LayerMem: Query-Adaptive Hierarchical Memory via Sleep-Time Updates

Anonymous ACL submission

Abstract

Memory systems for LLM agents have advanced rapidly, yet how to align stored representations with future queries remains an open challenge: facts are extracted and stored in the language they were expressed, while queries arrive in the language of the user's intent. We propose LayerMem, a layerwise hierarchical memory system that closes this *extraction-query domain gap* through a *sleep-time update*. After ingesting content into a four-layer hierarchy, LayerMem simulates the future query distribution by generating synthetic queries from stored trajectories, uses retrieval outcomes to identify gaps, and applies a credit-assignment procedure to plant cross-layer connections and enrich stored representations. The enriched representations and learned connections allow future queries to navigate the hierarchy even when their framing diverges from the stored content, without any model training or inference-time overhead. Evaluated on LoCoMo and GVD, LayerMem outperforms all six baseline methods on both benchmarks. Targeted ablations further demonstrate the contribution of each component in the sleep-time update.

1 Introduction

Long-running LLM agents rely on memory to give accurate, personalized responses across sessions (Liang et al., 2025; Zhang et al., 2025). Such memory is typically built by extracting salient facts from each interaction and storing them persistently, with no dedicated model training required. At query time, relevant facts are retrieved and used to ground the agent's response. Yet storing facts faithfully does not guarantee they can be retrieved effectively: a user who asks *"does my colleague prefer remote work?"* may have their answer stored as *"Alex mentioned he hates commuting"*, and the two describe the same fact from entirely different angles. A core challenge is the *extraction-query domain gap*: memory representations capture how facts were originally phrased, while queries reflect the user's intent, and the two often diverge. Because the future query distribution is unknown at ingestion time, this gap cannot be closed by extraction alone.

Existing memory systems offer limited remedies to the extraction-query domain gap. *Flat retrieval* systems embed documents and queries in a shared space (Chhikara et al., 2025); this works well when query and source representations align, but degrades as they diverge. *Hierarchical* systems improve retrieval precision by organizing content at multiple abstraction levels (Sun and Zeng, 2025; Hu et al., 2025; Han et al., 2025), yet fix their structure at construction time with no mechanism to adapt it to how the memory will actually be queried. *Memory consolidation* systems, such as A-Mem (Xu et al., 2025), take a step toward adaptability by consolidating and linking existing memories, but do not revisit stored representations in light of the latent query distribution. *Learning-based* approaches train memory agents with reinforcement learning to optimize memory operations for downstream retrieval outcomes (Wang et al., 2025; Yan et al., 2025; Ma et al., 2026), but require a dedicated model training phase. None of these systems closes the extraction-query domain gap at inference time: most fix stored representations with no mechanism for alignment, while those that attempt it require model training.

We propose LayerMem, a memory architecture that closes the extraction-query domain gap through a *sleep-time update*. The key insight is that stored memory can be retroactively reorganized to align with the actual query distribution: by simulating future queries after ingestion and grading retrieval outcomes against the source trajectory as ground truth, LayerMem identifies structural gaps and corrects them. Concretely, content is ingested into a four-layer hierarchy: Concepts, Reflections, Trajectory Summaries, and Trajectories.

A credit-assignment procedure evaluates each synthetic query's retrieval outcome and takes corrective action: enriching stored representations with missed or discriminative facts, and planting cross-layer connections that route similar future queries to the correct evidence. Because this process runs between interactions, it requires no model training and adds no inference-time latency.

We evaluate the system on the LoCoMo conversational memory benchmark (Maharana et al., 2024) and the GVD probing-question benchmark (Zhong et al., 2024). Using `gpt-4o-mini` (OpenAI, 2024a) for all memory operations, the system achieves 84.2% LLM-judge accuracy on LoCoMo and 94.3% on GVD, outperforming all baselines on both.

Our main contributions are:

1. A four-layer hierarchical memory architecture with learned cross-layer connections between the Concept and Reflection layers, designed to support sleep-time updates.
2. A sleep-time update procedure that closes the extraction-query domain gap via synthetic query simulation, a credit-assignment decision tree that drives targeted re-extraction and connection learning, and persona maintenance.
3. An empirical evaluation on LoCoMo and the GVD benchmark demonstrating strong retrieval and answer-generation performance, outperforming all baselines on both datasets.

2 Related Work

We review relevant work in LLM agent memory systems, offline experience processing, learning-based memory optimization, and hierarchical memory organization. Most of these systems make no attempt to align stored representations with the query distribution. Learning-based approaches are an exception, but require a dedicated model training phase. Our approach addresses the extraction-query domain gap at inference time with a sleep-time update, which simulates future queries in order to guide targeted re-extraction and connection learning, without any model training.

2.1 LLM Agent Memory Systems

LLMs lack persistent long-term memory, and external memory systems have addressed this through a variety of designs (Liang et al., 2025; Zhang et al., 2025; Packer et al., 2023; Rasmussen et al., 2025; Liu et al., 2023; Chhikara et al., 2025; Salama et al., 2025; Lee et al., 2024). MemoryOS draws on operating-system design, organizing memory into short-term, mid-term, and long-term tiers and promoting content between tiers based on recency and access frequency (Kang et al., 2025). MemoryBank stores timestamped conversations aggregated into daily and global summaries, and applies Ebbinghaus-style forgetting based on retrieval recency and frequency (Zhong et al., 2024). LightMem classifies tokens by importance, chunks interactions into episodes using attention and semantic boundaries, and updates stored episodes between inference calls (Fang et al., 2025). A-Mem constructs a self-organizing memory graph inspired by Zettelkasten note-taking, with offline restructuring of existing memory nodes (Xu et al., 2025). None of these systems proactively reorganizes stored representations to align with the query distribution.

2.2 Offline Experience Processing

Several systems use LLMs to derive higher-level representations from past experience, though the scope of what is processed varies. Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2023) extract lessons from successful or failed task episodes to improve future agent behavior. R2D2 combines such experience with trajectory memory for web navigation (Cao et al., 2025). MemWeaver distinguishes behavioural memories from cognitive memories encoding user preferences (Yu et al., 2026). These approaches enrich representations from past experience but do not revisit stored content in light of how it will subsequently be retrieved.

2.3 Learning-Based Memory Optimization

Recent work frames memory management as a policy learning problem, training LLM agents to make memory operations via reinforcement learning. Mem-α equips agents with a tripartite memory structure (core, semantic, episodic) and optimizes memory construction using GRPO, rewarding correctness, compression, and semantic validity (Wang et al., 2025). Memory-R1 fine-tunes a memory manager and an answer agent jointly with PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024), using downstream exact-match score as the sole reward signal (Yan et al., 2025). Fine-Mem addresses sparse reward and credit assignment by introducing chunk-level step rewards and evidence-anchored reward attribution, tracing which stored

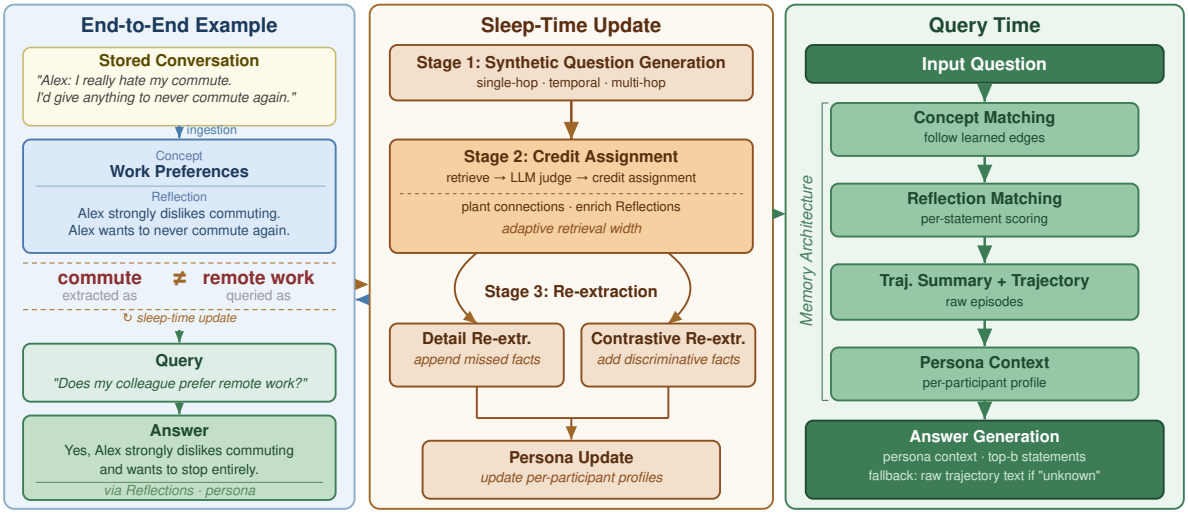

Figure 1: LayerMem overview. *Left:* End-to-end example illustrating the extraction-query domain gap: a stored conversation is ingested; the sleep-time update simulates future queries, observes retrieval failures, and restructures the memory to close the gap. *Center:* Sleep-time update pipeline: synthetic questions are generated per trajectory, retrieval is simulated, and a credit-assignment procedure (see Fig. 2) plants learned connections and queues re-extractions. *Right:* Query-time retrieval cascade traversing the memory hierarchy from Concepts through Trajectories to assemble evidence for answer generation. Orange and blue arrows indicate sleep-time read/write access; green indicates query-time access to the enriched hierarchy.

items contributed to correct answers (Ma et al., 2026). These systems optimize memory for retrieval outcomes but require a dedicated training phase; our sleep-time update achieves the same alignment at inference time, without any model training.

2.4 Hierarchical Memory Architectures

Hierarchical memory reduces retrieval cost and improves precision by organizing information at multiple abstraction levels. H-MEM introduces a coarse-to-fine hierarchical memory structure, reducing the number of semantic comparisons during retrieval (Sun and Zeng, 2025). HiAgent decomposes past tasks into subgoal hierarchies, exposing only high-level summaries for agent interaction (Hu et al., 2025). LEGOMem separates task-level and subtask-level memories with explicit links between them (Han et al., 2025). PISA organizes memories into schema-structured buckets (Jia et al., 2025). MAGMA represents information across four specialized graph types (Jiang et al., 2026). All of these systems fix their structure at construction time, with no mechanism to adapt stored representations to the query distribution. Our sleep-time update addresses this directly, simulating future queries offline and applying a credit-assignment procedure that uses retrieval outcomes to drive targeted re-extraction and connection learning.

3 Methods

LayerMem is built on a four-layer memory hierarchy (Section 3.1) and operates in three stages. During *ingestion* (Section 3.2), raw text is segmented into episodes and processed by a single extraction call that populates the hierarchy bottom-up. During the *sleep-time update* (Section 3.3), the system generates synthetic queries from stored content, simulates retrieval, and applies a credit-assignment procedure to structurally enrich the hierarchy. At *query time* (Section 3.4), a hierarchical cascade traverses the enriched hierarchy to assemble evidence for answer generation. Figure 1 illustrates the entire approach.

3.1 Memory Architecture

The memory hierarchy comprises four layers of increasing abstraction. Layers are linked by directed edges; the Concept-to-Reflection connections are learned and mutable, updated by the sleep-time procedure, while lower-layer links are static.

Concepts (Layer 1) are short topical labels extracted from each trajectory.

Reflections (Layer 2) store lists of self-contained factual statements extracted from each trajectory.

Trajectory Summaries (Layer 3) are short descriptions of each trajectory, linking downward to the raw trajectory and upward to its Reflection.

Trajectories (Layer 4) store raw text with source

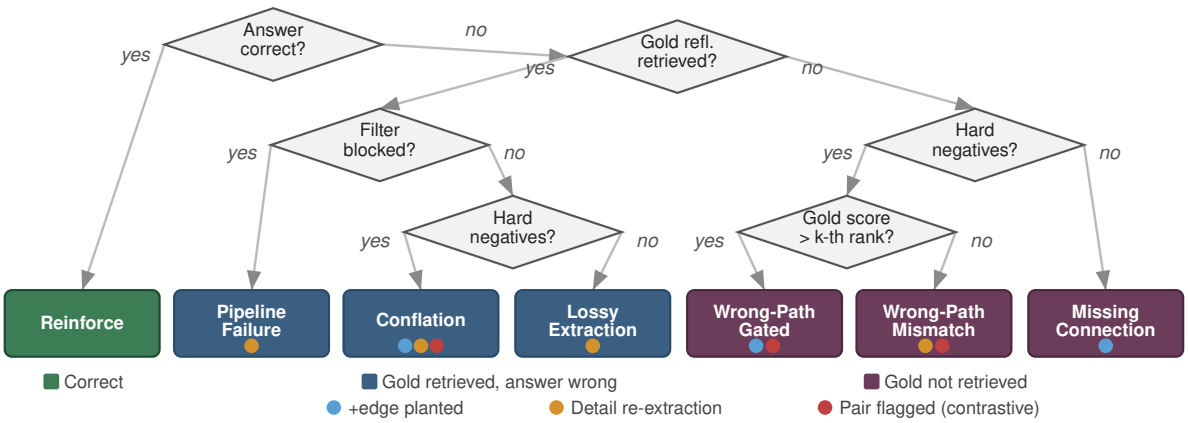

Figure 2: Credit-assignment decision tree applied to each training question during sleep-time connection learning. Leaf color indicates the structural branch: green = correct answer (no change); teal = gold Reflection was retrieved but answer was wrong; plum = gold Reflection was not retrieved. Badges show which corrective actions are triggered: • edge planted, • detail re-extraction queued, • pair flagged for contrastive re-extraction.

metadata and timestamps. They are the leaves of the hierarchy.

Persona Memory maintains a cross-session profile summary for each conversation participant.

3.2 Content Ingestion

Sessions are first segmented into topically coherent episodes via an LLM that identifies natural topic-shift boundaries. For each episode, a single LLM call extracts three components: 2–5 short Concept labels, 5–10 self-contained Reflection statements, and a one-sentence Trajectory Summary, with the episode text stored as a Trajectory. Concept labels are then matched against existing Concepts by cosine similarity: labels above the threshold are merged into the nearest existing Concept, while novel labels instantiate new ones. Initial edges are planted from the matched or new Concepts to the Reflection.

3.3 Sleep-Time Update

3.3.1 Connection Learning

Connection learning is the central mechanism for bridging the extraction-query domain gap, targeting which Reflections are surfaced by the cascade. It proceeds in three stages: synthetic question generation, retrieval simulation with credit assignment, and re-extraction.

Stage 1: Synthetic question generation. For each Trajectory, an LLM generates synthetic questions in three categories: n_{sh} single-hop factual questions, n_{t} temporal questions requiring date reasoning, and n_{mh} multi-hop questions. When question generation is disabled (n=0 for all types), the raw trajectory text serves as a proxy query, allowing the sleep-time update to run at reduced cost.

Multi-hop pairs are preferentially drawn from trajectories that share at least one Concept, falling back to random pairing otherwise. The resulting multi-hop questions thus span topically related episodes when possible. Temporal questions are paired with two gold answer forms, a relative expression and an absolute date, so that either form is accepted during grading. Questions are processed in curriculum order (single-hop, temporal, then multi-hop), so that edges from simpler question types are available when the more demanding multi-hop cases are processed.

Stage 2: Credit assignment. The system runs the hierarchical retrieval cascade using the synthetic query, then grades the answer using the same synthesis pipeline as at evaluation time (Section 3.4). An LLM judge scores the answer against the gold. *Hard negatives* are non-gold Reflections scoring above a noise threshold τ, indicating that the gold Reflection is being confused with topically similar content. Each hard negative causes the source trajectories of the gold and confuser Reflections to be recorded as a *flagged* pair; pairs accumulating at least τ_{flag} flags trigger contrastive re-extraction in Stage 3. The credit-assignment tree branches on retrieval outcome, answer correctness, and hard negative presence; its seven leaves are illustrated in Figure 2:

Reinforce Answer correct: no structural change.

Pipeline failure gold Reflection retrieved; gold statements below synthesis budget: detail re-extraction queued.

Conflation gold Reflection retrieved with hard negatives present: edge planted; detail re-extraction queued; pair flagged.

Lossy extraction gold Reflection retrieved; gold statements in context; answer wrong: detail re-extraction queued.

Wrong-path gated gold Reflection not retrieved; gold score above the w-th candidate cutoff: edge planted; pair flagged.

Wrong-path mismatch gold Reflection not retrieved; gold score below the w-th candidate cutoff: detail re-extraction queued; pair flagged.

Missing connection gold Reflection not retrieved; no hard negatives: edge planted.

For multi-hop questions, the tree is applied independently to each source trajectory, treating each hop's gold Reflection as a separate credit-assignment target.

Adaptive retrieval-width update. After grading, the retrieval width w for Reflections is set to the minimum value that would have surfaced the gold Reflection for every incorrectly answered training question, maximizing recall while delegating precision to statement-level scoring and the synthesis statement budget.

Stage 3: Re-extraction. Two complementary passes enrich Reflection content based on credit assignment outputs. *Detail re-extraction* is triggered for trajectories queued by the Pipeline Failure, Conflation, Lossy Extraction, or Wrong-Path Mismatch branches: an LLM reads the raw Trajectory alongside the existing Reflection and generates 2–3 new statements not yet captured. For Pipeline Failure cases, re-extraction is preferred over widening the synthesis budget because it raises the scores of the relevant statements directly, rather than admitting more potentially irrelevant ones. *Contrastive re-extraction* is triggered for each flagged pair: a single LLM call generates 2–3 new Reflection statements unique to the gold Trajectory and absent in the confusers, sharpening the gold Reflection's discriminability.

3.3.2 Reflection Consolidation

After re-extraction, Reflections within overloaded Concept clusters are consolidated. A Concept is overloaded if its linked Reflection count exceeds the mean plus two standard deviations across all Concepts. Within each overloaded cluster, pairs of Reflections whose cosine similarity exceeds δ are merged: statement lists are LLM-deduplicated and structural links are absorbed by the surviving Reflection.

3.3.3 Persona Update

Conversation texts from sources ingested since the last sleep are collected and passed to an LLM that updates per-participant persona summaries. New entries are merged with existing ones; participants absent from the current batch are preserved.

3.4 Retrieval and Answer Generation

3.4.1 Hierarchical Retrieval Cascade

The input question is embedded into a single query vector, which drives each step of the cascade.

Concept matching. The query embedding is matched against all stored Concept embeddings by cosine similarity; the top-w Concepts are selected, where retrieval width w is set per-conversation by the adaptive mechanism described in Section 3.3.

Reflection matching. The set of candidate Reflections is the union of Reflections reachable from matched Concepts via the learned Concept-to-Reflection connections. Each candidate Reflection is scored by per-statement cosine similarity between the query embedding and stored statement embeddings. Each Reflection's score s_r is the mean of its top-2 statement scores, balancing supporting context against noise: single-statement scoring amplifies outliers, while averaging all statements dilutes scores for information-dense Reflections. The top-w Reflections are returned.

Temporal re-ranking. Questions containing temporal language are screened by keyword match and an LLM call that determines whether a precise date range is derivable. When LLM confidence is high, per-Reflection scores are adjusted as

$$s'_r = \alpha_s \cdot s_r + (1-\alpha_s) \cdot \mathbf{1}[t_r \in [d_{\text{start}}, d_{\text{end}}]],$$

where s_r is the Reflection matching score, t_r is the session timestamp of Reflection r's trajectory, $[d_{\text{start}}, d_{\text{end}}]$ is the date range derived by the LLM, and α_s is the temporal re-ranking weight.

Trajectory Summary and Trajectory matching. All Trajectory Summaries reachable from matched Reflections via the static Reflection-to-summary connections are retrieved; the linked Trajectories are returned accordingly.

Table 1: LoCoMo results (LLM-judge accuracy, `gpt-4o-mini`). LayerMem achieves the best result in every category, with the largest per-category margin on multi-hop (+10.1 points over LightMem). SE shown for overall accuracy. Best result per column is **bold**.

Method	Temporal	Single-hop	Multi-hop	Open-domain	Overall	SE
LightMem	0.788	0.824	0.693	0.507	0.773	0.003
A-Mem	0.617	0.782	0.567	0.410	0.685	0.005
Naive RAG	0.503	0.751	0.629	0.608	0.668	0.003
MemoryOS	0.456	0.735	0.584	0.497	0.634	0.004
MAGMA	0.543	0.729	0.441	0.288	0.615	0.004
MemoryBank	0.487	0.687	0.449	0.347	0.581	0.005
LayerMem (ours)	**0.805**	**0.893**	**0.794**	**0.649**	**0.842**	0.006

3.4.2 Answer Generation

Reflection statements are collected from the matched Reflections and tagged with session timestamps. Statements are scored by cosine similarity to the query, and the top-b by the synthesis statement budget are selected. The selected statements are then re-grouped by their source Reflection: groups are ordered by each group's highest-scoring statement, and statements within each group appear in their original extraction order. The most relevant persona entry, selected by cosine similarity between the query embedding and stored persona embeddings, is prepended as a user-context note.

If the primary answer is `"unknown"`, a trajectory fallback is triggered: relevant Trajectory Summaries are selected by an LLM relevance filter, the corresponding raw trajectories are retrieved, and a second answer attempt is made directly from the passage text.

4 Experiments

4.1 Experimental Setup

We evaluate on two benchmarks. **LoCoMo** (Maharana et al., 2024) consists of long-horizon dyadic conversations, each spanning multiple sessions separated by days or weeks. We use all ten available conversations, yielding 1,540 questions across four categories: single-hop (841), temporal (321), multi-hop (282), and open-domain (96); adversarial questions are excluded. **GVD** (Zhong et al., 2024), a probing-question benchmark introduced alongside MemoryBank, provides 10-day conversation histories for 15 users with 5–7 probing questions each (100 questions total); no gold answers are provided. In both settings, all sessions of a conversation are fully ingested before the sleep-time update runs.

Setup. End-to-end answer quality is measured by LLM-as-judge accuracy (binary correct/wrong per question). For LoCoMo, `gpt-4o-mini` grades each generated answer against the provided gold answer using the correctness prompt in Prompt 11. For GVD, because no gold answers are provided, `gpt-5` (OpenAI, 2026) instead reads the full 10-day conversation history and judges correctness directly (Prompt 12). `gpt-4o-mini` (OpenAI, 2024a) is used for all memory operations; embeddings use `text-embedding-3-small` (OpenAI, 2024b). All reported results are averaged over three runs; standard errors are reported alongside means.

Baselines. We compare against six baselines spanning retrieval-based methods (Naive RAG; MemoryBank (Zhong et al., 2024)), graph-based memory (A-Mem (Xu et al., 2025); MAGMA (Jiang et al., 2026)), OS-inspired hierarchical memory (MemoryOS (Kang et al., 2025)), and compression-based memory (LightMem (Fang et al., 2025)). All systems are evaluated with the same LLM-as-judge protocol. Implementation details are given in Appendix E.

4.2 Results

LoCoMo. LayerMem achieves the best result in every category, improving over the strongest baseline (LightMem) by 6.9 points overall (Table 1). Single-hop questions achieve the strongest accuracy (89.3%), reflecting the Reflection layer's direct factual coverage and connection learning ensuring correct routing despite framing divergence.

Temporal questions score second (80.5%), with a large margin over five of six baselines (45–62%). LayerMem addresses this through two complementary mechanisms: temporal re-ranking adjusts Reflection scores by session timestamp when a date range is derivable, and temporal synthetic questions during sleep train the cascade to surface time-anchored evidence.

Multi-hop questions show lower absolute accuracy (79.4%), yet the gap over the next-best baseline (LightMem, 69.3%) is 10 points, the largest per-category margin. This is the setting connection learning is most directly designed for: cross-

Table 2: GVD benchmark results (LLM-as-judge accuracy, `gpt-5` (OpenAI, 2026) reads the full conversation history because no gold answers are provided). LayerMem achieves 94.3%, outperforming all six baselines. Best result is **bold**.

Method	Accuracy	SE
LightMem	0.703	0.015
A-Mem	0.823	0.009
Naive RAG	0.910	0.006
MemoryOS	0.900	0.012
MAGMA	0.727	0.007
MemoryBank	0.887	0.003
LayerMem (ours)	**0.943**	0.003

Table 3: Additive ablation on LoCoMo (LLM-judge accuracy). Each row adds one component to all preceding ones. The connection learning row uses single-hop synthetic questions only; the final row adds temporal and multi-hop question diversity.

Configuration	Accuracy	SE
Architecture only (no sleep)	0.652	0.004
+ Persona updates	0.693	0.003
+ Adaptive retrieval width	0.766	0.003
+ Connection learning	0.831	0.001
+ Temporal & multi-hop (full)	**0.842**	0.006

episode edges planted during sleep provide direct routing between Reflections that would otherwise be unlinked, allowing facts to be surfaced from separate sessions. Open-domain questions are weakest (64.9%) across all methods, consistent with the difficulty of anchoring answers to expressed preferences rather than recorded facts.

GVD. LayerMem achieves 94.3% on GVD, outperforming all baselines (Table 2), though the margin over the best baseline (Naive RAG, 91.0%) narrows to 3.3 points compared to the 6.9-point advantage on LoCoMo. This contraction is expected: GVD's 10-day histories are short enough that basic retrieval can surface relevant evidence without needing to bridge an extraction-query domain gap.

Four baselines score above 82%, consistent with GVD's short histories requiring less memory specialization than LoCoMo's long multi-session conversations. The most notable reversal is LightMem: the strongest LoCoMo baseline (77.3%) scores lowest on GVD (70.3%). Compression-focused designs trade completeness for conciseness; on LoCoMo this helps manage long-horizon noise, but on GVD's precise factual questions it discards the specific details the judge locates in the full conversation history.

The contrast with LoCoMo confirms that the sleep-time update's advantage scales with history length: baselines range from 58–77% while LayerMem holds at 84.2%. As histories lengthen across many sessions, the extraction-query domain gap grows and the update's advantage compounds.

4.3 Ablation Study

Table 3 reports an additive ablation starting from the bare four-layer architecture and progressively enabling each sleep-time component. The full sleep-time update contributes +19.0 points over the architecture-only baseline. Components are added in order of increasing interaction with the retrieval mechanism: persona updates operate independently; adaptive width then calibrates the candidate pool; connection learning exploits that pool; and question diversity extends its coverage.

Persona updates contribute +4.1 points. This gain reflects the value of persistent user-level context: persona summaries capture recurring preferences and background facts that span episodes, providing the answer generator with a characterization of the participant that no single Reflection holds. Questions requiring this broader understanding of a person benefit directly.

The adaptive retrieval width provides the largest single gain (+7.3 points). The mechanism sets the candidate width to the minimum value that surfaces the gold Reflection for every incorrectly answered training question, balancing two competing pressures: too narrow a pool risks missing gold evidence, while too wide a pool degrades toward brute-force retrieval over all Reflections, losing the precision advantage of concept-based routing. By targeting the minimum sufficient width, the mechanism maximizes recall without sacrificing selectivity. The adapted w averages 21.8 across the ten conversations (from $w_0 = 5$), with the minimum sufficient width varying per conversation in a way no fixed value could match; Appendix B confirms that expanding uniformly to all Reflections harms precision.

Connection learning adds a further +6.5 points (using single-hop synthetic questions only). With recall secured by the widened pool, the credit-assignment procedure plants edges that route future queries to the right Reflections even when their framing diverges from stored content, closing the extraction-query domain gap beyond what width alone achieves. Adding temporal and multi-hop synthetic questions yields the final +1.1 points, confirming that question diversity during sleep improves generalization across question types.

Results are robust to the main hyperparameter choices. Sensitivity details are reported in Appendix A. Ingestion cost and per-query latency comparisons are reported in Appendix C.

4.4 Error Analysis

Table 4: Failure taxonomy for the 225 incorrect answers by tier and question category (SH = single-hop; Temp = temporal; MH = multi-hop; OD = open-domain). Sub-types subdivide the Synthesis tier only.

Tier	SH	Temp	MH	OD	Total
Retrieval	2	0	0	0	2
Synthesis	64	46	49	26	185
Distractor	61	41	49	26	177
Temp. conf.	2	4	0	0	6
Misinterp.	1	1	0	0	2
Budget-cap	15	9	9	4	37
Extraction	0	0	0	1	1
Total	81	55	58	31	225

We classify 225 incorrect answers from a single representative run into four failure tiers by question category (Table 4). *Retrieval failure* occurs in only 2 cases, confirming that the sleep-time update has effectively closed the extraction-query domain gap. For the remaining 223 cases, a `gpt-4o-mini` judge determines where in the pipeline the answer was lost.

The dominant tier is *Synthesis failure* (185 cases, 82% of all errors): the gold answer was present in the synthesis context but the model selected a different statement. The Distractor sub-type, where the model selects semantically related but incorrect content over the gold, accounts for 177 of these; notably, all multi-hop and open-domain in-context failures are Distractor, confirming that the lowest-scoring categories (79.4% and 64.9%) are limited by synthesis precision rather than retrieval. The 6 temporal-confusion cases, where the model identifies the right fact type but from the wrong session, represent the residual gap that temporal re-ranking does not fully close. Table 5 shows two representative cases. Example 1 illustrates the dominant sub-type: the model selects semantically related but incorrect content over the gold, despite both being present in the synthesis context. Example 2 shows the same tier: the model correctly names one child but does not enumerate the second, despite the answer generation prompt explicitly instructing enumeration of all statements.

Budget-cap failure accounts for 37 cases, where gold Reflection statements rank below the synthesis statement budget cutoff; the failure is in statement-level ranking within the synthesis budget, not in retrieval. *Extraction failure* accounts for 1 case. Even combining retrieval and budget-cap failures, gold evidence failed to reach the synthesis context in only 39 of 1,540 questions (2.5%), confirming that synthesis precision is the primary bottleneck.

Table 5: *Synthesis failure* cases. The gold answer was present in the synthesis context in both cases: Example 1 shows the model selecting related but incorrect content; Example 2 shows incomplete enumeration.

Example 1 (multi-hop)

Q:	What do Melanie's kids like?
Gold:	dinosaurs, nature
Pred:	Melanie's kids like learning about animals, going to the beach, camping, pottery workshops, and painting together.

Example 2 (multi-hop)

Q:	What are the names of John's children?
Gold:	Kyle, Sara
Pred:	John has a one-year-old son named Kyle.

5 Conclusion

We present LayerMem, a memory system whose central contribution is a sleep-time update that closes the extraction-query domain gap. After ingestion, the system simulates future queries offline and learns cross-layer connections that re-align stored representations with the query distribution, requiring no model training and adding no inference-time latency.

Evaluated on LoCoMo across 1,540 questions, the system achieves 84.2% LLM-judge accuracy, with the largest per-category margin on multi-hop (+10 points over the next-best baseline). On GVD the system achieves 94.3%, outperforming all six baselines. Ablation results confirm that connection learning is a central contributor, with additional gains from temporal and multi-hop synthetic question diversity. Error analysis confirms that retrieval is not the bottleneck: gold evidence failed to reach the synthesis context in only 39 of 1,540 questions (2.5%), with *Synthesis failure* accounting for the large majority of errors.

Several directions remain open. First, the credit-assignment procedure targets retrieval structure; extending it to statement-level scoring would directly address the Budget-cap failure tier, where gold statements are retrieved but ranked below the synthesis cutoff. Second, multi-hop and open-domain questions remain the weakest categories; error analysis shows their failures are exclusively synthesis failures, so chain-of-thought reasoning over retrieved content or synthesis re-ranking represent the most direct path forward.

6 Limitations

Synthesis bottleneck. The dominant failure mode is *Synthesis failure* rather than *Retrieval failure*: in 82% of incorrect answers, the gold evidence was present in the prompt but the model selected a wrong statement. This is a limitation of the answer-generation step that memory improvements alone cannot fully address; gains here likely require dedicated re-ranking or chain-of-thought reasoning over retrieved content. A secondary synthesis-path failure occurs when gold statements are retrieved but fall below the fixed synthesis budget cutoff; an adaptive budget, or a sleep-time procedure that targets statement-level scoring, represents a natural extension.

Judge model overlap. On LoCoMo, `gpt-4o-mini` serves as both the answer generator and the LLM-as-judge evaluator, which may introduce self-preference bias (Wataoka et al., 2024). However, the judge grades binary correct/wrong decisions against provided gold answers, and all baselines are evaluated under the same protocol, so relative comparisons remain valid. On GVD, `gpt-5` serves as judge, avoiding this overlap entirely.

LLM dependency and cost. All extraction, sleep-time, and answer-generation operations rely on a proprietary LLM. Ingestion cost is approximately 1.3M tokens across GVD and 6.5M across LoCoMo's ten longer conversations, which may be prohibitive at scale or in cost-sensitive deployments. Replacing the LLM backbone with a different model would require re-validation of extraction quality.

Evaluation scope. Both benchmarks used in this work consist of English dyadic conversations. Whether the system generalizes to multi-party, multilingual, or domain-specific settings remains untested.

7 Ethical considerations

LayerMem persistently stores personal information extracted from user conversations, including factual statements, persona summaries, and session timestamps. This creates privacy risks if the memory store is accessed by unauthorized parties or deployed in contexts where users have not consented to long-term retention of personal data. The sleep-time update compounds this by enriching stored representations with additional extracted detail, increasing the sensitivity of the stored content over time. Deployments should apply appropriate access controls, offer users the ability to inspect and delete stored memories, and comply with applicable data-retention regulations.

References

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

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. In *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025)*, Frontiers in Artificial Intelligence and Applications, pages 2993–3000. IOS Press.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. 2025. Lightmem: Lightweight and efficient memory-augmented generation. *CoRR*, abs/2510.18866.

Dongge Han, Camille Couturier, Daniel Madrigal Díaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. 2025. Legomem: Modular procedural memory for multi-agent LLM systems for workflow automation. *CoRR*, abs/2510.04851.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2025. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 32779–32798. Association for Computational Linguistics.

Shian Jia, Ziyang Huang, Xinbo Wang, Haofei Zhang, and Mingli Song. 2025. PISA: A pragmatic psych-inspired unified memory system for enhanced AI agency. *CoRR*, abs/2510.15966.

Dongming Jiang, Yi Li, Guanpeng Li, and Bingzhe Li. 2026. Magma: A multi-graph based agentic memory architecture for ai agents.

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

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John F. Canny, and Ian Fischer. 2024. A human-inspired reading agent with gist memory of very long contexts.

In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 26396–26415. PMLR / OpenReview.net.

Jiafeng Liang, Hao Li, Chang Li, Jiaqi Zhou, Shixin Jiang, Zekun Wang, Changkai Ji, Zhihao Zhu, Runxuan Liu, Taolin Ren, Jinlan Fu, See-Kiong Ng, Xia Liang, Ming Liu, and Bing Qin. 2025. Ai meets brain: Memory systems from cognitive neuroscience to autonomous agents.

Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. 2023. Think-in-memory: Recalling and post-thinking enable llms with long-term memory. *CoRR*, abs/2311.08719.

Weitao Ma, Xiaocheng Feng, Lei Huang, Xiachong Feng, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Bing Qin. 2026. Fine-mem: Fine-grained feedback alignment for long-horizon memory management. *ArXiv*, abs/2601.08435.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 13851–13870. Association for Computational Linguistics.

OpenAI. 2024a. Gpt-4o system card. *CoRR*, abs/2410.21276.

OpenAI. 2024b. New embedding models and API updates. Accessed: 2026-05-20.

OpenAI. 2026. Openai GPT-5 system card. *CoRR*, abs/2601.03267.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. 2023. Memgpt: Towards llms as operating systems. *CoRR*, abs/2310.08560.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pages 963–981. Association for Computational Linguistics.

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

Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. 2025. Meminsight: Autonomous memory augmentation for LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 33136–33152. Association for Computational Linguistics.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*.

Haoran Sun and Shaoning Zeng. 2025. Hierarchical memory for high-efficiency long-term reasoning in llm agents. *ArXiv*, abs/2507.22925.

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

Koki Wataoka, Tsubasa Takahashi, and Ryokan Ri. 2024. Self-preference bias in llm-as-a-judge. *CoRR*, abs/2410.21819.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for llm agents. *ArXiv*, abs/2502.12110.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Hinrich Schutze, Volker Tresp, and Yunpu Ma. 2025. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *ArXiv*, abs/2508.19828.

Shuo Yu, Mingyue Cheng, Daoyu Wang, Qi Liu, Zirui Liu, Ze Guo, and Xiaoyu Tao. 2026. Memweaver: A hierarchical memory from textual interactive behaviors for personalized generation. In *Proceedings of the ACM Web Conference 2026, WWW 2026, Dubai, United Arab Emirates, originally scheduled for April 13-17, 2026, rescheduled for June 29 - July 3, 2026*, pages 6920–6931. ACM.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025. A survey on the memory mechanism of large language model-based agents. *ACM Trans. Inf. Syst.*, 43(6):155:1–155:47.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Y. Liu, and Gao Huang. 2023. Expel: Llm agents are experiential learners. In *AAAI Conference on Artificial Intelligence*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 19724–19731. AAAI Press.

A Hyperparameter Settings and Sensitivity

Table 6: Hyperparameter values used in the reported experiments.

Component	Parameter	Symbol	Value
Ingestion	Concept merge threshold	—	0.65
Sleep-time	Questions per type	$n_{\text{sh,t,mh}}$	1, 1, 1
Sleep-time	Hard negative threshold	τ	0.3
Sleep-time	Contrastive flag count	τ_{flag}	2
Sleep-time	Max confusers per call	N_{conf}	3
Sleep-time	Initial retrieval width	w_0	5
Sleep-time	Reflection merge threshold	δ	0.85
Retrieval	Temporal weight	α_s	0.7
Generation	Synthesis statement budget	b	60

Table 6 lists hyperparameter values used in the experiments.

Table 7 reports sensitivity to three key hyperparameters on LoCoMo. The concept merging threshold controls whether a new concept label is merged into an existing one (higher threshold = more separate concepts). The Reflection merging threshold controls the same for Reflection lists. The synthesis statement budget controls how many Reflection statements are passed to the answer generator; a budget of 60 is roughly equivalent to a traditional top-6 retrieval given a typical Reflection list length of $\sim$10 statements.

Table 7: Hyperparameter sensitivity on LoCoMo (LLM-judge accuracy). Default values (used in all other experiments) are **bold**.

Parameter	Value	Accuracy	SE
Concept merge threshold	0.55	0.842	0.003
	0.65	**0.842**	**0.006**
	0.75	0.838	0.005
Reflection merge threshold	0.75	0.729	0.005
	0.85	**0.842**	**0.006**
	0.95	0.837	0.004
Synthesis statement budget	50	0.840	0.004
	60	**0.842**	**0.006**
	70	0.843	0.006

The Reflection merge threshold has the largest effect: a threshold of 0.75 merges Reflections too aggressively, losing distinct facts and dropping accuracy by 11.3 points. Thresholds of 0.85 and 0.95 perform similarly. The concept merge threshold shows modest sensitivity ($\pm$1 point around the default). For the synthesis statement budget, 60 and 70 statements perform comparably; we use 60 as it requires fewer tokens with negligible accuracy cost.

B Brute-Force Retrieval Baseline

To assess the value of concept-based routing, we run a variant in which all Reflections are placed in the candidate set at every query, bypassing the Concept layer entirely, while keeping the full sleep-time update and answer generation unchanged. Table 8 shows the result.

Table 8: Comparison of concept-routed retrieval vs. brute-force retrieval over all Reflections (LoCoMo, LLM-judge accuracy). Sleep-time update is active in both.

Configuration	Accuracy	SE
All Reflections (no concept routing)	0.826	0.002
Full system (concept routing)	0.842	0.006

Without sleep-time training, the architecture-only system scores 0.652 (Table 3). With sleep-time connection learning, concept-routed retrieval (0.842) surpasses brute-force retrieval over all Reflections (0.826). Learned connections not only recover the recall of exhaustive search but also improve precision by routing queries away from irrelevant Reflections, maintaining the scalability of the hierarchy without sacrificing accuracy. The Concept layer's primary design motivation is sub-linear retrieval cost as memory grows; the accuracy gain over brute-force reflects the added precision of concept-based routing at this scale.

C Computational Efficiency

Tables 9 and 10 report ingestion token cost and per-query answer latency on GVD and LoCoMo, respectively. Ingestion tokens cover all LLM calls during memory building (extraction, sleep-time updates, summarization). Latency is measured sequentially (one question at a time) without the judge call. LightMem's compression step uses LLMLingua-2 (Pan et al., 2024) (355M parameters), run on an NVIDIA A40 GPU for approximately 10 hours total across both benchmarks. All other components use OpenAI API models (`gpt-4o-mini`, `gpt-5`, `text-embedding-3-small`); parameter counts for these models are not publicly disclosed.

Table 9: Efficiency comparison on GVD. Ingestion tokens are totals across all 15 users; latency is mean ± std per question (sequential, no judge). LightMem ingestion uses LLMLingua-2 (local GPU) for compression; token counts reflect LLM API calls only.

Method	Ingest tok.	Ans. tok./Q	Latency (s)
Naive RAG	0	1,457	1.17 ± 0.42
A-Mem	1,301K	454	0.76 ± 0.75
MemoryOS	425K	2,518	1.95 ± 1.58
MemoryBank	251K	558	1.17 ± 0.70
MAGMA	764K	1,653	1.44 ± 0.53
LightMem	235K†	1,960	0.99 ± 1.71
LayerMem	1,270K	1,820	1.52 ± 1.17

On GVD, LayerMem's ingestion cost is comparable to A-Mem (1.27M vs. 1.3M tokens), reflecting the cost of sleep-time synthetic question generation and connection learning. Per-query answer token costs are mid-range, and sequential latency (1.52s) is modest given the multi-layer retrieval cascade. MemoryOS has the highest per-query latency (1.95s) despite moderate ingestion cost, reflecting its tiered routing overhead. A-Mem has the lowest answer token cost due to its compact memory format.

Table 10 reports the same metrics on LoCoMo. Ingestion costs are substantially larger than on GVD due to LoCoMo's longer multi-session histories; LayerMem's 6.5M tokens reflect the cost of sleep-time question generation and connection learning over ten conversations. MAGMA shows the highest per-query latency by a wide margin (6.72s; see Appendix E for implementation notes), well above all other methods. LayerMem's latency (1.98s) remains modest given the multi-layer retrieval cascade.

Table 10: Efficiency comparison on LoCoMo. Ingestion tokens are totals across all 10 conversations; latency is mean ± std per question (sequential, no judge). LightMem ingestion uses LLMLingua-2 (local GPU) for compression; token counts reflect LLM API calls only.

Method	Ingest tok.	Ans. tok./Q	Latency (s)
Naive RAG	0	3,210	1.85 ± 1.85
A-Mem	7,131K	932	1.07 ± 0.81
MemoryOS	4,162K	5,154	2.29 ± 1.74
MemoryBank	705K	667	1.52 ± 1.49
MAGMA	3,982K	2,034	6.72 ± 3.71
LightMem	1,243K†	2,630	1.51 ± 2.02
LayerMem	6,519K	2,756	1.98 ± 1.92

D Prompts

```
System:
You identify natural episode boundaries
    in conversations. Respond in JSON.

User:
You are given the turns of a single
    conversation session.

CONVERSATION TURNS:
{numbered_block}

Identify up to {max_splits} points where
     the conversation shifts to a
    genuinely new, independently
    meaningful topic or episode. Return
    the turn numbers AFTER which to
    split (1-based). For example, if
    turns 1 9  form one episode and
    turns 10 20  another, return [9].

Constraints:
- Each resulting episode must contain at
     least {min_turns_per_episode} turns
    .
- Only split at clear topic/subject
    changes     not mid-topic or at
    minor asides.
- If there are no natural divisions,
    return an empty list.
- Return at most {max_splits} split
    points. Do not feel pressured to use
     all {max_splits} if fewer would be
    more natural.

Return JSON with "split_after": list of
    1-based turn numbers after which to
    split.
```

Prompt 1: Episodic chunking prompt (ingestion). Values used: max_splits=4, min_turns_per_episode=3.

```
System:
You extract structured memory components
     from text. Respond in JSON.

User:
Extract memory components from the
    following text chunk.

TEXT:
{chunk_text}

Return JSON with:
- "concept_texts": list of 2-5 short
    topic labels (1-4 words each) that
    categorise this chunk
- "reflection_list": list of 5-10
    specific factual statements
    capturing ALL key information from
    this chunk. Be comprehensive
    include specific names, dates,
    places, quantities, decisions, and
    personal details. Each statement
    must be self-contained and
    understandable without the original
    text. Preserve precise details (e.g.
     exact dates, locations, names)
    rather than generalising them.
- "trajectory_summary": one concise
    sentence ( 30  words) summarising
    what this chunk is about
```

```
Return ONLY the JSON.
```

Prompt 2: Content extraction prompt (ingestion).

```
System:
You generate factual training questions
    from conversation text. Respond in
    JSON.

User:
Generate {n} specific factual question(s
    ) from this conversation episode.

EPISODE:
{chunk_text}

Each question must be answerable from
    this episode alone.
Questions must ask about WHAT, WHO, or
    HOW     not WHEN or time-related
    aspects.
Do NOT use temporal qualifiers such as '
    recently', 'last time', 'before', '
    after', or 'when'.
For each question provide:
- "question": the question text (do not
    reference "the episode" or "the text
    ")
- "gold_answer": a concise correct
    answer (1-3 sentences)
- "expected_facts": list of 1-3 key
    facts from the episode needed to
    answer

Return JSON: {"questions": [{"question":
     "...", "gold_answer": "...", "
    expected_facts": ["..."]}]}
```

Prompt 3: Single-hop training question generation prompt (sleep-time).

```
System:
You generate time-based training
    questions from conversation text.
    Respond in JSON.

User:
Generate {n} temporal question(s) from
    this conversation episode (session
    timestamp: {timestamp}).

EPISODE:
{chunk_text}

Each question must name a specific event
     or fact and ask WHEN it occurred
        the date or time must appear
    only in the answer, never in the
    question itself.
Good examples: 'What date did Alice
    receive her test results?', 'When
    did Bob start his new job?'
Each question must be answerable from
    this episode.
For each question provide:
- "question": the question text (do not
    reference "the episode" or "the text
    ")
- "gold_answer_relative": the answer
    expressed as a relative time (e.g. "
    last Saturday", "three weeks ago", "
    yesterday")
- "gold_answer_absolute": the answer
    expressed as an absolute date/time
    resolved using the session timestamp
     (e.g. "Saturday 14 October 2023").
    Always provide both forms.
- "expected_facts": list of 1-3 key
    facts from the episode needed to
    answer

Return JSON: {"questions": [{"question":
     "...", "gold_answer_relative":
    "...", "gold_answer_absolute":
    "...", "expected_facts": ["..."]}]}
```

Prompt 4: Temporal training question generation prompt (sleep-time).

```
System:
You generate multi-hop training
    questions from conversation episodes
    . Respond in JSON.

User:
Generate 1 question that requires
    information from BOTH episodes to
    answer correctly.

EPISODE A:
{chunk_text_a}

EPISODE B:
{chunk_text_b}

The question must be impossible to
    answer from either episode alone
     the answer must combine facts from
    both.
Provide:
- "question": the question text (do not
    reference "the episode" or "the text
    ")
- "gold_answer": a concise correct
    answer drawing on both episodes
- "expected_facts": list of 2-4 key
    facts needed, note which episode
    each comes from

Return JSON: {"questions": [{"question":
     "...", "gold_answer": "...", "
    expected_facts": ["..."]}]}
```

Prompt 5: Multi-hop training question generation prompt (sleep-time).

```
System:
You maintain concise persona summaries
    for every participant in a
    conversation. Respond in JSON.

User:
Current persona summaries (keyed by
    participant name, may be empty):
{current_fmt}

Recent conversations:
```

```
{conv_text}

Instructions:
1. Identify every distinct participant /
     speaker in the conversations (e.g.
    "Alice", "Bob", "User", "Assistant",
     or any name/role that appears).
2. For each participant, write or update
     a detailed persona summary that
    captures their identity, interests,
    background, preferences, personality
    , and communication style as
    revealed by the conversations.
3. Each individual summary must be at
    most 200 words.
4. Only update a participant's summary
    if the new conversations add
    meaningful information.
5. Preserve participants from the
    current summaries even if they do
    not appear in the new conversations.

Return JSON with a single key "
    participants" whose value is an
    object mapping each participant's
    name to their updated summary string
    .
Example shape: {"participants": {"Alice
    ": "...", "Bob": "..."}}
```

Prompt 6: Persona update prompt (sleep-time).

```
System:
You extract dense factual memory from
    conversation text. Respond in JSON.

User:
Here is a conversation episode and its
    current memory reflections.

Episode:
{chunk_text}

Existing reflections:
{existing}

Generate 2-3 NEW reflection statements
    capturing specific facts, details,
    or nuances present in the episode
    that are not yet captured above. Do
    not repeat anything already listed.
    Each statement must be fully self-
    contained (use names, not pronouns).

Return JSON with "new_reflections": list
     of statement strings.
```

Prompt 7: Detail re-extraction prompt (sleep-time).

```
System:
You extract discriminative memory
    statements. Respond in JSON.

User:
The following conversation episode is
    being confused with {n_confusers}
    other episode(s) during retrieval.

Episode A (improve this one):
{chunk_text}

Confusing episode(s):
{confuser_section}

Existing reflections for Episode A:
{existing}

Generate 2-3 NEW reflection statements
    for Episode A that capture facts or
    details unique to Episode A and
    absent or clearly different in ALL
    of the confusing episodes above. Do
    not repeat anything already in the
    existing reflections. Each statement
     must be fully self-contained (use
    names, not pronouns).

Return JSON with "new_reflections": list
     of statement strings.
```

Prompt 8: Contrastive re-extraction prompt (sleep-time).

```
System:
You answer questions from conversation
    memory. Use the EXACT specific
    detail from the most relevant bullet
         prefer the precise name, quote,
     or phrase over a vague
    generalization. Each memory bullet
    has a [timestamp]. Use it to resolve
     relative time expressions: 'last
    year' means the year before the
    timestamp, 'next month' means the
    month after, etc. Respond in JSON.

User:
Question: {question}
User context [{name}]: {persona_summary}

Retrieved memory (with timestamps):
{content_block}

Instructions:
1. If a bullet contains a relative time
    expression (last week, two weeks ago
    , next month, etc.), resolve it to a
     concrete date/period using the [
    timestamp] on that bullet.
2. When the question specifies a
    particular time period, event, or
    context, prioritize bullets whose
    timestamps match that period. Do not
     use information from unrelated
    sessions.
3. Reason across the relevant bullets
         the answer may require combining
     information from multiple bullets.
    If the question asks for a list of
    things, activities, or people,
    enumerate ALL instances found across
     the bullets, not just the first one
    .
4. Answer as concisely as possible using
     the exact specific detail. Be brief
    , but never sacrifice the precise
    fact for brevity (e.g. prefer "
    guinea pig" over "a pet", "like
```

```
    being in a fairy tale" over "magical
     feeling").
5. Always provide your best answer based
     on the available evidence. Only say
     "unknown" if there is genuinely
    zero relevant signal across all
    bullets.
Return JSON with "answer": your short
    answer string.
```

Prompt 9: Answer generation prompt from Reflection statements.

```
System:
You answer questions from conversation
    memory. Use the EXACT specific
    detail from the most relevant
    passage     prefer the precise name,
     quote, or phrase over a vague
    generalization. Each passage has a [
    timestamp]. Use it to resolve
    relative time expressions. Respond
    in JSON.

User:
Question: {question}
User context [{name}]: {persona_summary}

Conversation passages (with timestamps):
{passages_block}

Instructions:
1. Resolve any relative time expressions
     using the [timestamp] of the
    passage they appear in.
2. When the question specifies a
    particular time period or event, use
     ONLY passages from sessions
    matching that period.
3. Answer as concisely as possible using
     the exact specific detail. Be brief
    , but never sacrifice the precise
    fact for brevity. If the question
    asks for a list of things,
    activities, or people, enumerate ALL
     instances found across the passages
    , not just the first one.
4. Only say "unknown" if there is
    genuinely zero relevant signal
    across all passages.
Return JSON with "answer": your short
    answer string.
```

Prompt 10: Answer generation prompt from raw passages (trajectory fallback).

```
System:
You are a fair evaluator. Label answers
    as CORRECT or WRONG and respond in
    JSON.

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

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

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

Now it's time for the real question:
Question: {question}
Gold answer: {gold_answer}
Generated answer: {generated_answer}

First, provide a short (one sentence)
    explanation of your reasoning, then
    finish with CORRECT or WRONG.
Do NOT include both CORRECT and WRONG in
     your response, or it will break the
     evaluation script.

Just return the label CORRECT or WRONG
    in a json format with the key as "
    label".
```

Prompt 11: LLM-as-judge prompt (adapted from Fang et al. (2025)).

```
System:
You are a careful and fair evaluator.
    Respond in JSON.

User:
You are evaluating a memory retrieval
    system. The system ingested a user's
     conversation history with their AI
    companion and generated an answer to
     a probing question about that
    history.

Below is the user's COMPLETE
    conversation history (10 days).
```

```
{history}

---
Question: {question}
Answer generated by the memory system: {
    generated_answer}
---

Instructions:
1. Find the part of the conversation
    history most relevant to the
    question.
   Quote the single most relevant
       exchange (one User line + one AI
       line is enough).
2. Decide whether the generated answer
    is CORRECT or WRONG based solely on
    the
   conversation history.
   - CORRECT: the answer accurately
       reflects what was discussed, even
        if phrased
     differently or more concisely than
         the original.
   - WRONG:   the answer is factually
       incorrect, refers to things not
       mentioned
     in the history, or clearly confuses
          details.
   Be generous with phrasing variation.

Return JSON:
{
  "cited_turn": "direct quote of the
      single most relevant exchange",
  "reasoning": "one sentence explaining
      your judgment",
  "label": "CORRECT" or "WRONG"
}
```

Prompt 12: GVD LLM-as-judge prompt (gpt-5 reads full conversation history).

E Baseline Implementation Details

Naive RAG was implemented from scratch. All other baselines used the authors' provided code as a starting point, with default hyperparameters unless noted below. Where a baseline provides its own answer-synthesis prompt, we use it as-is to remain faithful to each system's original design; otherwise we apply our standard synthesis prompt. All methods were modified to standardize the retrieval embedding to `text-embedding-3-small` and to adapt ingestion and evaluation to both benchmark formats. MemoryBank was adapted from the `local_doc_qa.py` entry point in the authors' repository, retaining its default parameters. MemoryOS was modified to use pre-computed stored embeddings at query time rather than recomputing them on the fly, consistent with the embedding standardization applied to all methods. MAGMA required additional adaptation: we implemented the OpenAI embedding module, corrected issues in the serialization pipeline, and modified retrieval to reuse pre-computed node embeddings. For GVD, we authored a script converting each user's conversation history to MAGMA's input format, treating all questions as single-hop queries (`top_k=15`) and judging with `gpt-5` to match the other GVD baselines.

F Licenses for Artifacts Used

The LoCoMo benchmark (Maharana et al., 2024) is released under the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) license. The GVD benchmark and MemoryBank codebase (Zhong et al., 2024), MAGMA (Jiang et al., 2026), A-Mem (Xu et al., 2025), and LightMem (Fang et al., 2025) are released under the MIT License. MemoryOS (Kang et al., 2025) is released under the Apache 2.0 License. Naive RAG was implemented from scratch and involves no third-party license. OpenAI models (`gpt-4o-mini`, `gpt-5`, `text-embedding-3-small`) are accessed via the OpenAI API under OpenAI's terms of use. Our use of all artifacts is for non-commercial academic research, consistent with their intended use and license terms.

G Generative AI Use

We used Claude[1] to assist in generating code for routine implementation tasks such as data processing functions and formatting conversions. The core algorithmic approaches, system architecture, and experimental methodology were conceived and developed by the authors. For writing this paper, generative AI use was limited to minor grammatical adjustments.

[1] https://claude.ai