# H-MEM: A NOVEL MEMORY MECHANISM FOR EVOLVING AND RETRIEVING AGENT MEMORY VIA A HYBRID STRUCTURE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Memory data are ubiquitous in Large Language Model (LLM)-based agents (e.g., OpenClaw and Manus). A few recent works have attempted to exploit agents' memory for improving their performance on the question-answering (QA) task, but they lack a principled mechanism for effectively modeling how memory data evolves over time and retrieving memory data effectively, leading to poor performance in memory utilization. To fill this gap, we present H-Mem, a novel memory mechanism via a hybrid structure that can not only effectively model the evolution of agent memory over a long period of time, but also provide an efficient memory retrieval approach. Particularly, H-Mem builds a temporal and semantic tree structure that allows the short-term memory data to evolve progressively into long-term memory data, where the latter provides summarized information about the former, while simultaneously constructing a knowledge graph to capture the relationships between entities in memory. Moreover, it offers an effective memory retrieval approach by exploiting the hybrid structure of the tree and graph structures. Extensive experiments on three agent memory benchmarks show that H-Mem achieves state-of-the-art performance on the QA task.

## 1 INTRODUCTION

LLM-based agents such as OpenClaw (OpenClaw Contributors, 2026) and Manus (Manus AI, 2025) have received tremendous attention owing to their powerful abilities in solving complex real-world tasks such as QA. During the interactions between users and agents, a large amount of memory data has been generated and accumulated. As a result, a crucial component of modern agents is the memory mechanism, which stores and manipulates the agent memory data.

To enable an LLM-based agent to exploit memory data, a naive memory mechanism is to store the memory as plain text, retrieve all memory data associated with a specific user, and then use the retrieved data to accomplish a task for that user. However, due to the finite context window of LLMs, this mechanism cannot effectively or efficiently process large amounts of memory data, especially when the agent has interacted with users over a long period of time. To alleviate this bottleneck, existing systems often apply Retrieval-Augmented Generation (RAG) techniques to memory data (Lewis et al., 2020; Wu et al., 2026); that is, the agent only retrieves relevant memory information from an external memory database when solving a task.

Under this memory-based RAG paradigm, existing methods differ not only in how memory data are indexed before retrieval, but also in their retrieval mechanisms which determine how relevant evidence is accessed from the index. Table 1 summarizes these index structures and retrieval mechanisms.

Despite this progress, existing memory mechanisms still face challenges in two major aspects: First, their index designs are still limited in modeling memory evolution, where short-term memory can be progressively consolidated into long-term memory, as suggested by studies of human memory consolidation (McGaugh, 2000; Squire & Alvarez, 1995). This motivates explicitly taking the temporal dimension into account when differentiating between short- and long-term semantic topics within the memory data. Second, accurately retrieving the relevant evidence from the memory index re-



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Taxonomy of representative agent memory methods.

| Method | Main Index Structures | Memory Retrieval Method |
|---|---|---|
| Mem0 (Chhikara et al., 2025) | Vector | Vector-based |
| MemOS (Li et al., 2025) | Tree | Lexical-based and vector-based |
| MemTree (Rezazadeh et al., 2025) | Tree | Vector-based and structure-based |
| MemoryOS (Kang et al., 2025) | Hierarchical | Lexical-based and vector-based |
| EverMemOS (Hu et al., 2026) | Hierarchical | Lexical-based, vector-based, and agentic |
| Zep (Rasmussen et al., 2025) | Graph | Lexical-based, vector-based, and structure-based |
| H-MEM | Hybrid of tree and graph | Vector-based, structure-based, and agentic |

mains challenging when performing QA tasks. Thus, jointly modeling long-term memory evolution and supporting accurate retrieval remains challenging.

To address the aforementioned challenges, we propose H-MEM, a novel memory mechanism via a hybrid structure of tree and graph. The key distinction of H-MEM is not merely using a tree index together with a graph index, but coupling temporal-semantic memory evolution with entity-centered multi-hop reasoning. The tree structure of H-MEM organizes memory data both temporally and semantically, where each tree node retains memory information regarding a specific semantic topic within a pre-defined time window. Furthermore, the graph structure of H-MEM maintains a knowledge graph of entities and their relationships extracted from the memory data, effectively recording the entity-centered information beyond temporal order and capturing multi-hop relationships between entities across different memory fragments. Overall, the tree and graph structures complement each other, and this hybrid structure mitigates the limitations of relying on a single index.

We have evaluated H-MEM against representative SOTA baselines on three public long-term memory benchmarks covering diverse QA scenarios. The results show that H-MEM achieves superior F1 scores and accuracy with a quality–cost trade-off. Further analyses validate the contribution of the temporal tree, the knowledge graph, and the agent-assisted retrieval strategy. Our principal contributions are summarized as follows:

- We propose H-MEM, a novel memory mechanism that can effectively model the evolution of agent memory over a long time by using a hybrid structure of tree and graph.
- Based on the hybrid structure above, we develop an effective method for retrieving the relevant memory evidence to support the QA tasks.
- Experiments on three public long-term agent memory benchmarks show that H-MEM achieves the highest overall F1 and LLM-Judge Accuracy among the evaluated methods in solving QA tasks, with a quality–cost trade-off.

## 2 RELATED WORK

### 2.1 RETRIEVAL-AUGMENTED GENERATION (RAG)

Within this landscape, RAG has become a widely used technique for incorporating external knowledge at inference time (Lewis et al., 2020; Zhou et al., 2025). Various types of RAG techniques have been studied: the naive RAG retrieves relevant passages from external corpora, graph-based RAG leverages a graph-structured index for multi-hop and relation-aware reasoning (Edge et al., 2024; Gutiérrez et al., 2024), and agentic RAG incorporates retrieval into an adaptive reasoning loop so that the model can decide when and how to retrieve during multi-step problem solving (Asai et al., 2023; Lee et al., 2024).

### 2.2 AGENT MEMORY-BASED RAG AND AGENT MEMORY MECHANISMS

The memory-based RAG techniques Wu et al. (2026) often first extract useful information from memory data, such as user preferences and events, then organize them into some index structures,

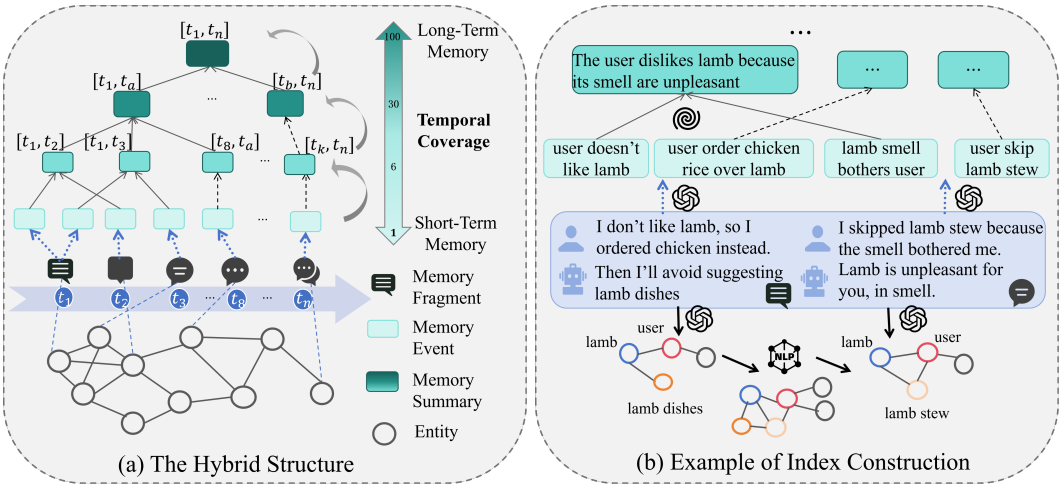


Figure 1: The offline indexing stage of H-MEM.

and finally retrieve relevant evidence and inject it into the prompt when answering a question. However, different from traditional RAG techniques, which often use static documents to provide factual grounding, the memory-based RAG techniques operate over stateful, interaction-derived memory data that evolves over time, and aim to understand context, maintain conversational coherence, and execute complex tasks. Therefore, they heavily rely on the memory mechanisms, which not only provide an effective organization of the memory data but also offer effective methods for evolving and retrieving the memory data.

Additionally, recent works have explored structured and adaptive memory mechanisms from related perspectives. A-Mem (Xu et al., 2025b) studies agentic memory mechanisms, while multi-granularity memory methods (Xu et al., 2025a) investigate memory association and selection across different abstraction levels.

## 3 OUR PROPOSED MEMORY MECHANISM H-MEM

To effectively support the memory-based RAG, we propose H-MEM, a novel memory mechanism for evolving and retrieving agent memory. H-MEM consists of two stages: **Offline Indexing** and **Online Retrieval**, where the former stage builds a hybrid structure of tree and graph, and the latter stage includes an agentic memory retrieval approach by exploiting the hybrid structure.

### 3.1 OVERVIEW

Let $\mathcal{F} = \{f_i\}$ be the set of original *memory fragments* of the memory data. The timestamp of a fragment is its ingestion time, i.e., when it enters the tree and graph; in benchmark replay, this ingestion follows the recorded conversation timeline rather than dates of events mentioned in the content. In the offline indexing stage, as depicted in Figure 1, H-MEM builds a hybrid structure for $\mathcal{F}$, mainly consisting of two parts:

- **Tree:** We build a temporal and semantic tree $\mathcal{T}$, where each node retains memory information regarding a specific semantic topic within a pre-defined time window. In the temporal view, for all the levels from the leaf to the root, their nodes have pre-defined time windows (e.g., one day, one week, one month, etc). In the semantic view, each leaf node contains a memory event extracted from an original memory fragment, and each non-leaf node stores a *memory summary*, providing an abstract and persistent representation of the fine-grained semantic topics of memory events/summaries in its child nodes.
- **Graph:** We build a knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ contains the entities extracted from the original memory fragments and $\mathcal{E}$ denotes relationships between entities. Besides, each entity is linked to the original memory fragment containing it and may have a profile.

The key rationale behind our design is that the tree supports memory evolution in both temporal and semantic dimensions, while the graph records entity-centered information and captures multi-hop relationships across fragments. In H-MEM, evolution is a provenance-preserving transformation rather than lossy replacement: original fragments remain an immutable evidence layer, whereas the derived memory state evolves from atomic events to consolidated long-term memories, records whether evidence is stable, evolving, or conflicting, and changes retrieval salience through recency and repeated reinforcement. Additionally, to enable efficient semantic search, we maintain embedding vectors of the memory fragments, events, and summaries.

### 3.2 OFFLINE INDEXING

We now introduce the construction process of the hybrid structure.

**Tree Construction.** We build a temporal and semantic tree $\mathcal{T}$ incrementally. The default configuration uses four temporal granularities—day, week, month, and year—with consolidation thresholds $0.8$, $0.7$, and $0.6$ for the upper three levels. Higher-level consolidation proceeds after temporal buckets are sealed; offline finalization completes the remaining buckets. Given timestamped memory fragments, H-MEM first extracts fine-grained atomic events and inserts each event as a leaf node. At level $l$, the new node is assigned to a window of size $\beta_l$, and only nodes in the same window are considered for semantic consolidation under threshold $\alpha_l$. If no eligible cluster exists, the event remains in a separate branch. Consolidation is selective and differs from ordinary summarization. A canonical long-term memory is created only when at least two distinct source events support the same subject and relation or attribute; otherwise, the events remain unmerged. Each consolidated node preserves its time range and source-event identifiers, together with a state label in $\{\text{STABLE}, \text{EVOLVING}, \text{CONFLICTING}\}$, stable invariants, evolution notes, conflicting claims, and confidence. When observations disagree, H-MEM records the conflict instead of selecting one side. The resulting node is propagated upward through the available levels. Thus, leaf nodes retain fine-grained short-term evidence, while upper-level nodes form provenance-aware long-term memories over broader temporal ranges.

**Graph Construction.** H-MEM also incrementally builds a knowledge graph $\mathcal{G}$. An LLM extracts entities and relations from each memory fragment; if no valid entity is returned, a named-entity recognizer provides a best-effort fallback. Entities are normalized and first matched by exact, type-compatible names, followed by bounded token-overlap and fuzzy matching. Matched surface forms are retained as aliases, whereas unmatched mentions create new nodes. Every resolved entity is linked to its original fragment, preserving provenance. Relations are mapped to resolved head and tail entities and stored with their label, timestamp, confidence, and supporting evidence. Duplicate relations merge their supporting evidence rather than creating duplicate edges. These provenance links allow retrieved graph evidence to be verified against the original fragments. For salient persons, organizations, and locations, H-MEM additionally maintains profiles containing persistent and recent memory data.

Overall, H-MEM constructs a hybrid structure with some vectors, a tree, and a graph that are built incrementally. It can not only effectively model the evolution of agent memory over a long time, but also provide a structure supporting efficient memory retrieval introduced in the next subsection.

### 3.3 ONLINE RETRIEVAL

Given a query $Q$, H-MEM identifies relevant memory evidence by searching over the hybrid structure, which consists of three steps, i.e., Retrieval Planning, Evidence Retrieval, and Generation Process.

**1) Retrieval Planning.** H-MEM first decomposes $Q$ into a list of sub-queries $\{Q_k\}_{k=1}^{K}$ with dependency relations in an agentic manner. For each $Q_k$, the planner predicts a memory scope in $\{\text{SHORT}, \text{LONG}, \text{MIXED}\}$ and a coverage mode in $\{\text{LOCAL}, \text{GLOBAL}\}$. SHORT focuses on moment-specific events, LONG on persistent consolidated memories, and MIXED on both. LOCAL targets a small number of snippets for a single incident, whereas GLOBAL aggregates evidence across multiple turns for counting, set collection, or cross-session reasoning. The planner also preserves dependency relations and explicit temporal hints, such as dates and relative time cues, and then generates a retrieval workflow for $Q_k$.

**2) Evidence Retrieval.** Following the retrieval workflow, H-MEM retrieves relevant evidence for each $Q_k$ by first exploring entities in the graph and then searching the tree in the hybrid memory structure. Specifically, by extracting entities from $Q_k$, it locates seed entities in the graph through NLP-based lexical entity matching and vector-based semantic similarity search. Starting from the seed entities, it then performs multi-hop expansion in the graph to identify more related entities.

Based on the identified entities, H-MEM further explores the tree to identify relevant memory evidence in a bottom-up manner. Specifically, it first maps the entities to their original memory fragments and then links them to memory events in the tree. These candidates supplement direct leaf-event retrieval, which remains available without matched entities. Afterwards, it searches the tree according to the inferred memory scope: If the scope is SHORT, it only uses the original memory fragments and memory events in the leaf nodes of the tree; If the scope is LONG, it only uses the memory events and memory summaries in the tree; and if the scope is MIXED, it uses the original memory fragments, memory events, and memory summaries.

For each memory evidence, denoted by $m$, H-MEM considers three aspects between $m$ and $Q_k$: *semantic similarity*, *temporal relevance*, and *memory robustness*. The semantic similarity as $S(m, Q_k)$ is derived by the cosine similarity between the embedding vectors of $m$ and $Q_k$. For temporal relevance, let $I_m = [s_m, e_m]$ denote the time interval of memory evidence $m$, and $I_k = [s_k, e_k]$ denote the temporal interval inferred from the query $Q_k$. The temporal relevance is computed by jointly considering temporal overlap and normalized center distance:

$$T(m, Q_k) = \lambda \cdot \frac{|I_m \cap I_k|}{|I_m \cup I_k| + \epsilon} + (1 - \lambda) \left(1 - \frac{|c_m - c_k|}{\max(e_m, e_k) - \min(s_m, s_k) + \epsilon}\right), \quad (1)$$

where $c_m$ and $c_k$ are the centers of $I_m$ and $I_k$, $\lambda \in [0, 1]$ controls the balance between temporal overlap and temporal distance, and $\epsilon$ is a small constant for numerical stability.

The memory robustness is used to reflect how likely a memory event or summary is retained during memory evolution. Inspired by the Ebbinghaus forgetting curve (Ebbinghaus, 1913), we formulate the memory robustness of $m$ at the query time $t$ as

$$R(m, t) = \exp\left(-\frac{t - r_m}{\tau(1 + \eta \ln(1 + n_m))}\right), \quad (2)$$

where $r_m$ is the most recent timestamp when $m$ is consolidated, $\tau > 0$ controls the forgetting time scale, $\eta \geq 0$ controls the reinforcement effect of repeated mentions, and $n_m$ is the number of times that $m$ is consolidated in the memory evolution. A higher robustness value indicates that $m$ is more recent or has been repeatedly mentioned, giving it a higher retrieval priority.

Candidate evidence is ranked by jointly considering semantic similarity, temporal relevance, and memory robustness as follows:

$$\mathcal{S}(m, Q_k, t) = \theta_1 S(m, Q_k) + \theta_2 T(m, Q_k) + \theta_3 R(m, t), \quad (3)$$

where $\theta_1$, $\theta_2$, and $\theta_3$ are non-negative weights, and $t$ is the query time of $Q$. The evidence yielding higher $\mathcal{S}(m, Q_k, t)$ scores is prioritized before reranking and deduplication to construct the final evidence chain for answering $Q$.

**3) Generation Process.** For each $Q_k$, H-MEM generates a sub-answer using its retrieved evidence. If $Q_k$ depends on other sub-queries, the answers to these prerequisite sub-queries are also used as additional context. When the first-pass evidence cannot fill a required slot, H-MEM issues at most one targeted missing-information query. The follow-up query must be anchored in the available evidence and ask for the missing bridge rather than paraphrasing the original sub-query. Finally, H-MEM synthesizes all sub-query answers as $\Psi(\{\mathcal{A}_k\}_{k=1}^{K})$, where $\mathcal{A}_k$ denotes the answer of $Q_k$ and we invoke an LLM to complete the synthesis process $\Psi$.

# 4 EXPERIMENTS

We evaluate H-MEM in terms of QA quality, efficiency, and ablation study.

## 4.1 SETUP

**Datasets.** We evaluate H-MEM on three public long-term agent memory benchmarks: **LoCoMo** (Maharana et al., 2024), **LongMemEval-S** (Wu et al., 2025), and **REALTALK** (Lee et al.,

2025). **Metrics.** We report **F1** and **LLM-Judge Accuracy** for QA performance. F1 measures token-level overlap between the predicted and reference answers. Since semantically correct answers may use different expressions, we also adopt the LLM-as-a-judge protocol used in prior long-term memory evaluation (Packer et al., 2023). The judge receives the question, gold answer, and predicted answer and determines whether the prediction is semantically correct. The judge uses the same model as the reasoning backbone within each configuration, with a shared judging prompt and temperature-zero setting across methods. For efficiency, we additionally measure indexing time, indexing token cost, storage, retrieval latency, retrieval token cost. **Baselines and Configurations.** We compare H-MEM with six representative agent memory methods: MemoryOS (Kang et al., 2025), Mem0 (Chhikara et al., 2025), MemTree (Rezazadeh et al., 2025), MemOS (Li et al., 2025), Zep (Rasmussen et al., 2025), and EverMemOS (Hu et al., 2026). We additionally report Full Context results, which use the conversation history directly as context. Full Context results on LoCoMo include two backbone settings; the LongMemEval-S and REALTALK Full Context rows use GPT-4.1-mini. These baselines cover vector-based, tree-based, graph-based, and hierarchical memory organizations, as summarized in Table 1. When the corresponding components are required, all methods use the same embedding model, reranker, answer-normalization procedure, and LLM-Judge prompt. The default retrieval stack uses Qwen3-Embedding-4B and Qwen3-Reranker-4B. Experiments are conducted on a Linux server with an Intel Xeon 2.0 GHz CPU, 1 TB memory, and eight NVIDIA A5000 GPUs, each with 24 GB of memory.

### 4.2 MAIN RESULTS

In this section, we present the main experimental results on QA performance, time/space/token efficiency.

• **QA Performance.** Tables 2, 3, and 7 report results on LoCoMo, LongMemEval-S, and REALTALK, respectively; the detailed REALTALK results are provided in Appendix A.2. Overall, H-MEM achieves the best overall performance among the evaluated methods on all three benchmarks.

Table 2: The F1 and LLM-Judge Accuracy (Acc.) for each question category on **LoCoMo**.

| Model | Method | Single Hop | | Multi Hop | | Temporal | | Open Domain | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | F1 | Acc. | F1 | Acc. | F1 | Acc. | F1 | Acc. | F1 | Acc. |
| GPT-4o-mini | Full Context | 48.37 | 83.0 | 34.26 | 67.0 | 41.82 | 56.1 | 24.73 | 49.0 | 42.95 | 72.3 |
| | MemoryOS | 42.95 | 63.85 | 35.18 | 56.74 | 44.41 | 57.63 | 24.63 | 45.83 | 40.69 | 60.13 |
| | Mem0 | 45.01 | 67.30 | 34.64 | 54.96 | 44.36 | 53.27 | 24.89 | 52.08 | 41.72 | 61.17 |
| | MemTree | 44.90 | 72.77 | 35.66 | 60.64 | 45.11 | 62.30 | 25.43 | 52.08 | 42.04 | 67.08 |
| | MemOS | 46.95 | 82.52 | 36.95 | 74.47 | 55.24 | 72.90 | 26.79 | 62.50 | 45.59 | 77.79 |
| | Zep | 46.81 | 83.23 | 33.57 | 69.15 | 55.93 | 70.09 | 24.26 | 61.46 | 44.88 | 76.56 |
| | EverMemOS | 48.72 | 89.06 | 39.03 | 85.11 | 58.79 | **82.87** | 28.11 | **64.58** | 47.76 | 85.52 |
| | H-MEM | **60.20** | **95.01** | **47.50** | **89.01** | **61.00** | 80.06 | **28.40** | 63.54 | **56.06** | **88.83** |
| GPT-4.1-mini | Full Context | 53.41 | 86.6 | 40.38 | 77.3 | 51.74 | 74.5 | 29.27 | 57.3 | 49.17 | 80.5 |
| | MemoryOS | 47.20 | 68.49 | 39.80 | 62.06 | 50.30 | 62.62 | 29.40 | 51.04 | 45.38 | 65.00 |
| | Mem0 | 49.08 | 72.53 | 38.77 | 60.28 | 49.22 | 57.63 | 29.78 | 57.29 | 46.02 | 66.23 |
| | MemTree | 48.35 | 78.36 | 40.25 | 65.25 | 50.90 | 67.29 | 30.10 | 58.33 | 46.26 | 72.40 |
| | MemOS | 52.16 | 87.63 | 41.36 | 79.79 | 61.58 | 77.88 | 30.65 | 68.75 | 50.80 | 82.99 |
| | Zep | 51.76 | 88.47 | 38.22 | 74.11 | 57.78 | 75.39 | 28.17 | 67.71 | 49.06 | 81.81 |
| | EverMemOS | 53.79 | 94.29 | 45.22 | 90.43 | 63.26 | 88.16 | 33.66 | 69.79 | 52.94 | 90.78 |
| | H-MEM | **57.67** | 96.08 | **46.81** | 89.01 | 64.00 | 89.72 | **34.90** | **72.92** | **55.58** | 92.01 |
| | H-MEM (k=30) | 56.80 | **96.31** | 46.79 | **93.62** | **64.41** | **90.03** | 30.62 | 69.79 | 54.92 | **92.86** |

On **LoCoMo**, with GPT-4o-mini, H-MEM achieves an overall F1 of 56.06, outperforming EverMemOS by 8.30 points. The single-hop and multi-hop F1 scores improve by 11.48 and 8.47 points, respectively, consistent with the intended roles of graph expansion and multi-granularity memory organization. With GPT-4.1-mini, H-MEM outperforms the evaluated baselines in overall F1 and

Accuracy, reaching 55.58 and 92.01%. The default setting is not optimal on every category: EverMemOS obtains slightly higher multi-hop Accuracy, while increasing the retrieval budget to $k = 30$ improves multi-hop Accuracy but lowers overall F1. Full Context obtains overall F1/Accuracy of 42.95/72.3% with GPT-4o-mini and 49.17/80.5% with GPT-4.1-mini on LoCoMo.

Table 3: Results by question category on **LongMemEval-S**. The Full Context row uses GPT-4.1-mini; SSU, MS, SSP, TR, KU, and SSA are the abbreviations of single-session-user, multi-session, single-session-preference, temporal reasoning, knowledge update, and single-session-assistant, respectively.

| Method | SSU | | MS | | SSP | | TR | | KU | | SSA | | Overall | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F1 | Acc. | F1 | Acc. | F1 | Acc. | F1 | Acc. | F1 | Acc. | F1 | Acc. | F1 | Acc. |
| Full Context | 68.34 | 85.7 | 31.62 | 54.1 | 12.18 | 73.3 | 37.38 | 60.2 | 50.23 | 69.2 | 60.71 | 69.6 | 43.29 | 65.4 |
| MemoryOS | 67.88 | 88.57 | 30.42 | 61.65 | 11.21 | 76.67 | 33.06 | 66.17 | 44.83 | 69.23 | 60.74 | 76.79 | 40.86 | 70.40 |
| Mem0 | 61.73 | 82.86 | 29.86 | 63.16 | 11.58 | 90.00 | 33.47 | 72.18 | 46.21 | 66.67 | 40.36 | 26.79 | 37.91 | 66.40 |
| MemTree | 69.84 | 87.14 | 34.18 | 63.91 | 10.96 | 63.33 | 36.92 | 69.17 | 48.63 | 69.23 | 68.74 | 75.00 | 44.63 | 70.60 |
| MemOS | 69.68 | 95.71 | 33.21 | 70.68 | 12.74 | **96.67** | 34.02 | 76.69 | 45.86 | 79.49 | 58.35 | 67.86 | 42.09 | 78.40 |
| Zep | 64.11 | 88.57 | 24.83 | 45.11 | 10.26 | 50.00 | 30.72 | 51.88 | 47.09 | 74.36 | 62.35 | 75.00 | 38.70 | 61.20 |
| EverMemOS | 78.03 | 97.14 | 43.27 | 73.68 | **13.52** | 93.33 | 43.88 | 77.44 | 58.61 | 89.74 | 79.46 | 83.93 | 52.96 | 82.80 |
| H-MEM | **78.70** | **98.57** | **46.70** | **83.46** | 12.08 | 93.33 | **55.35** | **84.96** | **66.60** | **91.03** | **82.35** | **96.43** | **58.50** | **89.20** |

On **LongMemEval-S**, H-MEM obtains an overall F1 of 58.50 and Accuracy of 89.20%, exceeding EverMemOS by 5.54 and 6.40 points, respectively. The largest improvement appears on temporal reasoning, where F1 increases from 43.88 to 55.35. For knowledge-update questions, F1 improves from 58.61 to 66.60 and Accuracy improves from 89.74% to 91.03%. These results suggest that H-MEM is effective on temporal-reasoning and knowledge-update questions. The method is also effective on multi-session questions, whose evidence is distributed across interactions. Its single-session-preference F1 is lower than EverMemOS, suggesting that simple questions supported by one explicit statement provide less opportunity for the hybrid structure to help. On LongMemEval-S, the GPT-4.1-mini Full Context setting obtains 43.29 F1 and 65.4% Accuracy.

On **REALTALK**, H-MEM achieves the best overall F1 and Accuracy, reaching 39.31 and 78.16%. Its multi-hop Accuracy reaches 82.39%, exceeding EverMemOS by 9.63 points and showing that entity-centered expansion remains effective in noisy real-world conversations. For temporal questions, H-MEM obtains the highest F1 but not the highest Accuracy. This suggests that implicit or noisy temporal expressions remain challenging even when the retrieved evidence has substantial lexical overlap. Full Context obtains 25.51 F1 and 59.1% Accuracy with GPT-4.1-mini on REALTALK.

• **Time/space/token efficiency.** Besides QA quality, we compare the offline indexing and online retrieval costs of H-MEM and the baselines. Figure 2 reports indexing time, indexing token cost, storage, retrieval latency, and retrieval token cost on LoCoMo.

As shown in Figure 2(a), H-MEM has a moderate indexing time. Its indexing token cost and storage are relatively high, although storage remains below EverMemOS. This overhead results from extracting atomic events, generating consolidated memories, resolving entities and relations, and maintaining the temporal-semantic tree, entity graph, and profiles. The additional cost therefore supports a richer memory representation rather than merely storing independent fragments.

For online retrieval, H-MEM has lower latency than EverMemOS, despite performing query decomposition, graph expansion, bottom-up tree search, deduplication, and reranking. Its retrieval token cost is higher than that of several simpler methods but lower than Zep. The QA improvements therefore exhibit a quality–cost trade-off under the evaluated settings.

The adaptive memory-scope planner further reduces unnecessary cost. A fixed MIXED policy obtains comparable Accuracy (92.21%) but consumes approximately $1.8\times$ as many retrieval tokens. Missing-information retrieval is also selective: it is triggered for only 6.36%, 15.15%, and 9.89% of sub-queries on LoCoMo, LongMemEval-S, and REALTALK, respectively. Thus, additional retrieval is performed only when the first-pass evidence contains a specific information gap.

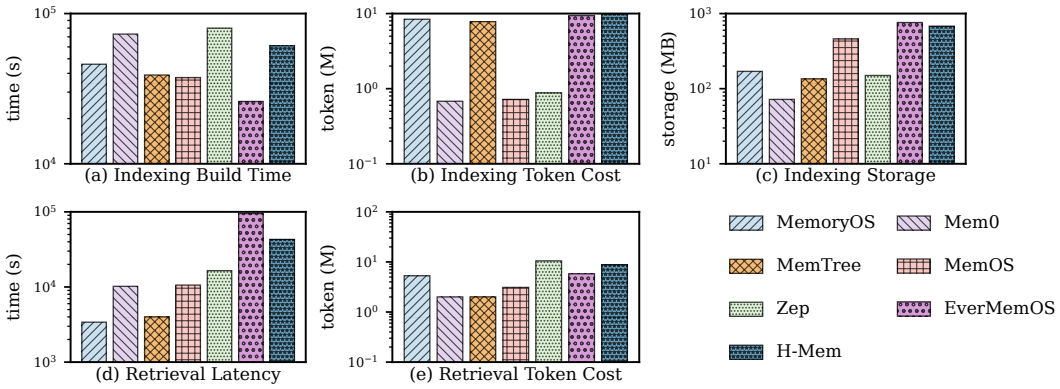


Figure 2: Cumulative indexing and retrieval costs of H-MEM and baselines on **LoCoMo**, including indexing time, indexing token cost, index storage, retrieval latency, and retrieval token cost.

### 4.3 ABLATION STUDY

Table 4: Ablation study on **LoCoMo**.

| Variant | F1 | Acc. |
|---|---|---|
| H-MEM (Full) | 55.58 | 92.01 |
| w/o tree | 49.76 | 82.73 |
| w/o graph | 52.20 | 87.86 |
| w/o long-term memory | 53.39 | 89.48 |
| w/o memory robustness | 53.72 | 89.09 |
| w/o missing-info query | 54.82 | 90.97 |
| w/o entity profile | 54.05 | 90.32 |

We ablate the tree, graph, upper-level memories, robustness score, missing-information retrieval, and entity profiles. As shown in Table 4, removing the tree causes the largest degradation: F1 decreases by 5.82 points and Accuracy by 9.28 points. This supports the importance of temporal-semantic organization and multi-level retrieval in H-MEM. Removing the graph produces the second-largest drop, reducing F1 by 3.38 points and Accuracy by 4.15 points, supporting the contribution of entity-centered relations and multi-hop expansion. Removing long-term memory decreases F1 by 2.19 points and Accuracy by 2.53 points, supporting the benefit of including consolidated memories alongside leaf-level events.

The remaining variants cause smaller but consistent reductions. Memory robustness prioritizes recent and repeatedly reinforced evidence; missing-information retrieval recovers bridge evidence absent from the first-pass results; and entity profiles provide compact access to persistent and recent entity-centered information. Overall, the gains primarily arise from cooperation between the temporal-semantic tree and entity graph, with adaptive retrieval and evidence scoring providing additional improvements.

## 5 CONCLUSION

In this work, we present H-MEM, a novel memory mechanism for evolving and retrieving agent memory via a hybrid structure. Particularly, H-MEM builds a temporal and semantic tree to organize memory data across different time windows and semantic granularities, where short-term memory data can evolve progressively into long-term memory data. It also constructs a knowledge graph to capture entity-centered information and multi-hop relations. To support the QA task, H-MEM retrieves relevant evidence by exploiting the above hybrid structure. Our method achieves the highest overall F1 and LLM-Judge Accuracy among the evaluated baselines on three public long-term memory benchmarks for the QA task. In the future, we will further improve H-MEM to support multimodal memory data, consistency-aware updates, and larger-scale distributed deployment.

## AI USE STATEMENT

Generative AI assisted with manuscript restructuring and drafting, literature discovery, interpretation of existing results and reviewer feedback, and suggestions for experimental checks. The use of LLMs within the evaluated memory system is described separately in the method and experimental setup. The authors are responsible for the final content and for verifying AI-assisted suggestions against the implementation, source literature, and experimental records.

## ETHICS STATEMENT

Conversational memory can contain personal information and sensitive inferences. Retaining source fragments and derived representations creates a need for access controls and coordinated correction or deletion across representations. The present evaluation does not establish these protections. Deployment should account for the provenance, permitted use, and retention conditions of conversational data. Performance on human-conversation benchmarks is not validation of privacy protections in an operational service.

## REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide an anonymous code supplement containing the H-Mem implementation, indexing and evaluation scripts, dependency specifications, and configuration instructions. The code package has been tested by the authors to run the indexing and evaluation pipeline end to end. Section 3 describes the memory representation and retrieval procedure, while Section 4.1 and Appendix A report the experimental setup and implementation details. Appendix C provides the core prompt templates for memory construction, evaluation, and retrieval planning.

## REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*, 2023. URL https://arxiv.org/abs/2310.11511.

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

Hermann Ebbinghaus. *Memory: A Contribution to Experimental Psychology*. Teachers College, Columbia University, New York, 1913. Original work published 1885.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024. URL https://arxiv.org/abs/2404.16130.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.14831.

Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, and Yafeng Deng. EverMemOS: A self-organizing memory operating system for structured long-horizon reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 45836–45853, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.2125. URL https://aclanthology.org/2026.acl-long.2125/.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory OS of AI agent. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of*

*the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25961–25970, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1318. URL https://aclanthology.org/2025.emnlp-main.1318/.

Dong-Ho Lee, Adyasha Maharana, Jay Pujara, Xiang Ren, and Francesco Barbieri. Realtalk: A 21-day real-world dataset for long-term conversation. *arXiv preprint arXiv:2502.13270*, 2025. URL https://arxiv.org/abs/2502.13270.

Myeonghwa Lee, Seonho An, and Min-Soo Kim. Planrag: A plan-then-retrieval augmented generation for generative large language models as decision makers. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 6537–6555, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.364. URL https://aclanthology.org/2024.naacl-long.364/.

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. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2005.11401.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, Qingchen Yu, Jihao Zhao, Yezhaohui Wang, Peng Liu, Zehao Lin, Pengyuan Wang, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhen Tao, Huayi Lai, Hao Wu, Bo Tang, Zhengren Wang, Zhaoxin Fan, Ningyu Zhang, Linfeng Zhang, Junchi Yan, Mingchuan Yang, Tong Xu, Wei Xu, Huajun Chen, Haofen Wang, Hongkang Yang, Wentao Zhang, Zhi-Qin John Xu, Siheng Chen, and Feiyu Xiong. Memos: A memory os for ai system. *arXiv preprint arXiv:2507.03724*, 2025. URL https://arxiv.org/abs/2507.03724.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://aclanthology.org/2024.acl-long.747/.

Manus AI. Manus: Experience ai that acts. https://manus.is/, 2025. Accessed: 2026-04-27.

James L. McGaugh. Memory–a century of consolidation. *Science*, 287(5451):248–251, 2000.

OpenAI. Gpt-4o mini: advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024. Accessed: 2026-05-04.

OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, 2025. Accessed: 2026-05-04.

OpenClaw Contributors. Openclaw: Personal ai assistant. https://github.com/openclaw/openclaw, 2026. Accessed: 2026-04-27.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025. URL https://arxiv.org/abs/2501.13956.

Alireza Rezazadeh, Zichao Li, Wei Wei, and Yujia Bao. From isolated conversations to hierarchical schemas: Dynamic tree memory representation for llms. In *International Conference on Learning Representations*, 2025.

Larry R. Squire and Pablo Alvarez. Retrograde amnesia and memory consolidation: A neurobiological perspective. *Current Opinion in Neurobiology*, 5(2):169–177, 1995.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=pZiyCaVuti.

Yanchen Wu, Tenghui Lin, Yingli Zhou, Fangyuan Zhang, Qintian Guo, Xun Zhou, Sibo Wang, Xilin Liu, Yuchi Ma, and Yixiang Fang. Memory in the llm era: Modular architectures and strategies in a unified framework, 2026. URL https://arxiv.org/abs/2604.01707.

D. Xu, Y. Wen, P. Jia, Y. Zhang, Wenlin Zhang, Y. Wang, H. Guo, R. Tang, X. Zhao, E. Chen, and T. Xu. From single to multi-granularity: Toward long-term memory association and selection of conversational agents. *arXiv preprint arXiv:2505.19549*, 2025a. URL https://arxiv.org/abs/2505.19549.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems*, 2025b. URL https://arxiv.org/abs/2502.12110.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025. URL https://arxiv.org/abs/2506.05176.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946.

Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, et al. In-depth analysis of graph-based rag in a unified framework. *arXiv preprint arXiv:2503.04338*, 2025. URL https://arxiv.org/abs/2503.04338.

# A Experiment Details

## A.1 Dataset and Index Statistics

This subsection provides supplementary corpus statistics and index statistics for the benchmarks used in the main paper. Table 5 summarizes the basic properties of the three datasets, including dataset scale, average conversation length, construction setting, and data source.

Table 5: Dataset details.

| Metric | LoCoMo | LongMemEval-S | REALTALK |
|---|---|---|---|
| # Conv. | 10 | 500 | 10 |
| # Questions | 1540 | 500 | 728 |
| Avg. Tokens / Conv. | 16.6K | 115.0K | 20.7K |
| Setting | LLM-simulated | LLM-simulated | Crowdsourced |
| Source | simulated conversations | synthetic assistant chats | messaging dialogues |

The three datasets cover complementary evaluation settings. LoCoMo provides dense question supervision over long multi-session dialogues, while LongMemEval-S contains substantially longer conversational histories and is therefore more challenging for long-context memory retrieval. REALTALK further complements the two LLM-simulated benchmarks with real-world human–human conversations, making it useful for evaluating robustness under noisier conversational conditions.

Table 6 reports the average index statistics after offline indexing, including raw conversation size, fragment count, hierarchical event count, and entity-graph statistics.

Table 6: Average index statistics per conversation after offline indexing.

| Metric | LoCoMo | LongMemEval-S | REALTALK |
|---|---|---|---|
| Avg. Raw Conv Size (MB) | 7.9 | 12.7 | 9.7 |
| Avg. Fragments | 596.5 | 493.5 | 894.4 |
| Avg. L1 Events | 995.1 | 1819.6 | 1209.7 |
| Avg. L2 Events | 563.7 | 1008.7 | 840.1 |
| Avg. L3 Events | 419.7 | 594.6 | 730.1 |
| Avg. L4 Events | 100.9 | 325.8 | 202.7 |
| Avg. Entities | 828.6 | 3267.6 | 1013.3 |
| Avg. Entity Edges | 3670.8 | 15911.5 | 4556.8 |

The index statistics show that different benchmarks stress different aspects of long-term memory. LongMemEval-S produces the largest event hierarchy and entity graph in these measurements, alongside the longest average conversational history. REALTALK has the largest number of fragments on average. Fragment counts also depend on segmentation, so these counts alone do not measure conversational noise. Across all datasets, H-Mem expands raw fragments into multi-level events and entity graphs, enabling retrieval at both temporal and entity-centric granularities.

## A.2 REALTALK Results

Table 7 reports F1 and LLM-Judge Accuracy by question category on REALTALK.

## A.3 Implementation Details

**Model configuration.** We evaluate H-Mem with GPT-4o-mini (OpenAI, 2024) and GPT-4.1-mini (OpenAI, 2025) as backbone LLMs. For semantic retrieval and evidence reranking, the default configuration uses Qwen3-Embedding-4B and Qwen3-Reranker-4B (Zhang et al., 2025). We also evaluate a lighter configuration with Qwen3-Embedding-0.6B and Qwen3-Reranker-0.6B (Zhang et al., 2025) in the hyperparameter analysis. All experiments are conducted on a Linux server

Table 7: Results by question category on **REALTALK**.

| Method | Multi-hop | | Temporal | | Open-domain | | Overall | |
|---|---|---|---|---|---|---|---|---|
| | F1 | Acc. | F1 | Acc. | F1 | Acc. | F1 | Acc. |
| Full Context (GPT-4.1-mini) | 29.37 | 63.1 | 22.84 | 54.9 | 22.63 | 60.2 | 25.51 | 59.1 |
| MemoryOS | 26.83 | 51.16 | 13.02 | 36.68 | 22.50 | 62.04 | 20.14 | 46.43 |
| Mem0 | 28.44 | 53.49 | 14.51 | 37.62 | 23.37 | 63.89 | 21.58 | 48.08 |
| MemTree | 31.18 | 66.45 | 25.67 | 58.93 | 25.24 | 58.33 | 27.88 | 61.95 |
| MemOS | 30.66 | 65.12 | 25.19 | 56.74 | 24.46 | 58.33 | 27.34 | 60.44 |
| Zep | 28.15 | 62.79 | 23.18 | 54.23 | 20.54 | 54.63 | 24.84 | 57.83 |
| EverMemOS | 35.78 | 72.76 | 41.77 | **80.56** | 25.03 | 71.30 | 36.81 | 75.96 |
| H-MEM | **37.41** | **82.39** | **45.59** | 74.29 | **26.03** | **77.78** | **39.31** | **78.16** |

equipped with an Intel Xeon 2.0GHz CPU, 1024GB of memory, and 8 NVIDIA RTX A5000 GPUs, each with 24GB of VRAM.

**Baseline configuration.** For each baseline, we follow its original memory organization and retrieval design as closely as possible. To reduce confounding factors, all baselines are evaluated under the same experimental framework. When semantic retrieval or reranking is required, all methods use the same embedding and reranking configurations. All methods are evaluated with the same answer simplification, normalization, F1 computation, and LLM-Judge protocol.

**Tree index hyperparameters.** For the temporal-semantic tree, we use four levels corresponding to day-, week-, month-, and year-level memory organization. The leaf level stores fine-grained memory events, while upper levels store consolidated summaries over longer temporal windows. Specifically, $\beta_l$ denotes the temporal window size at level $l$, and $\alpha_l$ denotes the semantic clustering threshold for consolidation within the corresponding temporal window. By default, we use day, week, month, and year as the temporal windows from L1 to L4, and set the consolidation thresholds of L2, L3, and L4 to $0.8$, $0.7$, and $0.6$, respectively. The threshold gradually decreases at higher levels because higher-level summaries are expected to capture more abstract and persistent memory patterns. The source configuration also considers conservative $(0.9, 0.8, 0.7)$ and aggressive $(0.7, 0.6, 0.5)$ threshold schedules; quantitative results for these alternatives are not reported here. Higher thresholds restrict merging, while lower thresholds permit broader consolidation. This design intuition should not be interpreted as a measured sensitivity result. Higher-level consolidation follows bucket sealing, and offline finalization completes the remaining buckets. Calendar weeks spanning months may be referenced by both month buckets; the hierarchy therefore permits shared lower-level evidence. Unmerged events retain hierarchy access through passthrough parents, while original fragments remain available through provenance links.

**Retention-inspired scoring hyperparameters.** For evidence ranking, H-MEM combines semantic similarity, temporal alignment, and retention-inspired salience. The event-level relevance score is computed as

$$s(m, q) = w_{\text{sem}} \cdot \text{sim}(m, q) + w_{\text{time}} \cdot \text{time}(m, q) + w_{\text{mem}} \cdot R(m, t),$$

where $\text{sim}(m, q)$ is the semantic similarity between memory $m$ and query $q$, $\text{time}(m, q)$ measures temporal alignment when an explicit temporal hint is available, and $R(m, t)$ denotes the retention-inspired salience score. We set $w_{\text{sem}} = 0.70$, $w_{\text{time}} = 0.15$, and $w_{\text{mem}} = 0.15$ by default. When no explicit temporal hint is available, the temporal alignment term is set to zero. Temporal scoring uses overlap weight $0.6$, tolerance $10^{-3}$ seconds, and numerical guard $10^{-7}$; overlap and distance components are clipped to $[0, 1]$. The center-distance term is normalized by the smallest enclosing interval.

The retention-inspired salience score follows an Ebbinghaus-style decay with reinforcement:

$$R(m, t) = \exp\left(-\frac{t - r_m}{\tau(1 + \eta \log(1 + n_m))}\right),$$

where $r_m$ and $n_m$ are the stored consolidation timestamp and count, respectively, $\eta$ controls the reinforcement effect, and $\tau$ is the decay time scale. We set $\tau = 365$ days and $\eta = 0.5$ by default. In implementation, both $t - r_m$ and $\tau$ are converted to seconds. Thus, when $\tau = 365$ days, an

unreinforced memory after one year has

$$R = \exp(-1) \approx 36.8\%,$$

corresponding to a decay of about $63.2\%$. This choice is reasonable for long-term agent memory because many user preferences, relationships, and recurring facts should remain retrievable over a year-scale horizon rather than being rapidly forgotten. The retention term has coefficient $w_{\text{mem}} = 0.15$. Its actual influence on ranking also depends on the scales and distributions of the other score terms; the coefficient alone does not establish that influence. Repeatedly reinforced memories decay more slowly through the factor $1 + \eta \log(1 + n_m)$, which allows stable and recurring memories to remain more salient during retrieval.

Stability labels describe consolidation outcomes and are not an additional ranking feature. Entity-profile updates support local additions and replacements; global correction and deletion across representations are outside the present evaluation.

**Entity extraction and disambiguation.** For graph construction, H-MEM processes each memory fragment through entity and relation extraction, entity resolution, and relation insertion. Specifically, H-MEM uses an LLM-based information extraction prompt to extract entities and relations from each memory fragment. Each extracted entity is represented as a structured record containing its surface name, entity type, optional text span, role, salience score, and auxiliary metadata. The entity type is normalized into a predefined type set, including person, organization, location, event, product, work, date, time, and other. Each extracted relation contains a source entity, a target entity, a relation label, a confidence score, and an optional text span. If LLM-based entity extraction fails or returns no valid entity, H-MEM falls back to spaCy NER to obtain a best-effort entity set.

After extraction, the extracted entities are normalized and resolved into entity nodes. The normalization step lower-cases entity names, removes unnecessary punctuation, normalizes whitespace, maps entity types into the predefined type set, and optionally uses lemmatization. For each newly extracted entity, H-MEM first checks whether it exactly matches an existing entity node with a compatible entity type. If no exact match is found, the entity is compared with existing entity names and aliases using token overlap and fuzzy string matching. If the entity satisfies the matching criteria, it is merged into the existing entity node, and its surface name is stored as an alias when applicable. Otherwise, a new entity node is created. Each resolved entity node is then linked to the original memory fragment containing it, preserving provenance for later evidence verification.

We distinguish entity merging from graph repair. Entity merging indicates that two extracted entities are resolved as the same real-world entity and therefore share one entity node. Graph repair does not merge nodes or modify aliases. Instead, for short single-token names or nickname-like variants that remain as separate nodes, H-MEM may add an *overlap* edge based on prefix/suffix matching. This edge is used only to improve graph traversal recall during retrieval and does not indicate identity equivalence.

Finally, the extracted relations are mapped to resolved entity nodes before being inserted into the graph. That is, the source and target entities of each relation are first resolved to their corresponding entity nodes. The relation is then inserted as an entity–entity edge with its relation label, confidence weight, timestamp, and supporting evidence. If the same relation between the same resolved entities already exists, H-MEM merges the supporting evidence instead of creating a duplicate edge. Since the current implementation does not rely on calibrated extraction confidence for pruning, it uses conservative type-compatible matching, bounded fuzzy matching, and provenance links to reduce the impact of extraction noise. During retrieval, provenance links provide access to the original fragments supporting entity and relation evidence. Such access does not itself guarantee that every extracted claim is verified.

### A.4 EVALUATION DETAILS

We provide the implementation details for the evaluation protocol used in the main paper.

**Answer Simplification.** Before lexical evaluation, the predicted answer is simplified into a short factual form. This step is intended to reduce verbosity effects in token-overlap scoring. Because simplification can also omit qualifications, the resulting F1 remains a lexical metric rather than a direct measure of semantic correctness. Answer simplification is only used for lexical F1 evaluation

and does not affect the generated answer used by the LLM judge. The full answer simplification prompt is provided in Appendix C.2.

**Normalization and F1 Computation.** After simplification, both the prediction and the reference answer are normalized before F1 computation. The normalization pipeline includes Unicode normalization, lower-casing, punctuation and article removal, and whitespace collapsing. Let $\hat{A}$ denote the normalized prediction and $A$ denote the normalized reference answer. After tokenization, we compute token-level precision, recall, and F1 as

$$P = \frac{|\mathrm{Tok}(\hat{A}) \cap \mathrm{Tok}(A)|}{|\mathrm{Tok}(\hat{A})|}, \quad R = \frac{|\mathrm{Tok}(\hat{A}) \cap \mathrm{Tok}(A)|}{|\mathrm{Tok}(A)|}, \quad \mathrm{F1} = \frac{2PR}{P+R}.$$

where $\mathrm{Tok}(\cdot)$ denotes the token multiset after normalization, and $\cap$ denotes multiset intersection. If both precision and recall are zero, the F1 score is set to zero.

**LLM-Judge Protocol.** For answer evaluation, we follow the LLM-as-a-judge prompt used in MemGPT (Packer et al., 2023), where the judge is given the question, the gold answer, and the predicted answer, and returns `CORRECT` or `WRONG` under a semantically tolerant criterion. The judge uses the same model as the reasoning backbone within each experimental configuration, with the same judging prompt and temperature-zero setting across methods. This standardizes evaluation within a configuration; it does not eliminate possible evaluator bias. The full LLM-Judge prompt is provided in Appendix C.2.

### A.5 Additional Experiments

We provide additional analyses of H-Mem on retrieval hyperparameters and retrieval planning behaviors, including the retriever scale, memory-scope distribution, and missing-information query behavior.

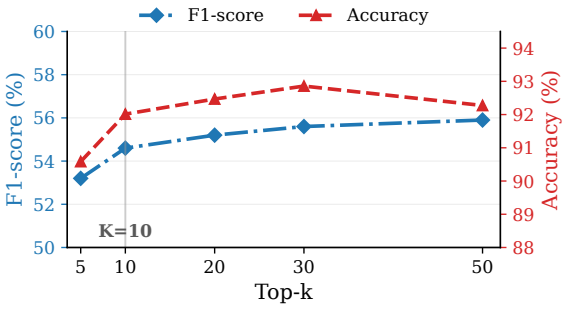


Figure 3: Sensitivity to the top-$k$ retrieval budget on **LoCoMo**.

• **Effect of top-$k$.** We analyze the sensitivity of H-Mem to the top-$k$ retrieval budget on **LoCoMo**. In our implementation, $k$ controls the budget for entity-related fragment retrieval and memory event retrieval. Figure 3 reports the F1 and LLM-Judge Accuracy under different top-$k$ settings. Overall, H-Mem remains stable across a moderate range of $k$. Increasing $k$ provides a larger candidate pool, but the main-table comparison also shows that a higher budget can improve judged correctness while lowering lexical F1. Accuracy also improves as $k$ increases from 5 to 30, but slightly decreases at $k = 50$, indicating that overly large candidate pools may introduce redundant or noisy context. The default main-table configuration uses $k = 10$. These results should be interpreted as a budget trade-off rather than evidence of monotonic improvement.

• **Effect of retrieval model components.** Table 8 compares different retrieval model configurations, including an embedding-only setting, a lighter 0.6B embedding/reranker pair, and a stronger 4B embedding/reranker pair. Compared with the embedding-only setting, adding the reranker improves both F1 and LLM-Judge Accuracy, showing that reranking helps prioritize more useful evidence for downstream reasoning. Notably, the embedding-only setting also performs well. The comparison evaluates retrieval-model size within the same memory system; it does not isolate the reason for the observed difference. The same memory representation is retained across these comparisons. The

stronger 4B configuration further achieves slightly higher F1 and accuracy than the 0.6B configuration, while the lighter model pair remains competitive. This indicates a benefit from the stronger retrieval stack within the evaluated configurations; it does not separate representation quality from all other sources of performance.

Table 8: Sensitivity to embedding and reranker models on **LoCoMo**.

| Retrieval Model | F1 | Acc. |
|---|---|---|
| Qwen3-Embedding-0.6B + Qwen3-Reranker-0.6B | 54.96 | 91.62 |
| Qwen3-Embedding-4B + Qwen3-Reranker-4B | 55.58 | 92.01 |
| Qwen3-Embedding-4B only | 53.72 | 89.81 |

• **Memory scope distribution.** We analyze the memory scopes predicted by the retrieval planner. For each sub-query, the planner selects one scope from SHORT, LONG, and MIXED, which determines the memory levels used during tree-based retrieval. SHORT is mainly used for moment-specific evidence, such as concrete events, recent actions, or temporally localized facts. LONG is used when the query requires stable or persistent memory, such as long-term preferences, relationships, or recurring facts. MIXED is selected when both fine-grained situational evidence and higher-level memory summaries are needed. Figure 4 reports the distribution of predicted memory scopes.

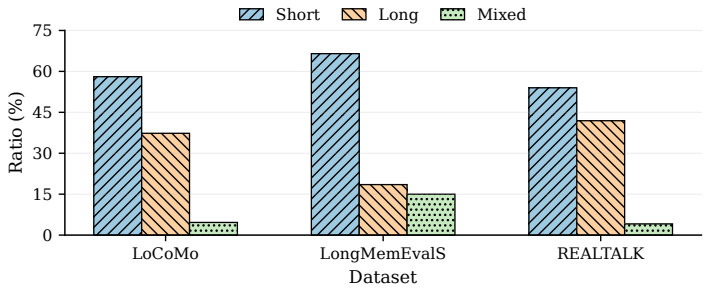


Figure 4: Distribution of memory scopes predicted by the retrieval planner.

The distribution shows that the planner does not rely on a fixed retrieval granularity. Instead, it adaptively selects the retrieval scope according to the information need of each sub-query. Across the three datasets, SHORT remains the most frequently selected scope, while LONG also accounts for a substantial portion of sub-queries, especially on LoCoMo and REALTALK. This describes the planner's assignments to different memory scopes; their frequency alone does not establish that each assignment is necessary or correct. We further compare the adaptive scope strategy with a fixed MIXED retrieval policy, where every sub-query retrieves both fine-grained events and higher-level summaries. The fixed MIXED policy also achieves comparable QA performance, with an accuracy of $92.21\%$. This provides an answer-quality comparison with fixed-scope retrieval; it does not directly measure evidence coverage. However, fixed MIXED uses a much larger evidence context and incurs about $1.8\times$ retrieval token cost compared with the planner-based strategy. Therefore, the main advantage of memory scope prediction is retrieval efficiency: it preserves comparable answer quality while reducing token usage by adaptively selecting SHORT, LONG, or MIXED for each sub-query.

• **Missing-information query analysis.** We further analyze how often H-MEM triggers missing-information queries. A missing-information query is generated only when the first-pass evidence is insufficient to answer a sub-query with the required specificity. Typical cases include unresolved entities, vague references, missing temporal anchors, underspecified event descriptions, and multiple plausible candidates. Table 9 reports the triggering statistics of missing-information queries.

Instead of simply increasing the retrieval depth for all queries, H-MEM performs targeted follow-up retrieval only when an evidence gap is detected. The generated missing-information query is

Table 9: Triggering statistics of missing-information queries.

| Dataset | Questions | Total Sub-queries | Missing-info Sub-queries | Trigger Ratio |
|---|---|---|---|---|
| LoCoMo | 1540 | 1777 | 113 | 6.36% |
| LongMemEval-S | 500 | 792 | 120 | 15.15% |
| REALTALK | 728 | 930 | 92 | 9.89% |

Table 10: Controlled stability analysis over three repeated evaluations. We reuse the same planner outputs across runs and set the temperature of all LLM-based components to 0. We report the F1 score and LLM-Judge Accuracy of each run, together with the mean and sample standard deviation. The error range corresponds to $\pm 1$ sample standard deviation.

| Dataset | F1 | | | | LLM-Judge Accuracy | | | |
|---|---|---|---|---|---|---|---|---|
| | Run 1 | Run 2 | Run 3 | Mean $\pm$ Std. | Run 1 | Run 2 | Run 3 | Mean $\pm$ Std. |
| LoCoMo | 55.58 | 55.39 | 55.66 | $55.54 \pm 0.14$ | 92.01 | 91.69 | 92.14 | $91.95 \pm 0.23$ |
| LongMemEval-S | 58.50 | 58.37 | 57.71 | $58.19 \pm 0.42$ | 89.20 | 89.00 | 88.00 | $88.73 \pm 0.64$ |
| REALTALK | 39.31 | 39.10 | 38.69 | $39.03 \pm 0.31$ | 78.16 | 77.74 | 76.92 | $77.61 \pm 0.63$ |

required to ask for the missing slot rather than paraphrasing the original sub-query, and it must contain at least one concrete anchor from the first-pass evidence. This strategy is intended to recover missing bridge evidence selectively. Trigger frequency alone does not measure how often follow-up retrieval succeeds or how much computation it saves.

### A.6 Controlled Stability Analysis

To examine the stability of H-Mem, we conduct three controlled repeated evaluations on representative main settings, including LoCoMo, LongMemEval-S, and REALTALK. Since H-Mem is a retrieval-based memory system and does not train a task-specific neural model, these repeated evaluations are used to measure evaluation-pipeline stability rather than training-run variability. To reduce accidental variation introduced by the planning stage, we reuse the same planner outputs across the three runs, including the decomposed sub-queries and predicted memory scopes. In addition, all LLM-based components are evaluated with temperature set to 0.

For each setting, we report the LLM-Judge Accuracy and F1 score of each run, together with the mean and sample standard deviation across the three runs. The reported error range corresponds to $\pm 1$ sample standard deviation.

The small standard deviations characterize repeated evaluation conditional on fixed planner outputs and temperature-zero decoding. They do not measure variability from independently reconstructing the memory or regenerating plans, and do not establish statistical significance relative to another method.

### A.7 Assets and Licenses

We use existing benchmarks, baselines, and model APIs or publicly available models in our experiments. The datasets and baselines are cited in the main paper and used only for research evaluation. For model usage, we follow the corresponding API terms of service or model usage terms. We do not redistribute any third-party datasets, model weights, or proprietary model outputs as new assets.

## B Case Studies

We present two representative qualitative examples of H-Mem. The first illustrates multi-step sub-query decomposition for a multi-entity question, while the second illustrates missing-information guided retrieval, where the system detects that first-pass evidence is insufficiently specific and issues a bridge-style follow-up query.

**(A) Case Study: Multi-step Sub-query Decomposition**

**Question.** What subject have Caroline and Melanie both painted?

**Planner output.** H-MEM decomposes the question into two entity-specific subqueries:

- **q1**: What subjects has Caroline painted? (SHORT, GLOBAL)
- **q2**: What subjects has Melanie painted? (SHORT, GLOBAL)

**Retrieved evidence.**

- **q1**: Caroline shared paintings involving sunsets and floral themes.
- **q2**: Melanie shared paintings including a lake sunrise, a sunset scene, and other nature-inspired subjects.

**Prediction. sunsets**
**Gold answer. Sunsets**

**(B) Case Study: Missing-Information Guided Retrieval**

**Question.** Which popular music composer's tunes does Tim enjoy playing on the piano?

**First-pass evidence.** The first retrieval pass finds evidence that Tim enjoys playing *a theme from a movie he really likes* on the piano, but the composer is not explicitly named.

**Reasoner decision.** The first-pass reasoner marks the subquery as `missing_info=true`, because the current evidence is suggestive but still underspecified for answering the question at the required level of specificity.

**Generated missing-info query.**

> Which popular music composer created the theme from the movie that Tim enjoys playing on the piano?

**Second-pass evidence and answer.** The follow-up retrieval uncovers the missing bridge between Tim's favorite movie theme and the intended composer, allowing H-MEM to produce the final answer: **John Williams**

**Gold answer. John Williams**

Figure 5: Two representative qualitative examples of H-MEM. (A) H-MEM decomposes a multi-entity question into two entity-specific sub-queries and retrieves evidence under global coverage before final synthesis. (B) When first-pass evidence is insufficiently specific, H-MEM explicitly detects missing information and generates a bridge-style follow-up query for targeted second-pass retrieval.

# C PROMPT TEMPLATES

This appendix provides the core prompt templates used by H-MEM. We include prompts for memory construction, evaluation, retrieval planning, and missing-information query generation.

## C.1 MEMORY CONSTRUCTION PROMPTS

**Memory extraction prompt**

```
You are an information extraction engine.
Your job is to extract the maximally fine-grained set of atomic,
    queryable memory units from conversation fragments.

Return ONLY valid JSON. No markdown. No explanations.

Definition:
```

```
- A memory unit is the smallest self-contained piece of information
    that could answer a future query.
- Memory units include events, facts, plans, preferences,
    relationships, statuses, goals, challenges, collections/items,
    and named-entity attributes.
- Each unit must be grounded in the provided fragments via frag_ids.
- Do NOT write narrative summaries or high-level commentary.

Granularity rules:
- Split aggressively: prefer many small units over fewer large ones.
- One unit = one claim.
- If a sentence lists multiple items, create one unit per item.
- If a statement contains multiple attributes, create separate units
    for each attribute.
- Keep units independently queryable.

Hard requirements:
Output JSON with the exact top-level key: "events".
Each memory unit must include:
- "event_text": a single canonical sentence describing one claim.
- "frag_ids": list of supporting fragment ids.
- "time_range": [start_ts, end_ts] as unix timestamps in seconds.
- "event_type": one of ["episodic","plan","preference","fact","
    relationship","status","procedure","other"].

Additional rules:
- Preserve named entities, numbers, dates, locations, and constraints
    .
- Do not invent unsupported details.
- Resolve vague references when the referent is clear from nearby
    fragments.
- Deduplicate only if two units express the same claim with the same
    specificity.
- Include short-lived chatter only if it creates a queryable unit.

Cross-fragment change and linkage:
- Consider how fragments change or relate to earlier fragments.
- If a meaningful change is detected, extract an additional unit
    describing the change.
- Change types include plans, preferences, goals, status, roles,
    locations, and availability.
- If a later fragment answers, corrects, or refers to an earlier
    fragment, extract a linkage unit if it is queryable.
- Delta units should use templates such as:
  "Previously <X>; now <Y>." or "Earlier <X>; later <Y>."

Optional fields:
- participants
- location
- confidence
- notes

If no meaningful units can be extracted, return:
{"events": []}
```

**Memory consolidation prompt**

```
You are a long-term memory consolidation engine.
Your job is to consolidate lower-level memory units into stable, long-term consolidated events.

Return ONLY valid JSON. No markdown. No explanations.

Key idea:
Consolidation is not summarization.
It converts repeated, stable, or high-signal units into canonical long-term memories.
Do NOT force coverage; many lower-level units may remain unmerged.

Hard constraints:
- Output JSON with top-level keys: "consolidated_events" and "unmerged_event_ids".
- Only output a consolidated event if it is supported by at least two distinct source events.
- Each consolidated event must have source_event_ids length >= 2.
- Do NOT invent facts. Preserve names, numbers, dates, and constraints.
- Each consolidated event should represent one stable memory.
- If the evidence evolves, record the stable core and evolution notes.
- If the evidence conflicts, record the conflict rather than choosing one side.

Merge rules:
- Only merge units that share the same subject/entity and the same relation or attribute.
- Prefer type-homogeneous merges.
- For itemized lists, consolidate into a collection memory only when all items share the same relation.
- Do not over-generalize beyond the evidence.
- If clusters are provided, merge only within the same cluster.

Output schema:
{
  "consolidated_events": [
    {
      "consolidated_text": "string",
      "memory_kind": "stable_fact|preference|ongoing_plan|relationship|recurring_theme|status|other",
      "time_range": [number, number],
      "participants": ["string", ...],
      "entity_hints": ["string", ...],
      "source_event_ids": ["string", ...],
      "stability": "stable|evolving|conflicting",
      "invariants": ["string", ...],
      "evolution": ["string", ...],
      "conflicts": [
        {
          "claim_a": "...",
          "claim_b": "...",
          "source_event_ids_a": [...],
          "source_event_ids_b": [...]
        }
      ],
      "confidence": number
    }
  ],
  "unmerged_event_ids": ["string", ...]
}
```

### C.2 Evaluation Prompts

**LLM-Judge prompt**

```
Your task is to label an answer to a question as "CORRECT" or "WRONG".
You will be given:
(1) a question,
(2) a gold answer,
(3) a generated answer.

The question asks about something one user should know about another user based on prior conversations.
The gold answer is usually concise.
The generated answer may be longer, but should be counted as CORRECT if it refers to the same fact or topic as the gold answer.

For time-related questions, consider different surface forms of the same date or time period as CORRECT.
For example, "May 7th" and "7 May" should be considered equivalent if they refer to the same date.

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

Return ONLY valid JSON:
{"label": "CORRECT"} or {"label": "WRONG"}.
```

**Answer simplification prompt**

```
You are an answer simplifier.
You will be given a question and a generated answer.
Extract only the minimal core fact that directly answers the question.

Rules:
- Do NOT add explanations or extra context.
- Preserve names, numbers, dates, and units when they are part of the answer.
- If the generated answer contains multiple facts, keep only the fact required by the question.
- Output MUST be valid JSON only.

JSON schema:
{"answer": <string>}

Question: {question}
Generated Answer: {answer}
```

### C.3 Retrieval Planning Prompts

**Subquery planner prompt**

```
You are to decompose the user's query into atomic subqueries and their dependencies.
Your goal is to produce subqueries that are SMALLER, MORE CONSTRAINED, and SEMANTICALLY SINGLE-PURPOSE than the original query, never broader.

You must think silently and output ONLY ONE JSON object.
```

```
No chain-of-thought, no analysis, no <think>, no markdown, no code fences.

JSON schema:
- "subqueries": a list of objects each with:
    - "id": "q1","q2",...
    - "text": string (the atomic subquery)
    - "memory_scope": one of ["SHORT","LONG","MIXED"]
    - "coverage_mode": one of ["local","global"]
    - "type_hint": optional short string hint for unit type (e.g., "intention","procedural","preference","episodic","semantic"); may be null
    - "deps": list of subquery ids this subquery depends on (may be empty)
    - "hint_time": null or [start_timestamp, end_timestamp] in seconds if time-constrained
- "dependency_graph": object mapping subquery id -> list of dependency subquery ids (prerequisites).

CRITICAL PRINCIPLE (DO NOT VIOLATE):
- Each subquery must preserve ALL explicit constraints from the user query that are relevant to answering it.
- NEVER drop or weaken constraints such as:
  - time windows / dates / relative-time anchors
  - named entities / subjects / objects
  - locations
  - quantities / counts
  - qualifiers like "major", "first", "most recent", "in that meeting", "during that trip", etc.
- If a constraint exists in the user query, it MUST appear in the subquery text OR be represented as hint_time when appropriate.
- Subqueries must not broaden the search space. They should reduce it.

TIME WINDOW RULES:
- If the query includes a clear absolute time (e.g., a month/year/date), keep it in the subquery text.

DECOMPOSITION RULES:
- Prefer 1-5 subqueries. If the user query is already atomic and well-scoped, output exactly 1 subquery that restates it faithfully (do NOT split).
- HARD RULE: If you output EXACTLY 1 subquery, its "text" MUST be EXACTLY IDENTICAL to the full original user query (verbatim). Do NOT paraphrase, shorten, translate, or reformat.
- Decompose only when it reduces difficulty by isolating distinct required pieces (e.g., identify entity, then retrieve evidence, then synthesize).
- When decomposing, subqueries should become more specific via:
  - separating different targets (different people/items/events) into separate subqueries, while preserving the original constraints for each target
  - separating different time windows or temporal anchors into separate subqueries, especially when one date/time/event must be used as the reference point for another step
  - separating anchor resolution from calculation/comparison/synthesis: first retrieve the date/entity/event that acts as the anchor, then use it in dependent subqueries
- NEVER create a subquery that is a strict superset of another subquery's scope unless it is explicitly needed for global coverage questions.

ATOMICITY & SEMANTIC PURITY (VERY IMPORTANT):
```

```
- EACH subquery must represent exactly ONE information need / ONE intent / ONE semantic predicate.
- A subquery must be answerable by ONE short response (or one focused list) without needing to address a second, separate question.
- DO NOT combine multiple asks in one subquery. Avoid conjunction-style bundling such as:
    - "and", "or", "as well as", "also", "+"
- If the user query contains multiple intents, split them into multiple subqueries.
- If a question contains "what" + "why" + "how" (or similar), split:
    - e.g., (1) what happened? (2) why did it happen? (deps=[(1)]) (3) how was it decided? (deps=[(1)])
- If the query asks for comparison or trade-off (A vs B), prefer:
    - one subquery to retrieve evidence about A,
    - one subquery to retrieve evidence about B,
    - optional synthesis subquery "Compare A vs B based on evidence" with deps on both.
- If you find yourself writing a long subquery with multiple clauses, split it.

DEPENDENCIES:
- Use deps only for true prerequisites (e.g., resolve an ambiguous referent before retrieving evidence).
- Do NOT produce circular dependencies: ensure the dependency_graph is acyclic.

COVERAGE MODE (local vs global):
- Keep coverage_mode="local" for single-incident, single-moment questions that can be answered from a small number of best-matching snippets.
- Use coverage_mode="global" whenever answering requires combining evidence from MULTIPLE turns.
    This includes aggregation intents even if the user does NOT explicitly say "all"/"so far".
    MUST set coverage_mode="global" for any counting/aggregation intent, e.g.:
        - "count ...", "how many ...", "number of ...", "total ...", "times ...", "frequency ...", "rate ..."
    Also set coverage_mode="global" for set-collection intents, e.g.:
        - "list ...", "names of ..."

MEMORY SCOPE:
- SHORT: use for episodic or situational dialogue evidence tied to a specific moment, date, meeting, action, decision, or temporary context.
  Examples: "meeting is tomorrow at 3pm", "the user bought a tennis racket yesterday", "focus on page 4 of the uploaded file".

- LONG: use for stable, reusable, or profile-like memory that remains useful beyond the original dialogue moment.
  such as enduring preferences, personal background, identity, education, occupation, relationships, hobbies, long-term goals, persistent statuses, and general procedures.
  Examples: "the user prefers concise answers", "the user graduated with a degree in Business Administration", "the user's favorite running shoe brand is Nike".

- MIXED: use when answering the query requires both LONG-style stable memory and SHORT-style situational evidence.
  Example: "the user generally prefers concise writing, but this specific email to the CEO should be more formal".
OUTPUT MUST be valid JSON following the schema above. Do NOT include any explanation, analysis, or markdown.
```

```

User query:
{QUERY_TEXT}
```

**Missing-information query prompt**

```
Return EXACTLY ONE compact single-line JSON object.
No chain-of-thought, no analysis, no <think>, no markdown, and no code fences.
Required key: missing_info_query.

You are generating ONE actionable retrieval query to fill the missing slot(s) AFTER a reasoner has already produced a preliminary answer.

Input is a JSON object with keys:
query (string),
subquery (string), and
reason_output (object with keys conclusions, confidence, and missing_info).

Missing-info policy:
missing_info MUST be true whenever the available evidence cannot answer the subquery as asked with the required specificity.
This includes, but is not limited to:
(a) the question asks for a specific value, but the evidence only provides a vague reference;
(b) the answer requires resolving an ungrounded pronoun or placeholder;
(c) there are multiple plausible candidates.

Anti-echo rule:
The missing-information query MUST NOT be a paraphrase or rewrite of the subquery.
It must ask for a missing slot explicitly suggested by the evidence but not specified.
It must include at least one concrete anchor from the evidence, such as a named entity, object, or time phrase.
If the subquery asks for X but the evidence only mentions Y, ask about Y first as a bridge step.
The query should be minimal and ordered by dependency.

Example:
Subquery: Which airline did John fly to Tokyo?
Evidence: John said, ``I took a red-eye to Tokyo last night.''
Good: {"missing_info_query":"Which airline operated the red-eye flight John took?"}
Bad: {"missing_info_query":"Which specific airline did John fly to Tokyo?"}

Output JSON only.
```

## D MEMORY INDEX STRUCTURES

Under the memory-based RAG paradigm, existing methods differ not only in how memory data are indexed before retrieval, but also in their retrieval mechanisms which determine how relevant evidence is accessed from the index. According to the memory index structures, existing memory mechanisms can be roughly classified into three categories as reported in Table 1. A first class of methods adopts the *vector index*, a single-level organization of memory, in which memory fragments are stored as independent entries. To enable efficient memory retrieval, these fragments are often encoded as embeddings, and some methods further store these embeddings in a vector database (Zhong

et al., 2024; Chhikara et al., 2025). A second class of methods mainly explores the *tree index*, where the semantic topics of the memory data are hierarchically organized across multiple levels, with lower levels preserving fine-grained semantic topics and higher levels providing the abstract or persistent representations of fine-grained semantic topics (Rezazadeh et al., 2025; Kang et al., 2025; Li et al., 2025; Hu et al., 2026). To query the memory fragments about a specific topic, they just need to traverse explicitly along the tree structure in a bottom-up or top-down manner. A third class of methods mainly uses the *graph index*, where entities and relationships are represented as nodes and edges, respectively (Rasmussen et al., 2025). By following the link relationships between entities, they can naturally support fast relational retrieval and multi-hop reasoning.