number of semantically similar or related interactions.
- We present RecMem, a three-tier memory ar-

chitecture that realizes this paradigm. Combining a lightweight subconscious store with a novel semantic refinement mechanism, RecMem achieves high accuracy while substantially reducing token cost.

2 Preliminaries

2.1 Problem Setting: Conversational Memory

Recent work on LLM-based agents increasingly focuses on conversational memory, where the agent accumulates information through long-term, multi-turn interactions (Hu et al., 2025; Xu et al., 2025a; Maharana et al., 2024). Formally, we denote the interaction history available at time step t as a sequence $\mathcal{O}_{1:t} = \{o_1, \ldots, o_t\}$. Each interaction unit o_t is defined as a tuple:

$$o_t = (s_t, x_t, \tau_t) \tag{1}$$

where $s_t \in \{\text{USER}, \text{ASSISTANT}\}$ represents the speaker role, x_t denotes the message content, and τ_t is the timestamp. Given a query q, the objective is to retrieve relevant evidence from an external memory derived from $\mathcal{O}_{1:t}$ to support reasoning and response generation.

Although conversational settings may appear more specific than general memory scenarios, they capture a fundamental property of real-world deployment: information arrives streamingly over time, and the agent must continually manage an ever-growing interaction history to support future queries and reasoning (Zhang et al., 2024). This formulation contrasts with retrieval-augmented generation (RAG), which typically assumes static or pre-ingested knowledge sources (Lewis et al., 2021; Han et al., 2025). In conversational memory, the key challenge is not retrieval, which can largely leverage existing techniques, but how the system constructs and updates the underlying memory from ongoing interactions in an online manner.

2.2 Memory Systems

We focus on training-free, text-based external memory systems for LLM agents in streaming conversational settings. For brevity, we refer to such systems as *memory systems* in the remainder of this paper. Parametric memory approaches (Fang et al., 2025; Wang et al., 2025a) require retraining or architectural modification to absorb new information and are thus less applicable in our setting (Hu et al., 2025), while RL-based methods (Yan et al., 2025; Wang et al., 2025b) are orthogonal to our focus, as they operate on top of a given memory architecture.

Most existing memory systems construct long-term memory by incrementally transforming incoming interactions (or short windows thereof) into retrievable memory units, such as summaries (Kang et al., 2025; Packer et al., 2024; Zhong et al., 2023), atomic facts (Chhikara et al., 2025; Wang and Chen, 2025), or structured nodes (e.g., graphs/trees) (Hogan et al., 2021; Rasmussen et al., 2025; Rezazadeh et al., 2025), and then rely on similarity-based retrieval or hybrid search (Kang et al., 2025; Rasmussen et al., 2025) to supply evidence at query time. We defer a detailed taxonomy of memory representations, retrieval mechanisms, and construction pipelines to Appendix A.

3 The RecMem Framework

3.1 Overview

RecMem is a three-tier memory system guided by the principle that not all interactions warrant LLM-level consolidation. Incoming messages are first organized as atomic interaction units and written to a *subconscious* store with only lightweight structuring and vectorization, making the raw interaction history directly accessible through embedding-based retrieval (§3.2). Building on this store, RecMem performs *recurrence-based consolidation*: instead of consolidating every turn, it invokes LLM-based processing only when the system observes clear evidence that similar interaction content recurs, thereby reserving LLM invocation for cases where aggregation is likely to be beneficial. Once triggered, RecMem produces an *episodic* abstraction over the selected turns (§3.3), and then applies *semantic refinement* to recover fine-grained, reusable facts that may be omitted by episodic abstraction, grounded in the episode and its underlying interactions (§3.4). At query time, RecMem retrieves a small budget of items from the subconscious, episodic, and semantic stores, and answers by conditioning the LLM on the merged context (§3.5).

Our use of episodic and semantic memory follows the convention in previous LLM memory literature (Li and Li, 2024; Wang and Chen, 2025). Specifically, episodic memory in RecMem stores temporally anchored event narratives, which are coherent summaries of how a topic evolves across multiple interaction turns, with explicit time grounding. Semantic memory stores atomic facts

about general knowledge, user preferences, constraints, and entity relations.

RecMem's design mirrors human memory: most experiences remain unconsolidated unless repeatedly activated (Atkinson and Shiffrin, 1968; O'Reilly et al., 2014; McClelland et al., 1995). By avoiding eager LLM-based consolidation of transient interactions, RecMem substantially reduces token consumption while preserving both event-level coherence and stable user-centric knowledge as memories. To facilitate understanding, Appendix B provides a minimal running example that walks through the memory ingestion workflow.

3.2 Subconscious Memory

The subconscious memory manager maintains a faithful record of interaction history at minimal computational cost. A critical design consideration here is the granularity at which conversational information is represented. Existing systems adopt diverse ingestion strategies, ranging from processing individual messages (Xu et al., 2025b; Wang and Chen, 2025; Rasmussen et al., 2025) or interaction pairs (Chhikara et al., 2025) to accumulating larger, fixed-size context buffers (Kang et al., 2025; Packer et al., 2024). Static grouping or buffering may conflate temporally adjacent but semantically unrelated topics, diluting the specificity of embeddings. Conversely, ingesting messages in isolation risks fragmenting the semantic context, as an assistant's response often relies heavily on the preceding user query for its meaning.

To address these issues, RecMem treats each *message exchange* (a user-assistant turn) as an atomic unit. Formally, we define a multi-turn conversation between a user and an assistant with t turns as

$$\mathcal{H}_t = \big(u_1, u_2, \ldots, u_t\big), \tag{2}$$

$$u_i = \big(m_i^{\text{usr}}, m_i^{\text{ast}}, \tau_i\big). \tag{3}$$

Here, u_i represents an interaction unit at turn i, composed of the user message m_i^{usr}, the assistant response m_i^{ast}, and a timestamp τ_i. The history $\mathcal{H}_t$ is a time-ordered sequence of these units.

As interaction units arrive in a streaming manner, each new message turn u_i is processed independently. We formally define the constructed subconscious memory unit s_i as:

$$s_i = (v_i, u_i) \quad \text{where} \quad v_i = \Phi(u_i). \tag{4}$$

These units, computed via the dense vector encoder $\Phi(\cdot)$, are immediately indexed into the subconscious memory store $\mathcal{S}_{\text{sub}}$. This store is implemented as a vector database to support efficient semantic retrieval and incremental updates without the need for batching or access to future context. This fine-grained representation encourages focused semantic embeddings at the level of individual interaction units, making it well-suited for streaming settings.

Recurrence-based Memory Consolidation Echoing cognitive principles (Atkinson and Shiffrin, 1968; O'Reilly et al., 2014; McClelland et al., 1995), we propose *recurrence-based consolidation*: raw interactions are retained in the subconscious buffer, with LLM-based abstraction triggered only when retrieval signals indicate sustained recurrence. Specifically, for a new arriving unit $s_i = (v_i, u_i)$, the system queries $\mathcal{S}_{\text{sub}}$ to retrieve the set $\mathcal{N}_i$ containing the top-k units ranked by cosine similarity to v_i. We then filter these candidates to define the *relevant set* based on strict semantic proximity:

$$\mathcal{R}_i = \{\, s_j \in \mathcal{N}_i \mid \cos(v_i, v_j) \geq \theta_{\text{sim}} \}. \tag{5}$$

Consolidation is triggered only if the relevant set size meets a recurrence count threshold (i.e., $|\mathcal{R}_i| \geq \theta_{\text{count}}$). In such cases, the cluster $\mathcal{C}_i = \mathcal{R}_i \cup \{s_i\}$ is promoted to higher-level memory modules including episodic memory and semantic memory; otherwise, s_i remains in $\mathcal{S}_{\text{sub}}$. This ensures consolidation is conducted exclusively in memories with demonstrated long-term recurrence.

3.3 Episodic Memory

Episodic memory captures event-level structure across multiple turns. To ensure memory remains compact, RecMem adopts a merge-first strategy. Upon the arrival of a subconscious unit $s_i = (v_i, u_i)$, we retrieve the nearest neighbor episode $E^\star$ from the episodic store $\mathcal{S}_{\text{epi}}$. Let $v_{E^\star} = \Phi(E^\star)$ denote the embedding of this episode. We strictly enforce an in-place update if semantic similarity permits:

$$\begin{aligned} E^\star &\leftarrow \text{LLM}_{\text{merge}}(E^\star, u_i) \\ \text{if} \quad & \cos(v_i, v_{E^\star}) \geq \theta_{\text{sim}}, \end{aligned} \tag{6}$$

where $\text{LLM}_{\text{merge}}$ integrates the content of the new turn u_i into the narrative of $E^\star$.

Without such a merge-first step, each recurrence-triggered consolidation on a topic would produce a

fresh episode in parallel with existing ones on the same topic, fragmenting the episodic representation of an evolving thread across multiple disconnected entries. Merge-first collapses these into a single continually-updated narrative, keeping the episodic store compact and the per-topic narrative coherent as the conversation evolves.

If merging is not applicable, the unit waits for the **recurrence-based consolidation** trigger. Given the triggered cluster $\mathcal{C}_i$ (from §3.2), we extract the interaction units $\mathcal{U}_\mathcal{C} = \{u_j \mid (v_j, u_j) \in \mathcal{C}_i\}$. We then sort these units by their timestamps to form a temporal sequence $U_i^{\text{seq}} = (u^{(1)}, \ldots, u^{(|\mathcal{C}_i|)})$ for episodic memory consolidation. The consolidation prompt is designed for *inductive organization* rather than simple summarization. The LLM processes the formatted sequence to synthesize coherent narratives, segmenting disparate sub-topics if necessary:

$$\mathcal{M}_i^{\text{epi}} = \text{LLM}_{\text{epi}} \left(\bigoplus_{k=1}^{|\mathcal{C}_i|} \text{Fmt}(u^{(k)}) \right). \quad (7)$$

Here, $\text{Fmt}(\cdot)$ denotes a fixed template that formats each interaction unit into a textual representation. The output $\mathcal{M}_i^{\text{epi}}$ is a set of new episodic units; each episode $E \in \mathcal{M}_i^{\text{epi}}$ is then encoded via $\Phi(\cdot)$ and stored in $\mathcal{S}_{\text{epi}}$. We provide a complete prompt list in Appendix F for reference.

3.4 Semantic Memory

Semantic memory complements episodic memory by storing fine-grained facts that may be missed by event-level summaries. It also mitigates a side effect of the merge-first strategy in episodic memory: as an episode absorbs more turns through repeated merges, its summary necessarily becomes broader and more abstract, which can dilute its retrieval precision for queries that target a specific detail buried within that episode. Semantic memory counteracts this by storing the same details as independent, narrowly-scoped entries, so that precise factual queries can hit them directly without having to surface the entire encompassing episode. In RecMem, we construct this memory layer through a process called **Semantic Refinement**. By strictly tying semantic extraction to episodic construction, this mechanism ensures that facts remain grounded in the current episodic context while explicitly recovering precise details that were abstracted away.

Formally, when a new episode $E \in \mathcal{M}_i^{\text{epi}}$ is generated from the source interaction units $\mathcal{U}_\mathcal{C}$, we first retrieve related existing semantic facts to provide historical context:

$$\mathcal{V} = \text{TopK}_k\big(\mathcal{S}_{\text{sem}}, \Phi(E)\big). \quad (8)$$

We then employ an LLM-based refiner to deduce new facts. Conditioned on the raw interaction units $\mathcal{U}_\mathcal{C}$, the episodic summary E, and the retrieved facts $\mathcal{V}$, the model is instructed to perform two parallel tasks: (1) **Detail Recovery**, which scans the raw texts in $\mathcal{U}_\mathcal{C}$ to identify critical entities omitted by the summary E; and (2) **Fact Maintenance**, which prevents redundancy by filtering out known information in $\mathcal{V}$ while updating evolving user states (e.g., preference changes).

The extraction process is formulated as:

$$\mathcal{M}_i^{\text{sem}} = \text{LLM}_{\text{refine}}\big(E, \mathcal{U}_\mathcal{C}, \mathcal{V}\big). \quad (9)$$

Each extracted fact $f \in \mathcal{M}_i^{\text{sem}}$ is stored as an independent entry to preserve retrieval specificity. This design reduces redundancy and enables incremental updates of user facts while keeping retrieval efficient.

3.5 Question Answering

To generate an answer, RecMem first encodes the user query q into a vector representation $v_q = \Phi(q)$ and retrieves the most relevant entries from the subconscious ($\mathcal{S}_{\text{sub}}$), episodic ($\mathcal{S}_{\text{epi}}$), and semantic ($\mathcal{S}_{\text{sem}}$) stores. To manage the context window efficiently while ensuring diverse coverage, we enforce a fixed subconscious retrieval budget and *couple* the episodic and semantic budgets by setting $k_{\text{sem}} = 2k_{\text{epi}}$, yielding three context sets: $\mathcal{K}_{\text{sub}}$, $\mathcal{K}_{\text{epi}}$, and $\mathcal{K}_{\text{sem}}$. The final answer is then generated by conditioning the LLM on the retrieved contexts alongside the original query.

3.6 Discussions

Setting the hyper-parameters RecMem's consolidation behavior is controlled by two key hyperparameters, i.e., similarity threshold θ_{sim} for relevant interactions and recurrence threshold θ_{count} to trigger consolidation. Larger θ_{sim} and θ_{count} make consolidation more conservative and favor more frequent patterns, while lower thresholds make consolidation more active and improve coverage for subtle details. According to empirical experiences, we recommend θ_{sim}=0.7, θ_{count}=5 for casual open-ended settings and θ_{sim}=0.6, θ_{count}=4 for longer and task-oriented interactions. For question answering, we fix the aggregate retrieval budgets across the memory layers and use k_{sub}=10,

k_{epi}=5, and k_{sem}=10 (i.e., k_{sem}=2k_{epi}) by default. Sensitivity experiments for the hyperparameters are conducted in Appendix C.

Robustness of threshold choice A natural concern is whether RecMem's performance depends on precise threshold calibration. Our sensitivity analysis in Appendix C shows that this is not the case: overall accuracy varies smoothly and is within a narrow band around the recommended defaults, so performance does not hinge on selecting a brittle operating point. Within this robust range, the thresholds instead serve as a strategic dial between memory selectivity and consolidation sensitivity. Higher values of θ_{sim} and θ_{count} render RecMem more *conservative*, prioritizing high-confidence patterns suitable for casual open-ended conversations where signal is sparse and noise filtering matters. Conversely, lower thresholds make the system more *active* in consolidation, ideal for task-completion workflows where capturing subtle details is critical.

Generality of recurrence-based consolidation Although RecMem organizes the episodic memory and semantic memory as flat entries for similarity-based retrieval, recurrence-based consolidation is a general idea and not limited to specific memory structures. The key to recurrence-based consolidation is to utilize a cheap subconscious memory to buffer the incoming interactions and trigger consolidation for higher memory layers based on recurrence, and the higher memory layers can also adopt alternative structures (e.g., knowledge graph).

4 Experimental Evaluation

4.1 Experiment Settings

To ensure a fair and standardized comparison, we strictly adhere to the incremental evaluation protocol established in prior studies (Chhikara et al., 2025; Xu et al., 2025b; Kang et al., 2025). In this setting, message turns are streamed sequentially into the memory system to mimic the natural flow of ongoing dialogues (Hu et al., 2025), followed by multi-round query sessions.

Datasets We evaluate RecMem on two English benchmarks selected to represent distinct interaction modalities: social companionship and long-context task completion. **LoCoMo** (Maharana et al., 2024) features companion-style, life-sharing dialogues, consisting of 10 multi-session conversations (avg. 16k tokens) with questions that probe reasoning over evolving personal history. In contrast, **LongMemEval-S** (Wu et al., 2025) focuses on agentic, task-oriented interactions with substantially longer contexts. Comprising 500 conversations averaging 115k tokens, it poses a rigorous test for memory systems under realistic, high-load user-assistant workflows. Detailed statistics and question types of these two datasets are provided in Appendix D.

Baselines We compare RecMem against various types of representative baselines:

- **Full Context**, which feeds all historical interactions to the LLM for answering each question.
- **Naive RAG**, a standard RAG baseline that segments the interactions into chunks and retrieves the relevant chunks based on embedding similarity. We employ a chunking strategy that respects message integrity (see Appendix E.1).
- **Mem0** (Chhikara et al., 2025) employs a fact-extraction pipeline to dynamically extract salient information from interactions and manage memory consistency via LLM-based update operations (e.g., add,update, delete).
- **A-Mem** (Xu et al., 2025b), an agentic memory system inspired by the Zettelkasten note-taking method (Kadavy, 2021; Ahrens, 2017), which organizes the interactions as discrete "memory notes" that are connected via entity linking to facilitate associative retrieval.
- **MemoryOS** (Kang et al., 2025), an OS-inspired hierarchical framework that manages information via short-term, mid-term, and long-term memory tiers. It also incorporates a dedicated module to maintain evolving user and agent personas to enable personalized interactions.

Performance Metrics We compare RecMem with the baselines along two dimensions.

- **Question answering accuracy.** We report accuracy as the fraction of questions answered correctly. Following (Chhikara et al., 2025), we use *GPT-4o-mini* as an LLM judge and treat its judgment score as the primary metric. We prioritize this semantic evaluation over token-overlap metrics like F1 score, which can under-estimate correctness for open-ended generation with paraphrases. For completeness, we also report F1 and a comparison in Appendix E.4. All reported task scores are averaged over three runs.
- **Computation efficiency.** We measure LLM to-

Category	FullContext	Naive RAG	MemoryOS	Mem0	A-Mem	RecMem (Ours)
GPT-4O-MINI						
Multi-Hop	69.86	54.61	58.87	52.84	52.13	**72.70**
Temporal	52.96	24.30	44.24	52.02	**61.99**	57.32
Open Domain	55.21	51.04	46.88	37.50	30.21	**58.33**
Single-Hop	**90.01**	57.55	74.55	65.52	66.83	79.79
Overall	**76.43**	49.68	63.64	58.64	60.84	72.47
Construct Token(K)	0.0	0	429.7	1233.5	1143.3	202.4
Query Token(K)	31.5	6.23	4.88	1.99	3.04	2.73
GPT-4.1-MINI						
Multi-Hop	**82.62**	58.87	66.31	58.16	59.93	78.37
Temporal	**79.13**	33.96	47.66	63.86	72.90	75.39
Open Domain	**57.29**	50.00	55.21	44.79	42.71	**57.29**
Single-Hop	**90.84**	61.24	77.05	66.23	73.25	86.92
Overall	**84.18**	54.42	67.60	62.92	68.83	81.10
Construct Token(K)	0.00	0.00	400.7	1520.80	1459.93	193.2
Query Token(K)	31.52	6.28	5.04	2.11	5.56	2.75

Table 1: Results on the LoCoMo benchmark. Bold and underline mark the best and second accuracies.

Category	FullContext	Naive RAG	MemoryOS	Mem0	A-Mem	RecMem (Ours)
GPT-4O-MINI						
Single-User	44.29	85.71	**97.14**	95.71	92.54	91.43
Single-Assistant	80.36	83.93	89.29	55.36	**98.21**	85.71
Single-Preference	53.33	46.67	**70.00**	**70.00**	36.67	46.67
Temporal-Reasoning	34.59	41.35	48.87	54.14	44.36	**61.65**
Knowledge-Update	55.13	65.38	73.08	69.23	70.51	**80.77**
Multi-Session	35.34	52.63	**58.65**	57.89	53.38	56.39
Overall	45.60	59.40	67.8	64.00	63.80	**69.20**
Construct Token(K)	0.00	0.00	705.34	1244.87	1180.23	329.55
Query Token(K)	112.30	11.92	8.89	1.90	13.10	5.63
GPT-4.1-MINI						
Single-User	91.43	85.71	94.29	95.71	95.71	**95.71**
Single-Assistant	**100.00**	82.36	89.29	50.00	**100.00**	96.43
Single-Preference	56.67	83.33	**100**	90.00	63.33	86.67
Temporal-Reasoning	48.12	45.86	54.89	62.41	52.63	**68.42**
Knowledge-Update	75.64	75.64	80.77	74.36	**82.05**	75.64
Multi-Session	53.38	63.91	67.67	**69.92**	61.65	65.41
Overall	66.20	67.00	74.4	71.20	71.60	**76.80**
Construct Token(K)	0.00	0.00	669.22	1626.54	1264.25	365.49
Query Token(K)	112.25	11.93	9.19	2.00	15.46	5.89

Table 2: Results on the LongMemEval-S benchmark. Bold and underline mark the best and second accuracies.

ken usage (input plus output) in two phases: (1) *construction cost*, averaged per conversation during memory ingestion, and (2) *query cost*, averaged per question during answering.

Implementation Details To evaluate the generalization of our approach, we conduct experiments using two distinct LLM backends: *GPT-4o-mini* and *GPT-4.1-mini*. To ensure fair comparison, for any given benchmark result, RecMem and all baselines share the identical underlying model version. For LLM calls, we set temperature=0.0 and utilize *text-embedding-3-small* for vector embedding generations. For RecMem, we configure the recurrence-based consolidation thresholds to adapt to the distinct interaction densities of each benchmark: $\theta_{sim} = 0.7, \theta_{count} = 5$ for LoCoMo, and $\theta_{sim} = 0.6, \theta_{count} = 4$ for LongMemEval-S. More detailed experiment settings are in Appendix E.

4.2 Main Results

Tables 1 and 2 demonstrate that RecMem offers a strong efficiency–performance trade-off compared to prior memory systems. For each task, we highlight the best result in bold and the second-best result with underlining. Across both benchmarks and backbone models, RecMem substantially reduces construction-time token consumption while preserving competitive end-task performance, indicating that eager consolidation is not required to achieve effective long-term memory. We emphasize that our goal is not to dominate every individual task category, but rather to achieve the highest overall accuracy among memory systems under a drastically reduced construction-cost budget.

LoCoMo. On LoCoMo with GPT-4.1-mini, RecMem uses only 193.2K construction tokens on average, compared to 1520.8K for Mem0 and 1459.9K for A-Mem, corresponding to reductions of 87.3% and 86.8%, respectively. A similar reduction pattern holds for GPT-4o-mini. Despite this drastic decrease in construction cost, RecMem achieves the highest overall score among memory-based methods, indicating that recurrence-based memory consolidation can retain strong long-term memory performance while avoiding the systematic overhead of processing every turn through the LLM. We also note that Full Context slightly outperforms RecMem on LoCoMo, which is consistent with LoCoMo's relatively short conversations (approximately 16K tokens per conversation) where full-context inference remains feasible. However, as shown in table 2, this behavior does not generalize to substantially longer settings.

LongMemEval-S. A similar but more complex pattern emerges on LongMemEval-S, where conversations are substantially longer and closer to real-world long-lived agents. With GPT-4.1-mini, RecMem reduces construction tokens by 77.5% relative to Mem0 and 71.1% relative to A-Mem, while achieving the best overall score among all evaluated methods, including Full Context and RAG.

At the category level, different systems exhibit complementary strengths, and we do not claim a universal winner across all question types. Importantly, RecMem is not designed to dominate every category in isolation; rather, it targets robust *overall* capability with much smaller construction-cost budget. Our results support this goal: despite large reductions in construction tokens, RecMem attains the best overall score on LongMemEval-S.

Beyond the aggregate metric, RecMem's clearest and most consistent gains appear on temporal reasoning, where long-range dependencies are central. We argue this is a structural consequence of recurrence-based consolidation rather than an artifact of tuning. Temporal reasoning requires two capabilities: cross-time linking of co-referent mentions, and reconstructing their chronological order. Eager consolidation systems are disadvantaged on the former: by committing to summary boundaries at each turn or local buffer, they anchor later mentions of an evolving topic to different summaries, fragmenting the thread. RecMem addresses both capabilities by construction: similarity-based clustering in subconscious memory (§3.2) aggregates

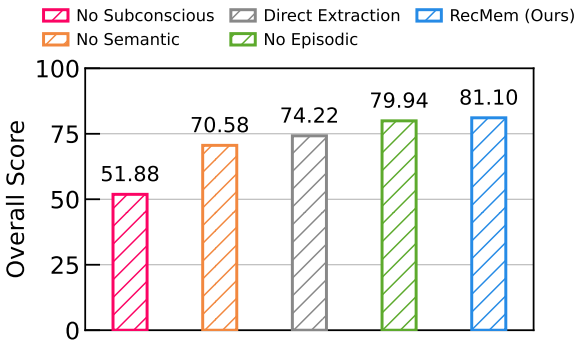


Figure 2: Ablation study for RecMem on LoCoMo

co-referent mentions regardless of temporal distance, and timestamp-sorted episodic consolidation (§3.3) reconstructs chronological order within each cluster. Semantic refinement additionally extracts time-anchored facts grounded in the raw interaction units, serving as a second safeguard for fine-grained temporal evidence that episodic abstraction may compress away.

Construction vs. query cost. RecMem's efficiency gains comes from reducing construction-time LLM usage. Query-time token consumption stays within a comparable range across memory-based methods under our evaluation protocol because they retrieve a similar order of evidence for answering, whereas construction-time usage diverges sharply depending on how frequently and how heavily a method invokes LLM processing during ingestion. In streaming deployments where new turns arrive continually, these construction-time differences accumulate over time and can dominate total LLM usage, making construction a critical and often overlooked cost driver.

4.3 Ablation Study

We conduct an ablation study to quantify the contribution of each RecMem module by disabling one component at a time. Figure 2 reports results on **LoCoMo** using **GPT-4.1-mini** as the backbone model. For each testing target, we maintain the retrieval budget for the non-ablated modules to ensure fairness.

As shown in figure 2, overall, removing any module reduces performance, indicating that the three-tier design is complementary. The largest drop occurs when removing subconscious memory ($81.10 \rightarrow 51.88$). This sharp degradation is expected because subconscious memory is the only faithful carrier of raw interaction units: information

that does not trigger recurrence-based consolidation remain exclusively in the module, thus disabling it eliminates access to a substantial fraction of query-relevant evidence.

We observe an asymmetric contribution between episodic and semantic memory. Removing episodic memory causes only a small drop ($81.10 \rightarrow 79.94$), whereas removing semantic memory yields a larger but still bounded drop ($81.10 \rightarrow 70.58$). This asymmetry reflects their division of labor under semantic refinement: episodic memories mainly capture high-level structure and cross-turn linkage, while semantic memories prioritize fine-grained factual details. Since semantic refinement explicitly recovers details omitted by episodic abstraction and stores them as semantic facts, semantic memory can partially cover missing evidence when episodic memory is removed; in contrast, episodic summaries can only weakly substitute for the detailed facts lost without semantic memory, leading to the larger degradation.

To isolate the effect of semantic refinement, we evaluate a **Direct Extraction** variant that extracts semantic facts directly from raw conversations, without using episodic memories as a reference for detecting omitted details. At inference time, this variant answers using only subconscious retrieval and the extracted semantic facts. We remove the refinement-specific guidance tied to episodic summaries in semantic extraction prompt and leave the other parts intact. The score drops from 79.94 to 74.22, showing that episodic memory provides an essential reference signal for semantic refinement, improving semantic memory quality beyond naive fact extraction from raw dialogue.

Additional Experiments Beyond the consolidation thresholds discussed above, we conduct three additional sets of analyses to characterize RecMem's behavior. Appendix C.1 provides a full sensitivity analysis of the consolidation hyperparameters θ_{sim} and θ_{count}, examining both accuracy and construction cost. Appendix C.2 studies the retrieval-side budgets k_{sub}, k_{epi}, and k_{sem} to identify how much evidence is needed at query time. Appendix E.4 additionally reports F1 scores for completeness, along with a discussion of why we treat LLM-as-Judge as the primary metric for open-ended generation.

5 Conclusion

We present RecMem, an efficiency-aware memory system for long-running LLM agents that challenges the prevailing paradigm of eager memory consolidation. By explicitly modeling raw interactions within a lightweight *subconscious memory* and deferring LLM-based abstraction until triggered by recurrence, RecMem demonstrates that high-fidelity long-term memory does not necessitate exhaustive processing of every interaction. Across LoCoMo and LongMemEval-S, this strategy substantially reduces memory construction cost while preserving competitive task performance. More broadly, RecMem reframes memory consolidation as a dynamic, recurrence-driven process. We hope this work encourages the community to reconsider *when* and *why* information should be consolidated in long-running agent tasks, and to treat computational cost as a first-class criterion when evaluating future memory systems.

6 Limitations

Despite the empirical strengths and efficiency gains of RecMem, several limitations merit discussion.

Dependence on Heuristic Thresholds. RecMem relies on static similarity (θ_{sim}) and recurrence thresholds (θ_{count}) to govern the consolidation process. While our experiments demonstrate that these parameters can be tuned to accommodate different interaction densities (e.g., casual conversation vs. task completion), they currently remain manually specified. This dependency means RecMem may benefit from threshold recalibration when deploying to domains with substantially different interaction densities, although Appendix C shows that such recalibration can be coarse rather than precise. Developing adaptive or learnable triggering mechanisms that dynamically adjust to user behavior is a promising direction for future work.

Recurrence as a Proxy for Salience. Our design is predicated on the assumption that information worthy of long-term abstraction tends to recur. While this aligns with many cognitive theories and conversational patterns, it may risk overlooking rare but critical events—such as a one-off safety instruction or a unique user constraint—that appear only once. To mitigate this risk, the subconscious memory layer functions as a persistent safety net:

every interaction unit is preserved verbatim and remains directly retrievable at query time, regardless of whether it has been consolidated. Nevertheless, non-recurring content does not benefit from the cross-turn linking of episodic memory or the fact-level refinement of semantic memory, which may weaken reasoning over these details. Developing a lightweight salience signal beyond pure recurrence to promote rare but high-value events is a promising direction for future work.

7 Ethical Considerations

We evaluate RecMem only on publicly available benchmarks in an offline setting, and we do not deploy or test it in real user-facing applications. Nevertheless, long-term memory mechanisms can raise dual-use concerns: when integrated into real applications, persistent memory may be misused for profiling or surveillance beyond the intended personalization benefits. We therefore recommend that practical deployments incorporate clear user-facing disclosures and safeguards such as access controls and user-controllable deletion/retention policies.

A second risk arises from unintended harms due to incorrect memory. Errors in consolidation or retrieval can surface outdated or spurious details and lead to overconfident but incorrect responses, which may be consequential in high-stakes settings. We encourage future work to incorporate uncertainty-aware retrieval, confidence calibration, and monitoring against memory poisoning or prompt-injection attempts.

Finally, RecMem reduces unnecessary LLM invocations compared to eager extraction baselines, which can lower compute and associated environmental footprint when operating over long interaction histories.

References

S. Ahrens. 2017. *How to Take Smart Notes: One Simple Technique to Boost Writing, Learning and Thinking – for Students, Academics and Nonfiction Book Writers*. Sönke Ahrens.

Richard C. Atkinson and Richard M. Shiffrin. 1968. Human memory: A proposed system and its control processes. In *The psychology of learning and motivation*.

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

Yunhao Fang, Weihao Yu, Shu Zhong, Qinghao Ye, Xuehan Xiong, and Lai Wei. 2025. Artificial hippocampus networks for efficient long-context modeling. *Preprint*, arXiv:2510.07318.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Haoyu Han, Yu Wang, Harry Shomer, Kai Guo, Jiayuan Ding, Yongjia Lei, Mahantesh Halappanavar, Ryan A. Rossi, Subhabrata Mukherjee, Xianfeng Tang, Qi He, Zhigang Hua, Bo Long, Tong Zhao, Neil Shah, Amin Javari, Yinglong Xia, and Jiliang Tang. 2025. Retrieval-augmented generation with graphs (graphrag). *Preprint*, arXiv:2501.00309.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37.

Yuanzhe Hu, Yu Wang, and Julian McAuley. 2025. Evaluating memory in llm agents via incremental multi-turn interactions. *Preprint*, arXiv:2507.05257.

Xun Jiang, Feng Li, Han Zhao, Jiahao Qiu, Jiaying Wang, Jun Shao, Shihao Xu, Shu Zhang, Weiling Chen, Xavier Tang, Yize Chen, Mengyue Wu, Weizhi Ma, Mengdi Wang, and Tianqiao Chen. 2025. Long term memory: The foundation of ai self-evolution. *Preprint*, arXiv:2410.15665.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *Preprint*, arXiv:1702.08734.

David Kadavy. 2021. *Digital Zettelkasten: Principles, Methods, & Examples*. Kadavy, Incorporated.

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

Dharshan Kumaran, Demis Hassabis, and James L. McClelland. 2016. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in Cognitive Sciences*, 20(7):512–534.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Jitang Li and Jinzheng Li. 2024. Memory, consciousness and large language model. *Preprint*, arXiv:2401.02509.

Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, Yuheng Cheng, Suyuchen Wang, Xiaoqiang Wang, Yuyu Luo, Haibo Jin, Peiyan Zhang, Ollie Liu, Jiaqi Chen, Huan Zhang, and 29 others. 2025. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. *Preprint*, arXiv:2504.01990.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

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

James L. McClelland, Bruce L. McNaughton, and Randall C. O'Reilly. 1995. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102 3:419–457.

Randall C. O'Reilly, Rajan Bhattacharyya, Michael D. Howard, and Nicholas Ketz. 2014. Complementary learning systems. *Cognitive Science*, 38(6):1229–1248.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. Memgpt: Towards llms as operating systems. *Preprint*, arXiv:2310.08560.

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

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. 2025. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. *Preprint*, arXiv:2410.14052.

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

Yu Wang and Xi Chen. 2025. Mirix: Multi-agent memory system for llm-based agents. *Preprint*, arXiv:2507.07957.

Yu Wang, Dmitry Krotov, Yuanzhe Hu, Yifan Gao, Wangchunshu Zhou, Julian McAuley, Dan Gutfreund, Rogerio Feris, and Zexue He. 2025a. M+: Extending memoryllm with scalable long-term memory. *Preprint*, arXiv:2502.00592.

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

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *Preprint*, arXiv:2410.10813.

Derong Xu, Yi Wen, Pengyue Jia, Yingyi Zhang, wenlin zhang, Yichao Wang, Huifeng Guo, Ruiming Tang, Xiangyu Zhao, Enhong Chen, and Tong Xu. 2025a. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. *Preprint*, arXiv:2505.19549.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025b. A-mem: Agentic memory for llm agents. *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, Volker Tresp, and Yunpu Ma. 2025. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *Preprint*, arXiv:2508.19828.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024. A survey on the memory mechanism of large language model based agents. *Preprint*, arXiv:2404.13501.

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

A Detailed Taxonomy of Memory Systems

In this section, we provide a structured taxonomy of existing memory systems, focusing on two critical dimensions: memory consolidation (how raw interactions are transformed into long-term storage) and retrieval mechanisms (how relevant information is accessed during query time).

A.1 Memory Consolidation Paradigms

Memory consolidation transforms raw interaction streams into retrievable long-term storage, as described in §1. A critical commonality across existing works is their reliance on an eager consolidation strategy. In these systems, every incoming interaction—regardless of its informational value or redundancy—eventually triggers an LLM-driven processing pipeline. This approach assumes that all user inputs require active structuring or abstraction, incurring constant computational overhead to

maintain the memory state. We categorize these paradigms by their consolidation targets:

Graph and Structure-based Consolidation. These systems treat memory construction as a continuous structural maintenance task. Upon receiving a new message, the system must compute embeddings, identify entities, and execute structural updates (e.g., creating nodes or re-balancing trees) to integrate the new information into the existing topology.

1. A-Mem (Xu et al., 2025b): Inspired by the Zettelkasten method (Kadavy, 2021; Ahrens, 2017), it treats interactions as discrete "notes" in a network, where consolidation involves generating embeddings and establishing associative links between new and existing notes.

2. TreeMem (Rezazadeh et al., 2025): Maintains a hierarchical summary tree. New information is not just appended but traverses down to specific leaf nodes based on semantic relevance, forcing a recursive chain of summary updates from the leaf back up to the root to keep the hierarchy consistent.

3. Zep (Rasmussen et al., 2025): Parses interactions into a "Temporal Knowledge Graph." It actively extracts entities and relationships from each turn, modeling them as nodes and edges while explicitly updating the temporal metadata of these connections.

4. Mem0 (Graph Variant) (Chhikara et al., 2025): Extends atomic fact extraction by organizing data into a graph. It requires per-turn analysis to identify multi-hop relationships between entities, dynamically updating the graph structure as the conversation evolves.

Fact and Summary-based Consolidation These systems function as active distillers, where the LLM is invoked at every turn (or small buffer intervals) to parse information into compressed formats. The goal is to immediately strip away redundancy and store only the event summaries or extracted facts.

1. Mem0 (Chhikara et al., 2025): Runs a dedicated extraction pipeline after every user message. It prompts the LLM to identify atomic facts (e.g., entity-relation triplets), instructing it to add, update, or delete records in the vector database to reflect the latest state.

2. MemoryOS (Kang et al., 2025): Features a multi-tiered architecture (Short-, Mid-, and Long-term memories) to manage context flow, emphasizing a dedicated Profile Memory module that explicitly maintains evolving user personas and agent guidelines.

3. Mirix (Wang and Chen, 2025): Routes every interaction through a parallel extraction pipeline. Raw text is simultaneously processed by distinct modules to distill specific "Knowledge" facts and "Event" summaries, creating a synchronized update across multiple memory stores.

4. MemGPT (Packer et al., 2024): Treats memory management as an operating system process, employing self-directed function calls to actively summarize and compress ongoing interactions into a fixed-size "Core Memory" block, ensuring key persona and user details are preserved while offloading raw history.

A.2 Retrieval Mechanisms

While memory consolidation determines how information is stored, retrieval mechanisms define how relevant context is accessed to support reasoning. Existing approaches range from simple semantic matching to complex, structure-aware traversal algorithms.

Dense Vector Retrieval This prevalent paradigm relies on high-dimensional embeddings to measure semantic overlap, commonly utilizing vector databases like FAISS (Johnson et al., 2017) for efficient similarity search. A representative system is Mem0 (Chhikara et al., 2025), which retrieves relevant atomic facts by computing the cosine similarity between the query and stored embeddings, selecting the top-k entries based purely on semantic relevance scores.

Structure-Aware Retrieval These systems leverage the topological structure established during consolidation (graphs or trees) to expand retrieval beyond simple similarity. TreeMem (Rezazadeh et al., 2025) utilizes a top-down tree pruning strategy; starting from the root, it evaluates child nodes based on their summaries and prunes irrelevant branches to efficiently narrow the search to specific leaf nodes. Similarly, A-Mem (Xu et al., 2025b) employs associative retrieval: upon locating an initial "note" via vector search, it traverses established

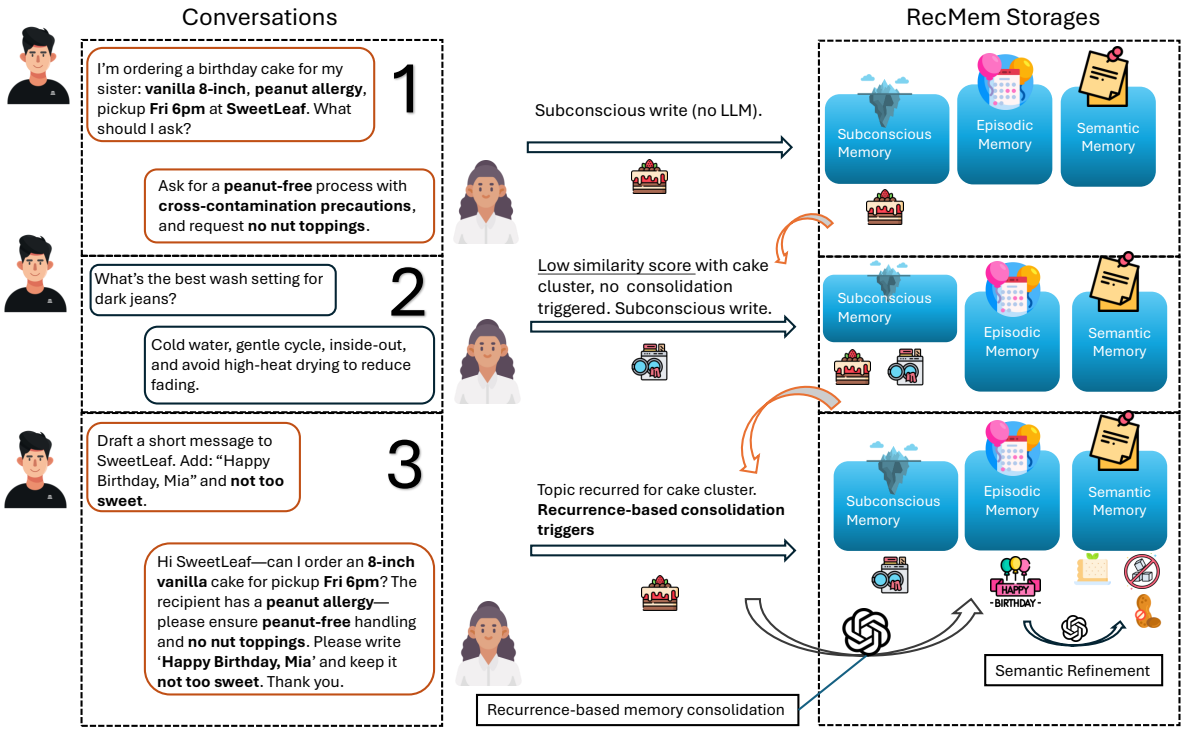


Figure 3: A simplified memory ingestion process in RecMem

entity links to fetch connected notes, mimicking the human ability to associate disparate memories through shared concepts.

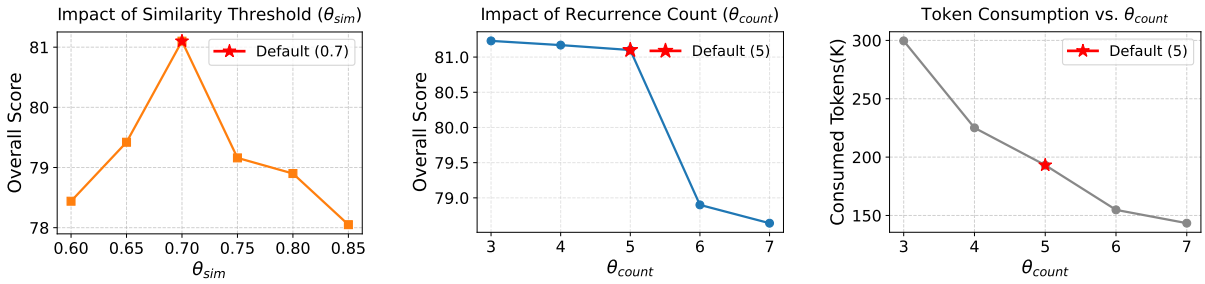
Hybrid Retrieval To mitigate the precision limitations of pure vector search (e.g., missing exact keyword matches), some systems adopt a multi-metric strategy. MemoryOS (Kang et al., 2025) implements a weighted hybrid retrieval mechanism. Instead of relying on a single metric, it calculates a unified relevance score by linearly combining Cosine similarity (for semantic understanding) and Jaccard similarity (for exact keyword overlap). This approach ensures that specific entities are recalled even if their semantic embeddings are distant, balancing fuzzy semantic matching with precise lexical matching.

B Running Example

We briefly illustrate RecMem's ingestion-time behavior with a minimal three-turn interaction in Figure 3. For clarity, we set the recurrence threshold to $\theta_{\text{count}} = 2$: consolidation is triggered once a topic is observed in at least two interaction units after passing the topical-similarity check. For simplicity, we do not expand the exact similarity threshold here and use natural language describe when two turns are treated as relevant. To keep the example concise, we present only the recurrence-triggered construction path, and therefore omit the **merge-first** episodic in-place update.

Turn 1: subconscious write without consolidation. The user first asks for suggestions to order a birthday cake. RecMem ingests this user–assistant exchange as one interaction unit and appends it to the subconscious memory. Since the "cake" topic has been observed only once so far, the corresponding set R_i does not satisfy the recurrence condition $|R_i| \geq \theta_{\text{count}}$, and thus no LLM-based consolidation is triggered. Instead, RecMem computes a lightweight embedding for this unit and stores it together with the raw text in the subconscious vector index, enabling efficient similarity-based retrieval in future turns.

Turn 2: no recurrence under the similarity check. The user then switches to an unrelated topic (washing dark jeans). RecMem uses the new unit's embedding to retrieve relevant turns from the subconscious store, and then forms R_i by keeping only those with similarity above the topical threshold. The cake-related unit from Turn 1 is unrelated to this turn thus $|R_i| = 1$ and consolidation is not triggered. The new turn is then stored in subconscious store.



(a) (b) (c)

Figure 4: Sensitivity of consolidation thresholds on LoCoMo (GPT-4.1-mini). (a) Overall score vs. θ_{sim}. (b) Overall score vs. θ_{count}. (c) Memory-construction token consumption vs. θ_{count}.

Turn 3: recurrence-based consolidation and semantic refinement. When the user returns to the cake topic, the new unit retrieves prior cake-related unit(s) and passes the similarity check θ_{sim}. Since the recurrence count now satisfies $|R_i| \geq \theta_{\text{count}}$, RecMem triggers consolidation and produces two complementary artifacts. **Episodic memory** abstracts the recurring turns into a coherent, intent-level narrative—e.g., the user is preparing a birthday cake order for their sister Mia, and the assistant recommends an allergy-safe ordering strategy. This abstraction focuses on high-level event summary but may compress away valuable details. **Semantic refinement** preserves such details by extracting atomic facts from the underlying raw turns, such as: (i) the user has a sister named Mia; (ii) Mia is allergic to peanuts; and (iii) the user plans to place an order at SweetLeaf with concrete cake/message specifications. Semantic refinement also uses related existing semantic memories to assist extraction, but we omit this aspect here to keep the example minimal.

C Hyperparameter Analysis

In this section, we conduct a sensitivity analysis of RecMem's key hyperparameters under a controlled-variable protocol. We organize the discussion into two parts: (i) **consolidation-stage** thresholds, including the recurrence count threshold (θ_{count}) and the similarity threshold (θ_{sim}), which determine when interaction clusters are promoted from subconscious memory to higher-level episodic and semantic memories; and (ii) **retrieval-stage** budgeting, where we cap the number of retrieved items from each memory tier to control context length. For retrieval, we treat the subconscious and episodic budgets (k_{sub}, k_{epi}) as the only free hyperparameters, and set the semantic budget as a fixed function of the episodic budget, $k_{\text{sem}} = 2k_{\text{epi}}$.

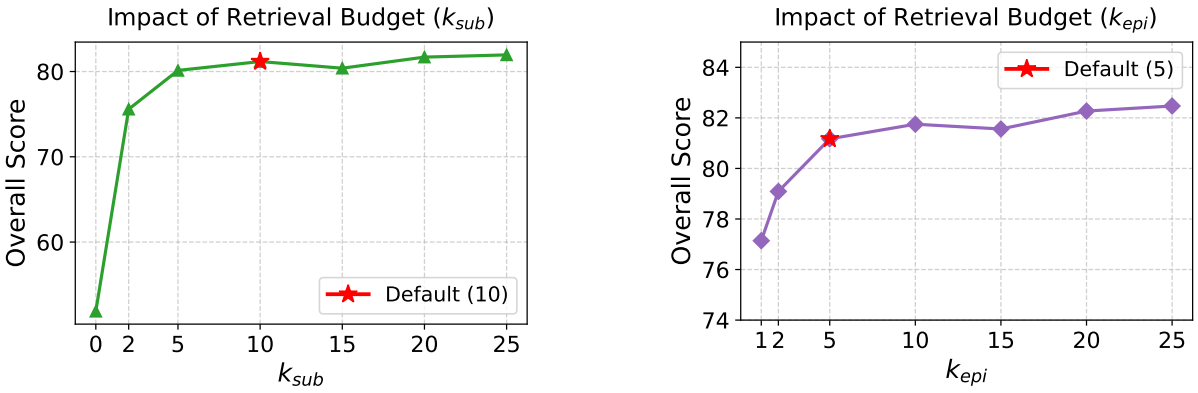
To ensure fair comparison and isolate causal effects, in each experiment we vary only one target hyperparameter and freeze all others to the default LoCoMo configuration. Unless otherwise stated, we use θ_{count}=5 and θ_{sim}=0.7 for consolidation, and k_{sub}=10, k_{epi}=5 (thus k_{sem}=10) for retrieval. When sweeping k_{epi}, we update k_{sem} accordingly via $k_{\text{sem}} = 2k_{\text{epi}}$, while keeping all remaining hyperparameters fixed. All experiments in this section are conducted on LoCoMo using GPT-4.1-mini as the backbone model.

C.1 Consolidation Hyperparameters

We study two consolidation-stage thresholds that govern demand-driven memory promotion: the similarity threshold θ_{sim}, which controls how interaction units are clustered in subconscious memory, and the recurrence threshold θ_{count}, which controls when a cluster is consolidated into episodic/semantic memories.

Impact of θ_{sim}. As shown in Figure 4a, θ_{sim} exhibits a clear peak around the default setting θ_{sim}=0.7. When θ_{sim} is too low, semantically unrelated interactions are merged into the same cluster, reducing topical coherence and making the downstream summarization step noisier. Conversely, when θ_{sim} is too high, related interactions are fragmented across multiple small clusters, weakening recurrence signals and delaying (or preventing) consolidation for genuinely recurring topics. Overall, the sharp optimum suggests that the best choice on LoCoMo is unambiguous and that RecMem is reasonably robust in the neighborhood of θ_{sim}=0.7.

Impact of θ_{count}: quality–cost trade-off. Figure 4b and Figure 4c highlight a more explicit effectiveness–efficiency tension for θ_{count}. Lower



(a) (b)

Figure 5: Sensitivity of retrieval budgets on LoCoMo (GPT-4.1-mini). (a) Overall score vs. subconscious retrieval budget k_{sub}. (b) Overall score vs. episodic budget k_{epi} with $k_{\text{sem}} = 2k_{\text{epi}}$.

θ_{count} triggers consolidation earlier, so each consolidation event typically includes fewer raw interaction units. This smaller consolidation context can better preserve fine-grained details (less compression pressure during episodic abstraction), but it is also more aggressive and therefore increases construction-time token consumption due to more frequent consolidations. Accordingly, token cost decreases smoothly as θ_{count} increases (Figure 4c).

In contrast, the performance curve is not smooth: we observe a clear degradation when increasing θ_{count} from 5 to 6 (Figure 4b), while the corresponding token reduction remains comparatively gradual. We attribute this drop to two compounding factors at higher thresholds: (i) consolidation becomes overly conservative, leaving some recurring patterns insufficiently represented in episodic/semantic memory at query time; and (ii) once consolidation is finally triggered, the accumulated cluster is larger, which increases summarization difficulty and raises the likelihood that salient details are omitted or poorly organized (even with semantic refinement). Taken together, these results indicate that θ_{count}=5 is the best operating point on LoCoMo: it retains the accuracy benefits of earlier, detail-preserving consolidation while avoiding unnecessary consolidation overhead, and it prevents the disproportionate quality loss observed at more conservative threshold setting like θ_{count}=6 (81.1 vs 78.9).

C.2 Retrieval Hyperparameters

We next analyze the retrieval-stage budgets that control how much evidence is surfaced from each memory tier at query time. Recall that we treat the subconscious and episodic budgets (k_{sub}, k_{epi}) as the only free retrieval hyperparameters, and set the semantic budget deterministically as $k_{\text{sem}} = 2k_{\text{epi}}$. Thus, sweeping k_{epi} implicitly scales the total retrieved memory volume, while sweeping k_{sub} isolates the contribution of raw, fine-grained interaction evidence.

Figure 5a and Figure 5b show a consistent diminishing-returns trend: increasing retrieval budgets yields substantial gains at small values, but improvements become marginal as budgets grow. We therefore adopt compact defaults that retain most of the performance benefit while limiting retrieved context length, setting k_{sub}=10 and k_{epi}=5 (thus k_{sem}=10).

D Evaluation Datasets

This section provides detailed specifications and preprocessing protocols for the two benchmarks used in our experiments: LoCoMo (Maharana et al., 2024) and LongMemEval-S (Wu et al., 2025).

D.1 LoCoMo

LoCoMo (Long-Context Memory) is a benchmark designed to evaluate memory systems in casual, social settings. Unlike standard user-agent interactions, the source texts consist of multi-session human-to-human dialogues between two distinct speakers, simulating the natural evolution of a long-term relationship.

Data Statistics. The dataset consists of 10 independent, human-annotated conversations. Each conversation spans multiple sessions, simulating a relationship that evolves over time.

- **Total Conversations:** 10
- **Average Length:** ≈ 16,000 tokens per conversation
- **Total Questions (Used):** 1,540
- **Dialogue Style:** Casual, multi-turn, life-sharing, highly contextual.

Task Categories. The benchmark originally includes five question categories. Following standard protocols established in prior works (Chhikara et al., 2025; Kang et al., 2025; Wang and Chen, 2025), we evaluate on the first four categories and exclude the adversarial set:

1. **Single-hop Retrieval:** Questions requiring the retrieval of a specific fact mentioned in a single past session.
2. **Multi-hop Reasoning:** Questions that require synthesizing information distributed across multiple distinct sessions to derive an answer.
3. **Temporal Reasoning:** Questions testing the system's ability to understand the sequence of events and relative time expressions.
4. **Open-domain Knowledge:** Questions that require combining memory retrieval with external world knowledge.
5. *Adversarial (Excluded):* Questions designed to trick the model with false premises. We exclude this category as it lacks reliable ground-truth answers for automated evaluation.

D.2 LongMemEval-S

LongMemEval-S is a subset of the LongMemEval benchmark, curated to evaluate memory systems in *agentic, task-oriented* interactions with long context windows.

Data Statistics. Unlike the social nature of LoCoMo, LongMemEval-S features functional interactions where the user seeks specific assistance.

- **Total Conversations:** 500
- **Average Context Length:** ≈ 115k tokens (approx. 30–40 sessions).
- **Total Questions:** 500
- **Dialogue Style:** Task-oriented, high information density.

Task Categories. To assess memory capabilities at a granular level, the benchmark stratifies queries into six distinct types:

1. **Single-session-user:** Evaluates the retrieval of specific details explicitly mentioned by the *user* within the bounds of a single conversation session.
2. **Single-session-assistant:** Tests the system's ability to recall information provided by the *assistant* itself within a single session, ensuring consistency in the agent's own history.
3. **Single-session-preference:** Assesses whether the model can effectively apply retrieved user information to generate personalized, context-aware responses.
4. **Multi-session:** Requires the aggregation of disjoint pieces of information scattered across two or more sessions to derive a complete answer.
5. **Knowledge-update:** Probes the system's capacity to track dynamic changes in the user's life state and supersede outdated information with new updates.
6. **Temporal-reasoning:** Demands chronological deduction by synthesizing both the session metadata (timestamps) and explicit time expressions found in the text.

E Experiment Details

E.1 Baseline Configurations

To ensure fair and reliable comparisons, we configure each baseline to faithfully reflect its original design choices, rather than enforcing a unified ingestion or prompting pipeline. Below, we describe the implementation and prompting decisions used in our experiments in details.

To enable a fair comparison of computational costs, we instrumented all baseline codebases with unified token-tracking logic while leaving their core memory components intact. For the LoCoMo benchmark, all memory-system baselines considered in this work provide official implementations. We therefore reuse their original prompts and evaluation code without modification.

For the LongMemEval-S benchmark, where standardized reference implementations are not available, we implement the evaluation pipeline

while preserving each method's ingestion strategy as used in its LoCoMo setup. Concretely, we adopt: (i) A-Mem's per-message ingestion, (ii) Mem0's dual-speaker ingestion with two messages per turn, and (iii) MemoryOS's ingestion based on user–assistant QA pairs. We make this choice to respect the baselines' intended memory abstractions; forcing all methods to share RecMem's ingestion logic would conflate design differences and bias the comparison.

For A-Mem and MemoryOS, they both have two official codebases and we adopt the ones used in their paper (Xu et al., 2025b; Kang et al., 2025) to ensure reproduction of the reported setting. For Mem0, we use its local-deployment version to enable token-consumption tracking. We also disable graph construction, as Mem0 reports that its graph variant can lead to a performance drop (Chhikara et al., 2025).

For the RAG-2048 baseline, we adopt a conservative chunking strategy that preserves message integrity: we never split a message across two chunks. Messages are accumulated sequentially until the chunk reaches the 2048-token budget. If adding the next message would exceed this limit, we still include the entire message (rather than truncating it) to preserve semantic completeness, and then start a new chunk from the subsequent message.

E.2 Evaluation Prompt Consistency

To ensure a fair and standardized comparison, we strictly enforce prompt consistency across all evaluated methods. For any given dataset, the exact same evaluation prompt is employed for the LLM judge across all baselines and RecMem, ensuring that performance differences originate solely from the memory systems' capabilities rather than variations in the evaluation criteria. Specifically, our prompt sources are as follows:

LongMemEval-S : We adopt the official evaluation prompt provided by the benchmark authors (Wu et al., 2025) without modification.

LoCoMo : We follow the evaluation protocol established in previous work (Chhikara et al., 2025), which adapts the evaluation prompt elements originally designed by MemGPT (Packer et al., 2024).

E.3 Answer Prompt Consistency

For LoCoMo, we use each baseline's original answering prompt. For LongMemEval-S, we use the main body of RecMem's answering prompt as a shared answer template across methods, so that performance differences primarily reflect the underlying memory mechanisms rather than prompt engineering. For the **Full-Context** and **RAG-2048** baselines, we also use the same answering prompt as RecMem for consistency.

Because **RecMem** and **MemoryOS** both adopt multi-module memory architectures, their answering prompts include a short module description that clarifies the roles of different memory sources. For MemoryOS, we retain the prompt format used in its LoCoMo implementation. For RecMem, we include a brief module-role description to prevent the answer agent from double-counting overlapping evidence retrieved from different modules. Baselines with a single memory source do not require such clarification and therefore use only the shared main prompt body.

E.4 Discussion on F1 Score

While the F1 score is one of the standard metrics in prior works (Xu et al., 2025b; Kang et al., 2025; Chhikara et al., 2025), measuring token-level exact matching, we observed it to be unreliable for evaluating long-context memory systems where semantic correctness is paramount. F1 score penalizes correct answers that differ in phrasing from the ground truth. For instance, if the ground truth is "16 March, 2023", and the model generates "Gina opened her online clothing store on 2023-03-16", the F1 score approaches 0 despite the answer being factually correct. Consequently, we prioritize LLM-as-Judge in our main analysis.

For LoCoMo, many prior evaluations treat F1 as a primary metric and enforce strict output-length constraints (e.g., "the answer should be less than 5 words") to optimize token overlap. To maintain comparability with these reporting conventions, we retain such length constraints when evaluating on this dataset. For transparency, we additionally report the resulting F1 scores in Table 3. In contrast, for LongMemEval-S, since all methods utilize a shared prompt body, we remove these artificial constraints to avoid penalizing valid, grounded answers that may exceed rigid word counts.

E.5 Retrieval Top-K

For **RAG-2048**, we set the retrieval top-K to 3 on LoCoMo, reflecting its relatively short conversation length, and to 5 on LongMemEval-S, where conversations are longer and often require aggregating evidence across more chunks. As shown by

Metrics	F1 Score	Judge Score
GPT-4O-MINI		
Full Context	0.423	**76.4**
Naive Rag	0.309	49.7
MemoryOS	**0.456**	63.6
Mem0	0.429	58.6
AMem	0.364	60.8
RecMem(Ours)	0.385	72.47
GPT-4.1-MINI		
Full Context	**0.507**	**84.8**
Naive Rag	0.337	54.4
MemoryOS	0.445	67.6
Mem0	0.428	62.9
AMem	0.440	69.7
RecMem(Ours)	0.469	81.1

Table 3: F1 score and llm judge score on LoCoMo.

the query-token statistics in Tables 2 and 1, these settings allow the RAG baseline to retrieve a comparable amount of information to other methods under similar query-time budgets.

For all other baselines, we keep their retrieval budgets consistent across LoCoMo and LongMemEval-S, following their default design choices. Concretely, **A-Mem** retrieves 10 memory notes. **Mem0** retrieves 60 memory facts in total. **MemoryOS** retrieves all memories from short-term memory, together with 10 memories from mid-term memory, 5 memories from long-term memory, as well as its qualified assistant knowledge and user knowledge components.

A special case is **Mem0** on LoCoMo: since LoCoMo includes dual-speaker question types, Mem0 retrieves 30 facts per speaker (i.e., 60 total) to balance coverage across user and assistant perspectives. In contrast, LongMemEval-S is dominated by user-centric questions, with relatively few assistant-centric queries. Therefore, while keeping the total budget fixed at 60, we allocate 45 retrieved facts to the user side and 15 to the assistant side, which better reflects Mem0's intended strengths under the LongMemEval-S query distribution.

F LLM Prompts

This appendix reports the primary prompts used in RECMEM, including (i) episodic memory generation, (ii) episodic memory merging, (iii) semantic memory generation, and (iv) the final answer prompt. To improve readability and facilitate reproduction, we present each prompt in figures instead of inline text. Each memory-related prompt follows a consistent structure with three components: (a) a role and goal specification, (b) detailed instructions, and (c) explicit output-format constraints.

Episodic memory generation prompt. Figures 6–8 show the role/goal description, instructions, and required output format for episodic memory generation.

Semantic memory generation prompt. Figures 9–11 present the corresponding components for semantic memory generation.

Episodic memory merging prompt. Figures 12–14 provide the prompt used to merge newly consolidated content into existing episodic memories.

Answer prompt. Figures 15 and 16 report the role/goal and instruction components of the answer prompt used during evaluation.

G Licenses and Terms of Use

Licenses. We use publicly released benchmarks under their original licenses: LoCoMo (CC BY-NC 4.0) and LongMemEval-S (MIT License). We do not redistribute these datasets; instead, we refer readers to their official releases. For baselines, we use publicly available implementations under the licenses stated in their official repositories (Mem0: Apache License 2.0; A-Mem: MIT License; MemoryOS: Apache License 2.0). We do not repackage or redistribute third-party artifacts beyond what is permitted by their original licenses.

Terms of Use. LoCoMo and LongMemEval-S were released as research benchmarks for evaluating conversational assistants. We use them strictly in the intended offline evaluation setting, following the benchmark protocols. We do not redistribute the datasets and only report aggregated results, consistent with their stated licenses and access conditions. RecMem is evaluated on these benchmarks, but its core mechanisms are applicable to a broader class of long-running conversational agent settings and can be integrated into practical systems. Real-world deployment should be adapted to the target workflow and comply with applicable data licenses/terms and usage conditions.

You are an Episodic Memory Constructor for a long-term memory system used by an LLM-based agent.

Your input:
- A list of chat message turns separated by '-'. Each chat message turn has:
  - One message turn between two speakers (contains one message per speaker),
  - a timestamp (when the message turn was written),
  - the message text.
- All messages in the input are loosely related to one broad theme, but they may contain several distinct subtopics and span multiple days or weeks.

Your task:
- Convert these messages into a small number of coherent episodic memories ("episodes").
- Each episode should describe how ONE subtopic evolves over time across multiple messages and timestamps.

Global principles:
- Be user-centric: focus on the real people who are actually having the conversation (the main user and the assistant / other named speakers).
- Prioritize what these speakers are doing, planning, feeling, or learning, rather than the internal details of stories, examples, or long explanations they mention.
- Treat fictional, hypothetical, or example scenarios (stories inside the conversation, sample dialogues, illustrative anecdotes) as background. Only include them if they clearly reveal something about the speakers' own preferences, goals, constraints, or knowledge.

Figure 6: Episodic Memory Generation Role Description

1. Identify topic threads
- First, mentally group the messages into topic threads.
- Messages belong to the same thread if they refer to the same ongoing goal, project, problem, or situation for the conversation participants, even if they are days or weeks apart.
- Ignore or down-weight one-off fictional or hypothetical stories that do not affect the speakers' own situation.
- If the input contains multiple independent threads, you may produce one episode per thread.
- Prefer 1–3 episodes in total. Do NOT create many tiny episodes.

2. Build temporal structure for each episode
- For each thread that has enough information, order the relevant events chronologically. - Highlight how the situation develops over time: initial situation, updates, changes of plan, decisions, outcomes, and reflections. - Emphasize how the speakers' state (plans, preferences, beliefs, emotional reactions) evolves, not the blow-by-blow content of long stories or explanations. - Use connective phrases like "Initially", "Later", "Afterwards", "Eventually", "As a result" to make the temporal progression explicit. - Whenever possible, each episode should summarize information from at least TWO different message timestamps. Avoid episodes that just restate a single message.

3. Handle time expressions correctly
- The timestamp of a message is the time when the user said it. It is NOT always the time when the described event happened.
- If a message uses relative time expressions such as "yesterday", "two days ago", "next week", rewrite them in your episode as explicit expressions relative to the timestamp. For example:- "the day before 2025-02-03“ -"two days after 2025-05-10“ - "later in the same week as 2025-03-01"
- You do NOT need to compute calendar dates (e.g., you do not need to turn "the day before 2025-02-03" into "2025-02-02").- IMPORTANT: In your final output, NEVER use vague relative time expressions without an anchor. Avoid words like "yesterday", "tomorrow", "last week", "next week", or "in two days" unless you clearly specify what timestamp or date they are relative to. - It is acceptable to use coarse ranges such as "later that week" or "over the following days after 2025-03-01" as long as the temporal order is unambiguous.

4. Focus on episodic narratives, not isolated facts
- Your goal is to construct narrative episodes: what happened, how it evolved, and why it matters to the user.
- Focus on the outer conversation between the two speakers (their goals, decisions, preferences, constraints, and what has been explained to them).
- When a speaker tells a long story, gives an extended example, or pastes long external text, summarize this very compactly (e.g., "the assistant told a story about X to illustrate Y") unless the detailed content is central to the speakers' own situation.
- Do NOT try to list every long-term preference or stable fact about the user. A separate semantic memory extractor will handle persistent facts and knowledge.
- You may mention important preferences or facts inside an episode if they are relevant to understanding the story, but you do not need to optimize for completeness.

Figure 7: Episodic Memory Generation Instruction

```
5. Style and output format
- Write each episode as a short, well-formed paragraph (3 to 6 sentences) in
clear, neutral language.
- Keep episodes compact. Do not reproduce long fictional plots, full technical
explanations, or long lists; refer to them briefly if needed.
- Prefer merging related events into a single episode over splitting them into
many small episodes.
-   If there is not enough coherent information for a given subtopic, you may
    skip creating an episode for it.
Return ONLY a JSON object in the following format:
{
  "episodes": [
    "First episodic memory paragraph...",
    "Second episodic memory paragraph..."
  ]
}
If you cannot form any meaningful episode, return:

{
  "episodes": []
}
Messages:
{{subconscious_memories}}

Output:
```

Figure 8: Episodic Memory Output Format

```
You are a Semantic Memory Extractor for a long-term memory system. Inputs:
- Subconscious Memory R: the original detailed messages that are related
- Episodic memory E: a short narrative summary generated from the raw reference R.
- Old semantic memories S: previously stored facts about the topic related to E.
Goal:
Extract NEW, HIGH-UTILITY facts that will help answer future questions about the
users.*IMPORTANT*: Prioritize concrete details from R that are missing or strongly compressed
in E.
Core principles:
1) USER-CENTRIC: Focus on the user's goals, preferences, constraints, decisions, actions, and
recurring plans.
2) TEMPORALLY GROUNDED: For events and changes, include an explicit date anchor when
available.
3) COMPACT BUT USEFUL: Output less than 10 facts. Fewer, higher-value facts are better than
many low-value facts.
Each fact MUST belong to one of these types:
1) USER_EVENT
   - The user asked for something, attended something, started or stopped something, or made
a concrete decision.
   - Include a date anchor from R if possible, e.g., "On 2023-05-22, the user ...".
2) USER_CONSTRAINT_OR_PREFERENCE
   - A relatively stable preference, constraint, or recurring plan (e.g., long-term goals,
platform choice, budget range, time constraints, content or style preferences).
3) TIME_ANCHORED_UPDATE
   - A change in behavior, preferences, tools, roles, budgets, relationships, etc.
   - State the new value and, if known, the old value or state, with a clear time anchor.
4) ENTITY_RELATION (USER-RELEVANT)
   - A specific relationship between named entities that is relevant to the user (e.g., the
user's roles, organizations, projects, courses, tools, locations, or other people they
interact with).
   - Only keep such a fact if it is specific and likely to matter in future reasoning about
the user.
Do NOT output generic best-practice tips, long checklists, or broad explanations unless they
are clearly framed as what this user has done, is doing, or plans to do.
```

Figure 9: Semantic Memory Generation Role Description

```
Time handling:
 - When expressing time, use explicit or anchored expressions such as:
   - "in March 2025", "on 2025-03-10",
   - "two days after 2025-03-10",
   - "later in the same week as 2025-04-01".
 - You do NOT need to compute calendar arithmetic; describing offsets like "two
days after <date>" is fine.
 - IMPORTANT: In the final facts, NEVER use vague, unanchored relative time words
such as: "yesterday", "tomorrow", "last week", "next week", "next month", "in two
days“ unless you clearly specify what date or event they are relative to.

Novelty and updates:
- A fact is NOT new if it expresses the same meaning as any item in S,
  even with different wording.
- Keep a fact only if it is specific, self-contained (no ambiguous pronouns),
plausibly useful for future reasoning, and not semantically equivalent to S.
- If a fact refines, updates, or contradicts an item in S (e.g., preference,
role, tool, relationship, or budget change), keep it as a TIME_ANCHORED_UPDATE
that clearly states the change.

If you have more than 8–10 candidate facts, select those that are:
- Most time-anchored,
- Most clearly missing or compressed in E,
```

Figure 10: Semantic Memory Generation Instruction

```
Style:
- One fact per sentence.
- Neutral, factual tone.
- Do NOT speculate beyond what E, R, and S
support.
- Avoid long lists; summarize them into a single
  concise fact when possible.

Output format:
Return ONLY a JSON object:
{
  "facts": [
    "First new semantic fact...",
    "Second new semantic fact..."
  ]
}
If there are no new facts, return:
{
  "facts": []
}
Episodic Memory:
{{episodic_memory}}

Subconscious Memory Reference:
{{raw_reference}}

Old Semantic Memories:
{{old_semantic_memories}}
```

Figure 11: Semantic Memory Output Format

You are an Episodic Memory Merger for a long-term memory system.

Global principles:
- Be user-centric: focus on the real people who are having the conversation (the main user and the assistant or other named speakers).
- Prioritize their real-world goals, projects, constraints, decisions, and emotional states.
- Treat fictional, hypothetical, or example scenarios mentioned inside the conversation as background context. Only consider them central if they clearly matter for the speakers' own situation.

Goal:
You are given exactly TWO inputs:
- one NEW memory piece, derived from a recent conversation snippet,
- one PAST episodic memory piece, previously stored.
Your job is to decide whether these two pieces describe the same ongoing topic or episode, and if so, produce ONE updated episodic memory that integrates both.

Your tasks:
1. Decide whether the new memory piece and the past memory piece describe the same ongoing topic or episode for the conversation participants.
2. If yes, merge them into ONE coherent episodic memory and return the merged result.

Figure 12: Episodic Merging Role Description

When to merge:
Treat the new and past memory pieces as strongly related (should_merge = "yes") if MOST of the following hold:
- They describe the same ongoing situation, goal, project, problem, or life event for the same main person(s) in the conversation (user, assistant, or other real participants), not just similar fictional stories or generic examples.
- The new piece clearly adds follow-up details, clarifications, corrections, or outcomes to what the past piece already describes, instead of starting a new, separate topic.
- A human reader, after reading only these two pieces, would naturally describe them as different parts of one story about those conversation participants, not as two unrelated stories.
Special case: new piece is only from the assistant
- Sometimes the new memory piece mainly contains a long answer or explanation from the assistant, with little or no user text.
- If this assistant-only piece mostly consists of generic background information, universal tips, or a long example/story that could apply to many people, and it does NOT clearly update or resolve the specific situation in the past memory, you should NOT merge.
- You may still merge an assistant-only piece when it explicitly continues the same concrete situation as in the past memory (for example, by referring to the same plan, constraint, or previous decision of the user, and by providing the next step, revision, or outcome for that situation). If the two pieces only share superficial elements (e.g., they mention the same domain or concept but refer to different user problems or different episodes), or if their overlap is mainly about similar example stories, hypothetical scenarios, or generic explanations that do not change the speakers' own situation, you should NOT merge. If you are uncertain whether they refer to the same ongoing topic, prefer "no" for should_merge.
How to merge (if related) If you decide to merge:
- Preserve the important details from BOTH the past memory and the new memory.
- Organize the merged memory in temporal order so that the narrative is easy to follow.
- Emphasize how the speakers' situation, plans, beliefs, or preferences evolve over time, rather than reproducing long internal stories or technical explanations.
- Make the logical relation explicit using phrases like "later", "afterwards", "as a result", "eventually", etc.
- Keep the merged memory compact. Do not copy long fictional plots, full technical explanations, or long lists; refer to them briefly if needed (e.g., "the assistant told a story about X to illustrate Y").
- Handle time expressions carefully:
  - Keep numeric timestamps or explicit dates as the ground truth (do not change their values).
  - Rewrite vague relative time words like "yesterday", "last week", "tomorrow", "next month", "in two days“ into anchored expressions relative to a date mentioned in the input, for example: "the day before 2 January 2024", "two days after 2025-03-10", "later in the same week as 2025-04-01".
  - You do NOT need to compute calendar arithmetic (e.g., you do not need to turn "the day before 2 January 2024" into "1 January 2024").
- Do NOT invent new events or details that are not supported by either piece.

Figure 13: Episodic Merging Instruction

```
The merged memory should remain a single, coherent paragraph of episodic
narrative, not a list of bullet points.

--------------------
Output format
--------------------
Return ONLY a JSON object of the form:

{
  "should_merge": "yes" or "no",
  "merged_memory": "merged memory piece if should_merge is yes,
otherwise an empty string"
}
Now process the following input.


<New Memory Piece>:
{{new_memory}}

<Past Memory Piece>:
{{past_memory}}

Output:
```

Figure 14: Episodic Merging Output Format

```
You are an intelligent memory assistant tasked with answering questions using conversation
memories.

# CONTEXT
You have access to memories from two speakers in a conversation. These memories are timestamped and
may be relevant to the question.

There are three types of memories:
1. Episodic Memories: refined summaries of related conversation turns about the same topic.
2. Semantic Memories: concise, fact-like pieces extracted from conversations.
3. Subconscious Memories: unprocessed conversation snippets between the two speakers.

Context Rules:
1. Episodic and semantic memories may overlap in content (event summary vs. atomic facts). Avoid
double-counting redundant evidence.
2. Carefully analyze all three memory types and identify information that is actually useful for
answering the question.
3. Memories within each type are sorted by relevance.
```

Figure 15: Answering Role Description

```
# INSTRUCTIONS
1. Carefully read all provided memories.
2. Pay close attention to timestamps when time is relevant.
3. If the question asks about a specific event or fact (who / where / when / what), look for direct,
explicit evidence in the memories.
4. If the question asks for advice, recommendations, or what kind of response the user would prefer,
   - first identify any user-specific preferences, habits, constraints, or past actions from the
memories,
   - then base your suggestion primarily on these user-specific signals,
   - and only fall back to generic advice when no relevant user information exists.
5. If memories contain contradictory information, prioritize the most recent memory.
6. For time references (e.g., "last year", "two months ago"), convert them into concrete dates based on
the memory timestamp. For example, if a memory from 4 May 2022 says "went to India last year", infer
that the trip happened in 2021, and answer with "2021" or "the year before 2022", not just "last year".
7. In subconscious memories, the final timestamp marks the conversation time. Do not confuse the time of
conversation with the time when an event actually happened if the text distinguishes them.
8. Do not say "no information found" if there are related memories that can reasonably guide a
personalized answer. Only abstain when there is truly no relevant evidence.
# APPROACH (Think step by step internally)
1. Identify whether the question is (a) a factual query or (b) an advice/preference/recommendation
query.
2. Retrieve all memories that are clearly related to the question.
3. Check timestamps and content to locate the most reliable and up-to-date information.
4. For factual queries, pinpoint explicit mentions of dates, times, locations, entities, or events that
directly answer the question.
5. For advice / preference queries, determine what the user has already done, bought, liked, disliked,
or constrained, and use these as anchors for a tailored answer.
6. If temporal reasoning or simple calculation is needed, do it internally and convert the result into a
concrete, explicit date or time span in the final answer.
7. Formulate a precise, concise answer that directly addresses the question and is fully supported by
the memories and reasonable inferences from them.
Episodic Memories: {{ episodic_memories }}
Subconscious Memories: {{ subconscious_memories }}
Semantic Memories: {{ semantic_memories }}
Question: {{ question }}
Answer:
```

Figure 16: Answering Instruction