MemBuilder: Reinforcing LLMs for Long-Term Memory Construction via Attributed Dense Rewards

Anonymous ACL submission

Abstract

Maintaining consistency in long-term dialogues remains a fundamental challenge for LLMs, as standard retrieval mechanisms often fail to capture the temporal evolution of historical states. While memory-augmented frameworks offer a structured alternative, current systems rely on static prompting of closed-source models or suffer from ineffective training paradigms with sparse rewards. We introduce MemBuilder, a reinforcement learning framework that trains models to orchestrate multi-dimensional memory construction with attributed dense rewards. MemBuilder addresses two key challenges: (1) Sparse Trajectory-Level Rewards: we employ synthetic session-level question generation to provide dense intermediate rewards across extended trajectories; and (2) Multi-Dimensional Memory Attribution: we introduce contribution-aware gradient weighting that scales policy updates based on each component's downstream impact. Experimental results show that MemBuilder enables a 4B-parameter model to outperform state-of-the-art closed-source baselines, exhibiting strong generalization across long-term dialogue benchmarks.

1 Introduction

Memory-augmented frameworks have emerged as a promising approach for maintaining consistency in long-term dialogues, which must track evolving contexts and historical states over extended timelines. While Retrieval-Augmented Generation (RAG) facilitates access to external knowledge, it treats retrieval units as independent, static chunks—failing to capture how information evolves or which historical facts have been superseded (Liu et al., 2024; Gao et al., 2024). Memory-augmented frameworks address this by decomposing information prior to storage: events receive independent timestamps and semantic concepts are structured into discrete units. This shifts the computational burden from processing entangled tokens during inference to retrieving precise, “pre-digested" fragments. Recent implementations like Mem0 (Chhikara et al., 2025), MIRIX (Wang and Chen, 2025), and MemGPT (Packer et al., 2023) exemplify this approach, constructing external memory that evolves with each interaction. However, these systems rely largely on fixed prompting templates and expensive closed-source models, operating in an “open loop" without feedback on whether the constructed memories actually benefit downstream tasks. This raises a critical question: **Can we instead train a model to perform memory construction through direct supervision?**

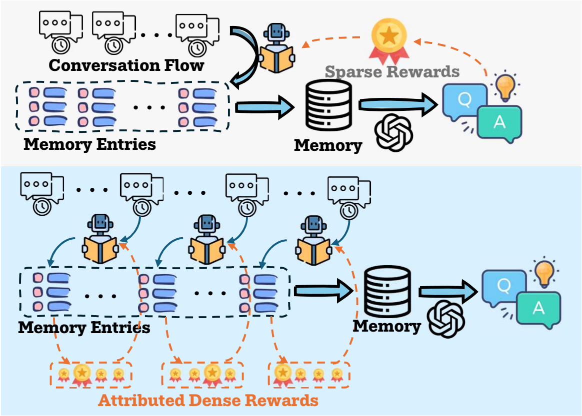

Figure 1: Sparse trajectory-level rewards (top) vs. our attributed dense session-level rewards (bottom). Dense rewards provide learning signals at each session rather than only at trajectory end.

Current training-based approaches, such as Memory-R1 (Yan et al., 2025) and Mem-α (Wang et al., 2025), attempt to address this learning gap but face two critical technical bottlenecks: 1) Sparse Trajectory-Level Rewards: In long-term dialogues, a single reward given at the end of a multi-session trajectory is too sparse. The model cannot discern which specific session's memory operations contributed to the final outcome, making gradient updates noisy and learning ineffective. 2) Multi-Dimensional Memory Attribution: While the previous method adopts a multi-dimensional memory, all components share a global reward, failing to distinguish between operations on different types of memories, regardless of actual downstream impact.

We present MemBuilder, a framework for training models to construct long-term memory with attributed dense rewards. Our architecture utilizes a multi-dimensional memory design, comprising Core, Episodic, Semantic and Procedural components, and trains a single, lightweight 4B model to manage all four components. Our approach introduces two key technical contributions to address the limitations of existing RL-based memory construction methods (Figure 1). First, we employ **dense session-level rewards**. Unlike traditional methods that assign a single reward after processing all sessions, we leverage synthetic session-level question generation to provide immediate feedback after each session's memory operations. Second, we introduce **contribution-aware gradient weighting** to resolve multi-dimensional memory attribution. Since all four memory components share a common reward, their individual contributions remain ambiguous. Our mechanism addresses this by scaling gradient updates based on the downstream utility of the constructed memories; specifically, the gradient impact of a memory operation is proportional to the usage of its corresponding memory component during retrieval. We integrate these techniques into **Attributed Dense Rewards Policy Optimization (ADRPO)**. Extensive experiments conducted on different benchmarks demonstrate the effectiveness of our proposed framework. Specifically, MemBuilder achieves 84.23% on LoCoMo, surpassing the baselines including Claude 4.5 Sonnet under the same setting.

2 Related Works

2.1 Long-Term Memory Management in Conversational Agents

Maintaining coherent and personalized interactions over extended dialogues remains a fundamental challenge for LLM-based agents (Zhong et al., 2024; Packer et al., 2023). Recent benchmarks such as LoCoMo (Maharana et al., 2024), LongMemEval (Wu et al., 2025), and PerLTQA (Du et al., 2024) evaluate long-term memory through multi-session QA, temporal reasoning, and evolving user profiles. Early approaches addressed context limitations through position encoding modifications (Chen et al., 2023; Peng et al., 2024) or dedicated long-context training (Tworkowski et al., 2023; Bai et al., 2024), but these incur high computational costs and struggle to capture the temporal dynamics in multi-session dialogues (Liu et al., 2024). While RAG offers better scalability (Gao et al., 2024), its chunk-level retrieval lacks the temporal and semantic organization needed for complex long-term reasoning (Liu et al., 2024), motivating memory-augmented solutions that decompose information into structured units.

2.2 Prompting-Based Memory Frameworks

Inspired by the cognitive science distinctions among episodic, semantic, and procedural memory (Tulving, 1972), recent frameworks construct structured external memory for LLM agents (Sumers et al., 2024; Laird, 2012). Representative prompting-based implementations include MemGPT with its operating system-like memory hierarchy (Packer et al., 2023), Mem0 for personalized memory extraction (Chhikara et al., 2025), and MIRIX for multi-dimensional organization (Wang and Chen, 2025), and SCM for self-controlled memory management (Wang et al., 2026). Recent work like MMS (Zhang et al., 2025b) and RMM (Tan et al., 2025) further incorporates cognitive principles into memory design. However, these prompting-based frameworks rely on expensive closed-source models and operate without feedback on downstream utility.

2.3 Learning-Based Approaches for Memory Construction

Training-based methods can be categorized by memory form. Latent memory approaches encode information into compact hidden states: MEM1 consolidates memory through internal state mechanisms via RL (Zhou et al., 2025), while MemGen generates latent memory tokens within the reasoning stream (Zhang et al., 2025c), and LongMem (Wang et al., 2023) uses a decoupled architecture with a frozen backbone as memory encoder and an adaptive side-network as memory retriever. Though efficient, these implicit representations sacrifice interpretability and fine-grained controllability.

Explicit memory approaches train models to manage structured external stores. Memory-R1 employs RL with sparse trajectory-level rewards (Yan et al., 2025), but lacks learning signals for dense memory operations. Mem-α trains multi-dimensional memory construction, generalizing from 30k to 400k+ tokens (Wang et al., 2025), yet applies a global reward across all memory operations regardless of their downstream impact. While RLVR with GRPO (Shao et al., 2024) has become widely adopted (Zhang et al., 2025a), sparse rewards remain insufficient for long-term dialogues, motivating our ADRPO.

3 Methodology

3.1 Problem Formulation

We address the task of long-term dialogue question answering. Given a sequence of conversation sessions $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ with associated timestamps $\{t_1, t_2, \ldots, t_n\}$, and a question q posed at time t_q where $t_q > t_n$, the goal is to generate an accurate answer based on information distributed across the entire conversation history. Since concatenating all sessions typically exceeds context limits, we introduce an external memory bank $\mathcal{M}$ that compresses and organizes historical information for selective retrieval at inference time.

3.2 Multi-Dimensional Memory Architecture

To effectively manage long-term dialogue information, we design a multi-dimensional memory system that decomposes conversations into four specialized memory types, each handled by a role-specific prompt to the same LLM (Figure 2).

Memory Structure. Our memory bank $\mathcal{M}$ consists of four components:

- **Core Memory** $\mathcal{M}^{\text{core}}$: A fixed-size block storing persistent user profile information including identity, preferences, and key relationships. This memory is always included in the context during question answering. Detailed prompt templates are provided in Appendix H.

- **Episodic Memory** $\mathcal{M}^{\text{epi}}$: Time-stamped event records capturing what happened and when. Each entry follows the format “`YYYY-MM-DD: Event summary | Details`”, enabling temporal reasoning.

- **Semantic Memory** $\mathcal{M}^{\text{sem}}$: Factual knowledge about entities in the user’s life, such as people, places, and user-specific concepts. Common knowledge is explicitly excluded to avoid redundancy.

- **Procedural Memory** $\mathcal{M}^{\text{proc}}$: Step-by-step processes, routines, and workflows mentioned in conversations, such as the user’s morning routine or problem-solving approach.

Given a new conversation session, "all four memory types are processed simultaneously, each extracting memories according to its specialized perspective. Core Memory is maintained as a fixed block with automatic compression when capacity is exceeded. The other three memory types are stored in a vector database and retrieved via semantic similarity during question answering.

Memory Operations. Since Core Memory operates on a single text block while the other three manage independent entries (Section 3.2), their action spaces differ accordingly:

$$\mathcal{A}^{\text{core}} = \{\textsc{Append}, \textsc{Replace}, \textsc{Rewrite}\} \quad (1)$$

$$\mathcal{A}^{\text{epi}} = \{\textsc{Add}, \textsc{Update}, \textsc{Merge}\} \quad (2)$$

$$\mathcal{A}^{\text{sem}} = \{\textsc{Add}, \textsc{Update}, \textsc{Skip}\} \quad (3)$$

$$\mathcal{A}^{\text{proc}} = \{\textsc{Add}, \textsc{Update}\} \quad (4)$$

At session τ, given state $State_\tau = (\mathcal{M}_{\tau-1}, s_\tau)$, the LLM selects an action $a \in \mathcal{A}^{(m)}$ for each memory type and generates the corresponding memory content.

For Core Memory, APPEND adds new information to the block, REPLACE updates specific fragments, and REWRITE reorganizes the entire block. For the other three types, ADD creates a new entry, and SKIP bypasses common knowledge already captured in the model’s parameters.

Unlike prior memory systems that delete old entries and replace them with new ones (Chhikara et al., 2025; Wang and Chen, 2025), we introduce two operations that preserve temporal history. The **UPDATE** operation creates a new entry with a fresh timestamp that explicitly references the previous entry, rather than overwriting it, enabling the model to trace how information evolved. The **MERGE** operation synthesizes multiple related events into a conclusion spanning a time range while preserving references to the original events as evidence, precomputing complex temporal reasoning to reduce the burden during question answering.

This architecture transforms unstructured dialogues into organized, queryable memory. The remaining challenge is how to train the LLM to construct memory that maximizes downstream QA performance.

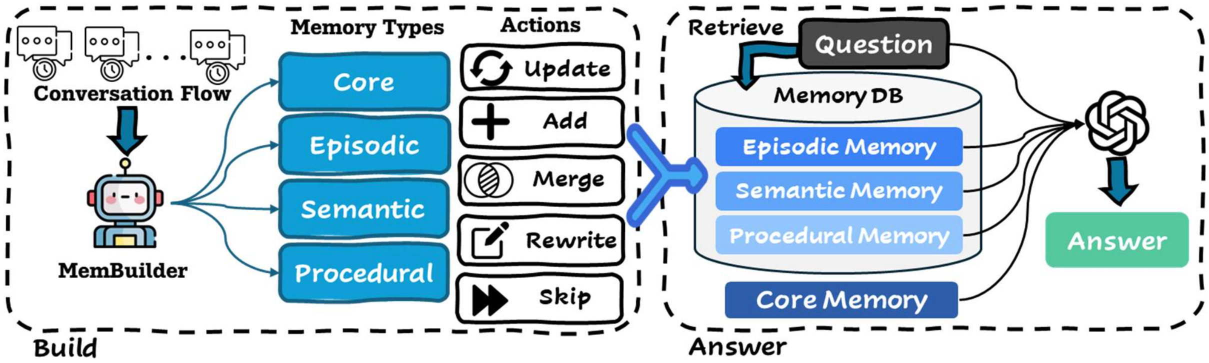

Figure 2: Multi-Dimensional Memory Architecture. Four memory types (Core, Episodic, Semantic, Procedural) are constructed during the Build Phase and selectively retrieved during the Answer Phase.

3.3 Supervised Fine-Tuning

While multi-dimensional memory architectures can offer finer-grained control (Wang and Chen, 2025), lightweight models, such as Qwen3-4B employed in our framework, often struggle with direct multi-dimensional memory construction, frequently producing invalid actions that impede effective RL exploration. To address this cold-start problem, we collect expert trajectories $\{(State_t, a_t)\}_{t=1}^{n}$ using Claude 4.5 Sonnet. This phase stabilizes the model's output format, providing a viable baseline for subsequent training. However, SFT primarily focuses on behavioral cloning with limited exploration capability; we further employ RL to maximize the model's ultimate utility in QA contexts.

3.4 Attributed Dense Rewards Policy Optimization (ADRPO)

While SFT enables valid action generation, the resulting policy lacks optimization for downstream QA utility. We introduce a reinforcement learning algorithm that addresses two key challenges: **sparse trajectory-level rewards** in long-term dialogues, and **multi-dimensional memory attribution** among memory components with varying downstream impacts. Figure 3 illustrates the ADRPO training pipeline.

3.4.1 Dense Session-Level Rewards via Synthetic Session-level QA

Prior RL approaches for memory construction (Yan et al., 2025; Wang et al., 2025) assign a single reward at the trajectory's end based on the final QA result. For dialogues spanning dozens of sessions, this provides no learning signal for dense memory operations.

We address this through synthetic session-level QA that evaluates memory quality at each step. Before the RL training, for each session τ, we retrieve the top-k memories from $\mathcal{M}_{\tau-1}$ most similar to the session s_τ, and let an expert model with $(s_\tau, \mathcal{M}_{\text{retrieved}})$ generate J question-answer pairs $\{(q_j, ans_j)\}_{j=1}^{J}$ targeting information in s_τ or its connections to retrieved memoey. Questions span three types: single-session (testing current session retention), multi-session (requiring cross-session aggregation), and temporal-reasoning (involving time-based inference).

During the RL training at s_τ, we sample N rollouts. Each rollout i produces memory operations for all four memory types, yielding a candidate memory bank $\mathcal{M}_\tau^{(i)}$. A capable model answers pre-generated questions by retrieving from $\mathcal{M}_\tau^{(i)}$, and an LLM judge assesses correctness against ground-truth ans_j. The task reward measures memory construction quality as the average QA accuracy:

$$r^{\text{task}} = \frac{1}{J}\sum_{j=1}^{J} \mathbb{1}[\text{correct}(q_j)] \tag{5}$$

The final reward incorporates two regularization terms:

$$r = \mathbb{1}[\text{valid}] \cdot r^{\text{task}} \cdot (1 - \lambda \cdot \ell) \tag{6}$$

Format Validity. The indicator $\mathbb{1}[\text{valid}]$ acts as a gate: outputs with malformed JSON structure, missing required fields, or undefined actions receive zero reward regardless of content quality.

Length Penalty. The term $\ell \in [0, 1]$, weighted by λ, regularizes the amount of memory content stored. Let $|\mathcal{M}_{\text{new}}^{(m)}|$ and $|\hat{\mathcal{M}}^{(m)}|$ denote the token counts of memories stored by the policy and expert for memory type m, respectively. For core Memory, let $\Delta_{\text{core}} = |\mathcal{M}_\tau^{(\text{core})}| - |\mathcal{M}_{\tau-1}^{(\text{core})}|$ be the token

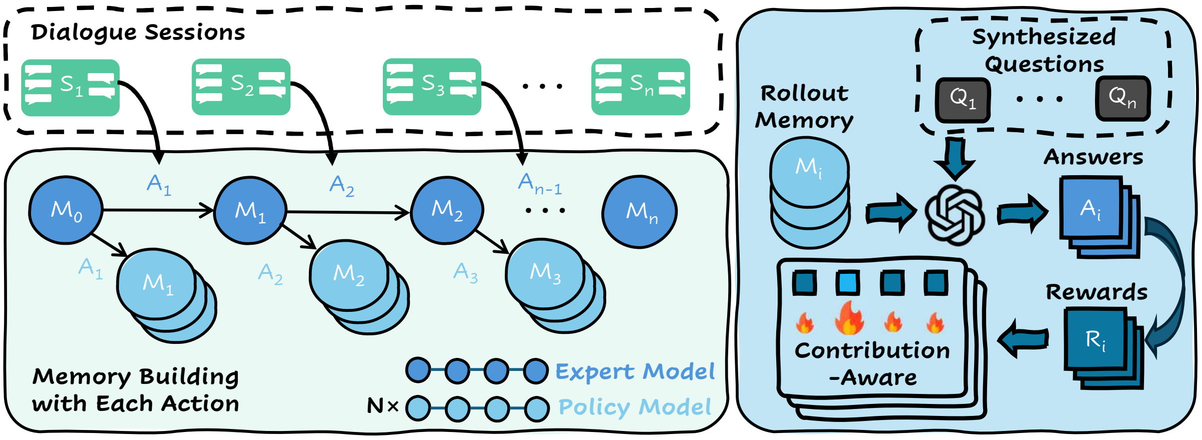

Figure 3: ADRPO training pipeline. Each session's memory rollouts are evaluated via synthetic QA, with gradients weighted by each memory component's downstream contribution.

increment after the operation:

$$\ell^{(\text{core})} = \begin{cases} 0 & \text{if } \Delta_{\text{core}} \le \theta_{\min} \\ \frac{\Delta_{\text{core}}-\theta_{\min}}{\theta_{\max}-\theta_{\min}} & \text{if } \Delta_{\text{core}} \in (\theta_{\min}, \theta_{\max}) \\ 1 & \text{if } \Delta_{\text{core}} \ge \theta_{\max} \end{cases} \tag{7}$$

where $\theta_{\min}$ and $\theta_{\max}$ are the penalty-free and full-penalty thresholds. For the other memory types, let $\rho = |\mathcal{M}_{\text{new}}^{(m)}|/|\hat{\mathcal{M}}^{(m)}|$ and $\Delta = \big||\mathcal{M}_{\text{new}}^{(m)}| - |\hat{\mathcal{M}}^{(m)}|\big|$:

$$\ell^{(m)} = \begin{cases} 0 & \text{if } \Delta < \delta \text{ or } \rho \in [\gamma_l, \gamma_u] \\ \frac{\rho-\gamma_u}{\gamma_{\max}-\gamma_u} & \text{if } \rho \in (\gamma_u, \gamma_{\max}] \\ \frac{\gamma_l-\rho}{\gamma_l-\gamma_{\min}} & \text{if } \rho \in [\gamma_{\min}, \gamma_l) \\ 1 & \text{otherwise} \end{cases} \tag{8}$$

where δ is the minimum difference threshold, $[\gamma_l, \gamma_u]$ the tolerance range, and $\gamma_{\min}$, $\gamma_{\max}$ the full-penalty boundaries.

3.4.2 Contribution-Aware Gradient Weighting

Within each rollout, actions operating on the four memory dimensions contribute to a shared memory bank and receive a global reward. However, the functional impact on downstream QA performance varies significantly across memory types; for instance, Episodic Memories may be frequently retrieved while Procedural Memories remain unused. To account for these discrepancies, we dynamically amplify gradient updates based on each component's downstream utility.

During QA evaluation, we record retrieval counts $h^{(m)}$ for each memory type $m \in \{\text{epi}, \text{sem}, \text{proc}\}$ across all questions. The dominant contributing type is:

$$d = \arg \max_{m \in \{\text{epi},\text{sem},\text{proc}\}} h^{(m)} \tag{9}$$

Gradient weights are assigned as:

$$w^{(m)} = \begin{cases} \alpha & \text{if } m = d \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

where $\alpha > 1$ amplifies updates for the dominant contributor. Core Memory, which is always included in the context rather than retrieved, receives a fixed weight $w^{(\text{core})} = 1$. This mechanism ensures that memory types whose entries directly contributed to successful QA receive proportionally stronger reinforcement.

3.4.3 Training Objective

We formulate the ADRPO training objective by extending GRPO (Shao et al., 2024) with the attributed session-level reward. At each session τ, we sample N rollouts from the current policy. Each rollout i invokes the model four times in parallel, producing memory operations $a_i = (a_i^{\text{core}}, a_i^{\text{epi}}, a_i^{\text{sem}}, a_i^{\text{proc}})$ for each memory type. All four memory types share the session-level reward r_i defined in Eq. 6, but receive differentiated gradient weights $w^{(m)}$ based on their retrieval-based attribution.

Advantages are computed via within-group normalization:

$$A_i = \frac{r_i - \mu}{\sigma + \epsilon} = \frac{\mathbb{1}[\text{valid}_i] \cdot r_i^{\text{task}} \cdot (1 - \lambda \ell_i) - \mu}{\sigma + \epsilon} \tag{11}$$

where μ and σ are computed over the N rollouts, and ϵ is a small constant for numerical stability. The training objective is:

$$\mathcal{J}(\theta) = \mathbb{E}\left[\sum_{m} \frac{1}{|a_i^{(m)}|} \sum_{k=1}^{|a_i^{(m)}|} \min\left(w^{(m)} \rho_{i,k}^{(m)} A_i, \mathrm{clip}(\rho_{i,k}^{(m)}, 1-\epsilon, 1+\epsilon) A_i\right)\right] - \beta \cdot D_{\mathrm{KL}}(\pi_\theta \| \pi_{\mathrm{ref}}) \tag{12}$$

where $\rho_{i,k}^{(m)} = \pi_\theta / \pi_{\mathrm{ref}}$ is the importance ratio for the k-th token of memory type m's output. The contribution-aware weights $w^{(m)}$ scale the unclipped term, encouraging larger updates for high-impact components while preserving the clipping mechanism for stability.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate MemBuilder on three long-term dialogue benchmarks: **LongMemEval** (Wu et al., 2025), which consists of user-assistant chat histories designed to evaluate the long-term memory capabilities of chat assistants; **LoCoMo** (Maharana et al., 2024), which contains human-human conversations between fictional personas grounded on temporal event graphs, spanning up to 35 sessions; and **PerLTQA** (Du et al., 2024), a dataset featuring 141 characters with rich personal profiles, social relationships, and life events. We train exclusively on the LongMemEval subset. LoCoMo and PerLTQA serve as OOD test sets that differ in both dialogue format and domain. Detailed statistics and data construction procedures are provided in Appendix A.

Baselines. We compare against: (1) **RAG-based**: Following common practice in RAG-based systems (Mastra, 2025), we implement two retrieval granularities: RAG-Session chunks dialogues at session boundaries and retrieves complete sessions, while RAG-Utterance embeds individual utterances for fine-grained matching but returns the containing session to preserve conversational context; (2) **Memory frameworks**: Mem0 (Chhikara et al., 2025) and MIRIX (Wang and Chen, 2025); (3) **Training-based**: Memory-R1 (Yan et al., 2025), whose results are taken from the original paper due to unavailable code.

Implementation Details. We use Qwen3-4B-Instruct-2507 as our base model, with SFT trajectories collected using Claude 4.5 Sonnet. GPT-4.1 serves as the LLM judge for evaluation. All retrieval uses text-embedding-3-small. To isolate memory construction quality, we fix the answer model to Claude 4.5 Sonnet unless otherwise specified. Full details are in Appendix B and D. Detailed configuration including embedding settings and action formats is in Appendix C. A detailed cost breakdown is provided in Appendix F.

4.2 Main Results

Table 1 presents performance across three benchmarks. To isolate the effect of memory construction quality, we set the answer model to Claude 4.5 Sonnet across all methods and compare three categories of approaches: retrieval-based methods, prompting-based frameworks, and training-based methods. Note that Memory-R1 trains Llama-3.1-8B-Instruct as both the memory construction and answer model. For a fairer comparison, we also evaluate our method with Qwen3-4B as the answer model (Table 2), which still achieves 82.00% on LoCoMo, significantly outperforming Memory-R1 (62.74%).

Method	LoCoMo	LongMemEval	PerLTQA
Retrieval-based Methods			
RAG-Session	70.35	66.75	79.21
RAG-Utterance	74.87	69.00	77.23
Prompting-based Memory Construction			
Mem0	51.64	47.00	62.04
MIRIX	77.48	73.25	83.11
Ours (GPT-4.1)	79.91	78.50	91.74
Ours (QwQ-32B)	77.47	76.00	88.96
Ours (Claude 4.5 Sonnet)	82.61	85.50	92.59
Training-based Memory Construction			
Memory-R1[†]	62.74	-	-
Ours (Qwen3-4B)	68.07	56.00	76.85
+ SFT	81.74	84.25	91.67
+ RL	79.31	62.75	82.19
+ SFT + RL	**84.23**	**85.75**	**93.14**

[†] Results from the original paper with a different answer model.

Table 1: Performance comparison of different memory construction methods. "Ours" denotes our memory architecture with different memory construction models.

Our method achieves SOTA performance across all three benchmarks. On LoCoMo, our trained Qwen3-4B model achieves 84.23%, surpassing the best prompting-based framework MIRIX (77.48%) by 6.75 percentage points and outperforming Claude 4.5 Sonnet as the memory construction model (82.61%). Similar trends are observed on LongMemEval (85.75%) and PerLTQA (93.14%), where our method also outperforms all baselines including Claude 4.5 Sonnet. These results demonstrate that, although memory construction requires frequent model invocations across sessions, a well-trained 4B model can effectively replace expensive closed-source APIs. Notably, our model is trained exclusively on LongMemEval, yet achieves strong performance on LoCoMo and PerLTQA, demonstrating robust generalization to OOD benchmarks with different dialogue structures and question types.

The training stage ablation reveals the complementary roles of SFT and RL. SFT alone improves the base model from 68.07% to 81.74% by enabling valid multi-dimensional outputs, while RL further boosts performance to 84.23% by optimizing for downstream QA utility. Notably, RL without SFT (79.31%) underperforms SFT alone, confirming that supervised fine-tuning is essential to address the cold-start problem before effective RL exploration can proceed.

4.3 Ablation Studies

We conduct ablation experiments to analyze the contribution of our key design choices. Implementation details are provided in Appendix E.

4.3.1 Effect of Gradient Weighting

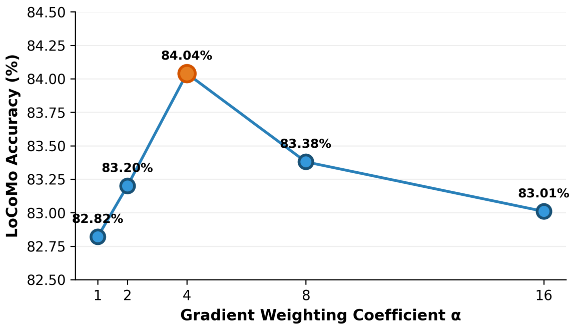

Figure 4: Training curves with different gradient weighting coefficients $\alpha \in \{1, 2, 4, 8, 16\}$ on LoCoMo.

To investigate the effect of contribution-aware gradient weighting (Section 3.4.2), we vary the weighting coefficient α that amplifies updates for the dominant contributing memory type. We conduct this ablation on a reduced training set for efficiency. As shown in Figure 4, performance improves as α increases from 1 (no weighting, 82.82%) to 4 (84.04%), confirming that attributing credit to high-contribution memory types enhances the final model performance. However, excessively large α values degrade performance due to gradient imbalance among memory types, with the optimal value at $\alpha = 4$.

4.3.2 Effect of Dense Rewards

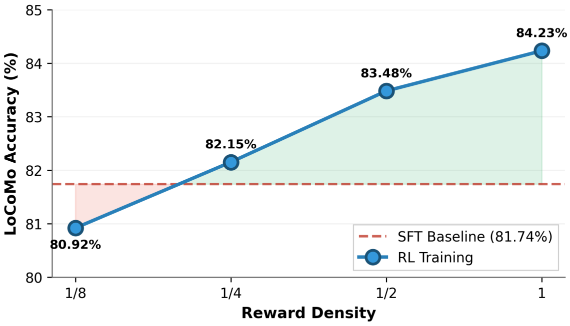

Figure 5: Effect of reward density on LoCoMo accuracy. The x-axis indicates the fraction of sessions receiving task rewards during training.

To validate the importance of dense session-level rewards, we vary the reward density by providing task rewards to only a fraction of sessions during training. As shown in Figure 5, under the same number of training epochs, model performance degrades consistently as the reward becomes sparser. When reward density drops to 1/8, performance falls below the SFT baseline (80.92% vs 81.74%).

These results reveal that sparse rewards not only slow convergence but can also be worse than using SFT alone, explaining why prior sparse-reward approaches achieve limited gains despite employing larger base models.

4.3.3 Answer Model Generalization

Answer Model	LoCoMo	LongMemEval	PerLTQA
Claude 4.5 Sonnet	84.23	85.75	93.14
GPT-4.1	81.51	82.50	91.83
Qwen3-4B Base	74.61	75.00	83.93
Qwen3-4B Ours	81.12	83.00	91.19

Table 2: Performance with different answer models using memory constructed by our trained Qwen3-4B model. “Qwen3-4B Base” denotes the base model, while “Qwen3-4B Ours” denotes our trained model.

To evaluate whether the constructed memory generalizes across different answer models, we fix the memory construction model to our trained Qwen3-4B and vary the answer model. Table 2 shows that our memory maintains high quality across answer models of varying capabilities.

Interestingly, when using our trained Qwen3-4B as an answer model, accuracy improves from

74.61% to 81.12% over the base model and remains competitive with GPT-4.1, suggesting RL training creates implicit alignment between memory structure and the model's reasoning patterns.

4.4 Further Analysis

4.4.1 Performance by Question Type

Method	SingleHop	MultiHop	OpenDomain	Temporal	Adversarial
RAG-Utterance	68.75	51.35	84.29	69.23	85.11
Memory-R1	59.83	53.01	68.78	51.55	-
Ours	82.27	77.88	84.66	71.71	90.58

Table 3: Performance breakdown by question type on LoCoMo.

Table 3 details performance across LoCoMo question categories. Our method achieves the largest gains on MultiHop questions (77.88% vs 53.01% for Memory-R1, +24.87pp) and Temporal questions (71.71% vs 51.55%, +20.16pp), both of which require synthesizing information across multiple sessions. On Adversarial questions, our method achieves 90.58%, demonstrating robustness against misleading information.

4.4.2 Action Distribution Analysis

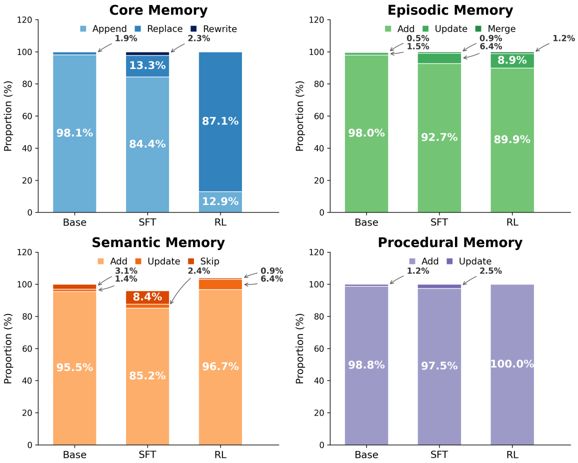

Figure 6: Action distribution across training stages (Base, SFT, RL) for four memory types.

Figure 6 visualizes how action distributions evolve across training stages. Concrete examples illustrating these behavioral changes are provided in Appendix G.

The most notable change is Core Memory's shift from APPEND (98.1% → 12.9%) to REPLACE (13.3% → 87.1%). Our analysis of the generated outputs demonstrates that the model learns to perform targeted updates to specific fields rather than appending at the end. The RL training also teaches the model to be more selective at generation time rather than relying on post-hoc filtering. For Semantic Memory, SKIP operations decrease, indicating that the model directly outputs relevant facts rather than enumerating candidates and then excluding. We also observe divergent UPDATE behavior: usage increases for Episodic and Semantic Memories but drops to zero for Procedural Memory, suggesting that evolving information (e.g., events and facts) benefits from explicit update chains while procedural knowledge is better maintained by adding discrete new entries.

4.4.3 Training Dynamics

Figure 7: Training dynamics: (a) overall reward trend, (b) rewards by memory type, and (c) response length by memory type. All metrics show stable improvement without reward hacking.

Figure 7 illustrates the training dynamics of our RL process. (a) Both task reward (QA accuracy) and mean reward (with length penalty) improve steadily, indicating effective learning. (b) All four memory types show consistent reward growth. (c) Response lengths remain stable throughout training, confirming that the length penalty prevents reward hacking through verbose outputs.

5 Conclusion

We presented MemBuilder, a reinforcement learning framework for multi-dimensional memory construction in long-term dialogues. By introducing ADRPO, Qwen3-4B achieves 84.23% on LoCoMo, surpassing prompting-based frameworks using expensive closed-source models and generalizing effectively to OOD benchmarks. Our results demonstrate that memory construction can be handled by lightweight open-source models with appropriate training.

Limitations

To isolate the impact of memory construction quality and ensure fair comparison across different methods, our evaluation relies on a fixed closed-source model (Claude 4.5 Sonnet) for question answering, which incurs API costs during evaluation. However, as shown in Table 2, our constructed memory generalizes well across different answer models, suggesting that practitioners can substitute with capable open-source alternatives for cost-sensitive deployments. Furthermore, although we employ Claude 4.5 Opus for synthetic question generation, the generated QA pairs may still contain occasional inaccuracies or ambiguities. Despite this, our experimental results demonstrate that training with these synthetic questions substantially improves performance over sparse reward baselines.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. 2024. Longalign: A recipe for long context alignment of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1376–1395.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. 2023. Longlora: Efficient fine-tuning of long-context large language models. *CoRR*, abs/2309.12307.

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

Yiming Du, Hongru Wang, Zhengyi Zhao, Bin Liang, Baojun Wang, Wanjun Zhong, Zezhong Wang, and Kam-Fai Wong. 2024. PerLTQA: A personal long-term memory dataset for memory classification, retrieval, and fusion in question answering. In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pages 152–164, Bangkok, Thailand. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey. *Preprint*, arXiv:2312.10997.

John E. Laird. 2012. *The Soar Cognitive Architecture*. MIT Press.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of LLM agents. 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.

Mastra. 2025. Yes, you can use rag for agent memory. `https://mastra.ai/blog/use-rag-for-agent-memory`. Accessed: 2025.

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.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. 2024. Yarn: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024*.

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025, Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025*, pages 1279–1297. ACM.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. 2024. Cognitive architectures for language agents. *Transactions on Machine Learning Research*.

Zhen Tan, Tianshu Shen, I-Hung Hsu, Anbang Xu, Longju Bai, Yash Jain, Chen-Yu Lee, Hamid Palangi, Tianlong Chen, Long T. Le, and Rujun Han. 2025. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pages 8416–8439.

Endel Tulving. 1972. Episodic and semantic memory. In Endel Tulving and Wayne Donaldson, editors, *Organization of Memory*, pages 381–403. Academic Press.

Szymon Tworkowski, Konrad Staniszewski, Mikolaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr

Milos. 2023. Focused transformer: Contrastive training for context scaling. In *Advances in Neural Information Processing Systems 36: NeurIPS 2023*.

Bing Wang, Xinnian Liang, Jian Yang, Hui Huang, Zhenhe Wu, ShuangZhi Wu, Zejun Ma, and Zhoujun Li. 2026. Scm: Enhancing large language model with self-controlled memory framework. In *Database Systems for Advanced Applications*, pages 188–203, Singapore. Springer Nature Singapore.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. 2023. Augmenting language models with long-term memory. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Yu Wang and Xi Chen. 2025. MIRIX: multi-agent memory system for llm-based agents. *CoRR*, abs/2507.07957.

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

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

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

Chong Zhang, Yue Deng, Xiang Lin, Bin Wang, Dianwen Ng, Hai Ye, Xingxuan Li, Yao Xiao, Zhanfeng Mo, Qi Zhang, and Lidong Bing. 2025a. 100 days after deepseek-r1: A survey on replication studies and more directions for reasoning language models. *Preprint*, arXiv:2505.00551.

Gaoke Zhang, Bo Wang, Yunlong Ma, Dongming Zhao, and Zifei Yu. 2025b. Multiple memory systems for enhancing the long-term memory of agent. *CoRR*, abs/2508.15294.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. 2025c. Memgen: Weaving generative latent memory for self-evolving agents. *CoRR*, abs/2509.24704.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. 2024. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410, Bangkok, Thailand. Association for Computational Linguistics.

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.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. MEM1: learning to synergize memory and reasoning for efficient long-horizon agents. *CoRR*, abs/2506.15841.

A Dataset Details

A.1 Benchmark Datasets

We evaluate on three long-term dialogue benchmarks that differ in dialogue format, domain, and data organization.

LongMemEval consists of user-assistant chat histories designed to evaluate the long-term memory capabilities of chat assistants. The dataset contains 500 independent questions, each with its own dialogue context, averaging 40 sessions and approximately 115K tokens. Questions test five core memory abilities: information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention. This benchmark serves as both our training source and in-distribution evaluation set.

LoCoMo contains human-human conversations between fictional personas, grounded on temporal event graphs. The dataset includes 10 dialogues with an average of 27 sessions (range: 19–32) and 14K tokens per dialogue. Each dialogue is associated with multiple questions, totaling 1,986 questions across five types: SingleHop (282), MultiHop (321), OpenDomain (841), Temporal (96), and Adversarial (446). Unlike LongMemEval's one-question-per-context format, LoCoMo tests memory systems on shared dialogue contexts with diverse question types.

PerLTQA is a dataset featuring 141 fictional characters, including 30 protagonists with rich personal profiles, social relationships, and life events. Questions are designed around the 30 protagonists, totaling 8,593 questions across five types: factual, reasoning, other, yes/no, and temporal. This dataset requires integrating Episodic Memories with Semantic Memories about characters.

Both LoCoMo and PerLTQA serve as out-of-distribution test sets, differing from LongMemEval in dialogue format (human-human vs. user-assistant), data organization (shared context vs. independent context), and domain.

A.2 Training Data Construction

We use LongMemEval as our sole training source. From the 500 available dialogues, we sample 50 dialogues for SFT trajectory collection and a separate 50 dialogues for RL dataset construction. All other benchmarks (LoCoMo and PerLTQA) serve as out-of-distribution test sets to evaluate generalization.

SFT Dataset. The 50 SFT dialogues comprise approximately 2,400 sessions. Since each session requires memory operations from all four memory types (Core, Episodic, Semantic, Procedural) and we train each memory type's output as a separate example, this yields 9,600 training samples. Each sample pairs an input (retrieved memories + current session) with a single type of memory operations in JSON format. Based on logged data, the average input length is approximately 6,000 tokens and the average output length is approximately 780 tokens, resulting in a total of approximately 65M tokens for SFT training.

RL Dataset. The 50 RL dialogues comprise approximately 2,400 sessions. For each session, we generate 5 synthetic QA pairs for dense reward computation, resulting in 12,000 QA pairs in total. During GRPO training with 8 rollouts per session and 5 epochs, this produces 96,000 session-rollouts for policy optimization.

B Training Pipeline Details

B.1 Expert Trajectory Collection

We collect expert memory construction trajectories using Claude 4.5 Sonnet. For each dialogue, we process sessions sequentially: at step k, the model receives the new session s_k along with relevant memories retrieved from the step $k-1$ memory bank, then generates memory operations for all four memory types. After each step, we compute embeddings for newly created memories and update the vector database. This produces complete trajectories of memory states $\{\mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_n\}$ for each dialogue.

B.2 RL Dataset Construction

The RL dataset construction involves three components:

Input Preparation. For each session k, we construct the policy model's input by retrieving the top-20 most relevant memories from the step $k-1$ vector database, concatenated with the new session s_k. This mirrors the inference-time setup where the model must decide what to extract given limited context about historical memories.

Synthetic Question Generation. To enable dense session-level rewards, we generate 5 question-answer pairs for each session using Claude 4.5 Opus. The generation model receives the new session s_k along with the top-20 most similar memories retrieved from the step $k-1$ memory bank. This retrieval provides context about historical information, allowing the model to generate questions that test not only the understanding of the new session content but also its connections to prior history when relevant relationships exist. The model is instructed to create questions spanning factual recall, temporal reasoning, and inference tasks.

Reward Computation. During the RL training, each session undergoes 8 rollouts, producing 8 candidate memory banks $\{\mathcal{M}_k^{(1)}, \dots, \mathcal{M}_k^{(8)}\}$. For each candidate, GPT-4.1-mini answers the pre-generated questions by retrieving from the candidate memory bank, and following prior work (Chhikara et al., 2025; Wang and Chen, 2025), and an LLM judge (GPT-4.1-mini) evaluates answer correctness. The average accuracy across 5 questions yields the task reward for each rollout. Note that within each rollout, all four memory types share the same task reward, which is then differentiated through contribution-aware gradient weighting (Section 3.4.2).

B.3 SFT Training

We perform supervised fine-tuning using LlamaFactory (Zheng et al., 2024). The training data consists of expert trajectories where each example pairs a session input (retrieved memories + new session) with the expert model's memory operations for all four memory types. We use Qwen3-4B-Instruct-2507 as the base model with learning rate 5×10^{-7}, batch size 4, and train for 10 epochs.

B.4 RL Training

We implement ADRPO by extending the verl framework (Sheng et al., 2025) with contribution-aware gradient weighting and session-level reward computation. Starting from the SFT checkpoint, we train with a learning rate 1×10^{-6} and a batch size of 128. The rollout number is set to 8, and the clipping parameter ϵ is set to 0.2. For contribution-aware gradient weighting, we use $\alpha = 4$ based on ablation results (Figure 4). The length penalty coefficient λ is set to 0.8, with Core memory thresholds $\theta_{\min} = 150$ and $\theta_{\max} = 400$, and other memory parameters $\delta = 200$ and $[\gamma_l, \gamma_u] = [0.5, 1.3]$. Training runs for 5 epochs on 32 H20 GPUs (4 nodes), taking approximately 70 hours.

C Framework Configuration

C.1 Embedding and Retrieval

We use text-embedding-3-small as the embedding model for all retrieval operations throughout the framework. For memory construction, we retrieve the top-20 most relevant old memories to provide context for the current session. For QA answering, we similarly retrieve the top-10 memories from the final memory bank. The RAG baselines (RAG-Session and RAG-Utterance) retrieve the top-5 chunks. For synthetic QA generation during RL dataset construction, we retrieve the top-20 memories to provide richer historical context.

C.2 Memory Configuration

Core memory is maintained as a single text block with a maximum capacity of 5,000 characters. When the content exceeds this limit after an Append or Replace operation, the same policy model is prompted to compress the content while preserving essential information, including user identity, key relationships, personality traits, important preferences, and long-term goals. The compression removes redundant descriptions, minor details, and verbose explanations. If the first compression attempt still exceeds the limit, a second, more aggressive compression pass is performed.

Episodic, Semantic, and Procedural Memories are stored as individual entries in a vector database with no explicit size limit. Each entry is embedded independently for retrieval.

C.3 Agent Action Format

We refer to the role-specific prompt for each memory type as an *agent*. For each memory type, the model receives the current session along with relevant retrieved memories and outputs a JSON object specifying the action type and content. Below, we describe the action format for each agent.

Core memory Agent. The Core memory Agent manages persistent user information, including identity, preferences, personality traits, and key relationships. It outputs one of three operations:

- **APPEND**: Add new information to the existing Core memory block (used when capacity <90%).
- **REPLACE**: Update specific outdated or incorrect information by specifying old and new text.
- **REWRITE**: Reorganize and consolidate the entire block (used when capacity >90% or when major updates are needed).

Example output:

```
{"operation": "APPEND",
 "content": "Works as a software engineer at
 Google, specializing in machine learning"}
```

Episodic Memory Agent. The Episodic Memory Agent manages time-ordered event memories. Each entry includes a timestamp, summary, and detailed description capturing who, what, when, where, and why. It outputs operations from:

- **ADD**: Create a new event entry not currently in memory.
- **UPDATE**: Add a new related event that references previous events (old versions remain for history).
- **MERGE**: Combine multiple related events into a timeline with a timestamp range, drawing conclusions from patterns (old versions remain for history).

Example output:

```
{"operations": [
  {"action": "ADD",
   "memory": "2024-03-15: Started new job at
   startup | Details: First day at TechCorp as
   senior engineer, met team lead Sarah..."}
]}
```

Semantic Memory Agent. The Semantic Memory Agent manages conceptual knowledge about people, places, objects, and concepts in the user's life. It explicitly skips common knowledge already captured in the model's parameters. Operations include:

- **ADD**: Create an entry for a new concept, person, or object.

- **UPDATE**: Add new information to an existing concept.
- **SKIP**: Bypass common knowledge or already fully captured information.

Example output:

```
{"operations": [
  {"action": "ADD",
   "memory": "Sarah (colleague) - Career: Team
   lead at TechCorp, 5 years experience in ML
   ..."},
  {"action": "SKIP",
   "reason": "Common knowledge about Python"}
]}
```

Procedural Memory Agent. The Procedural Memory Agent manages step-by-step processes, workflows, and instructions. Each entry includes a description, numbered steps, and optional context. Operations include:

- **ADD**: Create a new procedure entry.
- **UPDATE**: Modify an existing procedure with new information.

Example output:

```
{"operations": [
  {"action": "ADD",
   "memory": "Morning workout routine | Steps: 1.
    10min stretching 2. 30min jogging 3. 15min
   core exercises | Context: Daily routine
   before work"}
]}
```

Validity Criteria. An action is considered **valid** if: (1) the JSON structure is well-formed, (2) the action type is defined for that agent, (3) all required fields are present, and (4) for UPDATE/MERGE operations, referenced entries exist in the current memory bank. Invalid actions receive zero reward regardless of content quality.

D Baseline Implementation

D.1 RAG Baselines

RAG-Session segments dialogues at session boundaries, treating each session as a retrieval unit. Given a question, we retrieve the top-5 most similar sessions using text-embedding-3-small and provide them as context to the answer model. **RAG-Utterance** segments dialogues at the utterance level, treating each user-assistant turn pair as a retrieval unit. Given a question, we retrieve the top-5 most similar utterances using text-embedding-3-small and provide them as context to the answer model.

D.2 Memory Frameworks

For fair comparison, we evaluate all memory frameworks using Claude 4.5 Sonnet as both the memory construction model and answer model, except for ablations in Table 2 which vary the answer model.

Mem0 is configured with its default settings for memory extraction and organization.

MIRIX is configured with its default multi-dimensional memory structure.

Both frameworks construct memories by processing dialogues sequentially, then answer questions by retrieving from the constructed memory bank.

D.3 Training-based Methods

Memory-R1 results are taken from the original paper. Note that Memory-R1 uses Llama-3.1-8B-Instruct as both the memory construction and answer model, which differs from our evaluation setup. For reference, our method with Qwen3-4B as both the construction and answer model achieves 82.00% on LoCoMo (Table 2), substantially outperforming Memory-R1's reported 62.74%.

E Ablation Experiment Details

E.1 Gradient Weighting Ablation

To efficiently explore the effect of contribution-aware gradient weighting, we conduct this ablation on a reduced training set consisting of 10 dialogues sampled from the 50 RL training dialogues. We vary $\alpha \in \{1, 2, 4, 8, 16\}$ while keeping all other hyperparameters fixed. The setting $\alpha = 1$ corresponds to uniform weighting without contribution-aware scaling. Results are shown in Figure 4.

E.2 Reward Density Ablation

We conduct this ablation on the full training set of 50 dialogues. To validate the importance of dense session-level rewards, we vary the reward density by randomly skipping task reward computation for a fraction of sessions. Specifically, at density $1/d$, each session independently has a probability $1/d$ of receiving a task reward. Sessions without task rewards still receive format validity and length penalty signals, but no QA-based feedback. All configurations are trained for the same number of epochs to ensure fair comparison. Results are shown in Figure 5.

F Cost Analysis

Our training pipeline incurs API costs at two stages: (1) data preparation (one-time) and (2) RL training (per-run).

F.1 Data Preparation Costs (One-Time)

Expert Trajectory Generation. We use Claude 4.5 Sonnet to generate memory management demonstrations for 50 LongMemEval conversations (2,400 sessions). Each session invokes 4 memory agent calls (Core, Episodic, Semantic, Procedural), resulting in 9,600 API calls. Based on actual logged data from 952 agent calls, the average token consumption is 6,011 input tokens (agent prompt + existing memories + current session) and 784 output tokens (JSON memory operations). Total: $\sim$58M input + $\sim$8M output tokens. Cost: **$294**.

Synthetic Question Generation. For all 50 conversations (2,400 sessions), we generate 5 QA pairs per session using Claude 4.5 Opus. Based on analysis of generated QA files, each call uses $\sim$3,500 input tokens (QA prompt template + memory state + current session) and produces $\sim$460 output tokens (5 questions with answers in JSON). Total: $\sim$8.5M input + $\sim$1.1M output tokens. Cost: **$70**.

F.2 RL Training Costs (Per-Run)

During GRPO training (2,400 sessions × 8 rollouts × 5 epochs = 96,000 session-rollouts), reward computation evaluates 5 QA questions per rollout, yielding 480,000 QA evaluations. Each evaluation involves two API calls: answer generation and LLM judging, both using GPT-4.1-mini. Answer generation consumes approximately 960M input and 48M output tokens ($461), while LLM judging requires approximately 241M input and 15M output tokens ($120). The per-run total is **$581**.

G Case Study

To illustrate the behavioral changes discussed in Section 5.3.2, we present representative examples comparing model outputs across training stages (Base, SFT, RL) on identical inputs.

G.1 Core Memory: Precision Evolution

Scenario. The user discusses updates to their vegetarian lifestyle. The current Core Memory contains:

```
Dietary preference: Exploring plant-based
options
Family: Has children
Hobbies: Cooking, reading
```

User Message. "I've been vegetarian for about 6 months now. My kids love when I make vegan pancakes on Sunday mornings. We tried oat milk last week and it's now my favorite plant-based milk!"

Base Model. Produces a fragmented append without structure:

```
{"operation": "APPEND", "content": "vegetarian 6
 months. oat milk favorite. vegan pancakes."}
```

SFT Model. Rewrites the entire memory block:

```
{"operation": "REWRITE", "content": "Dietary
preference: Vegetarian for 6 months, exploring
plant-based options; kids love vegan pancakes on
 Sunday mornings; tried oat milk last week and
now it's favorite plant-based milk\nFamily: Has
children\nHobbies: Cooking, reading"}
```

RL Model. Performs a targeted replacement of only the relevant field:

```
{"operation": "REPLACE", "old_text": "Dietary
preference: Exploring plant-based options", "
new_text": "Dietary preference: Vegetarian (6
months); favorite plant milk: oat milk; makes
vegan pancakes for kids on Sundays"}
```

Analysis. The Base model outputs unstructured fragments. SFT learns valid formatting but defaults to REWRITE, risking information loss when reorganizing the entire block. The RL model learns that REPLACE achieves targeted updates while preserving unrelated fields (Family, Hobbies).

G.2 Episodic Memory: Temporal Chain Construction

Scenario. The user follows up on a previously mentioned trip. The retrieved memory contains:

```
[EPISODIC] 2024-02-15: User planned trip to
Japan | Details: Planning vacation to Japan,
interested in visiting Kyoto and Tokyo.
Researching cherry blossom season timing.
```

User Message. "Remember when I told you about my trip to Japan last month? Well, I finally uploaded all the photos! I got some amazing shots at the bamboo forest in Kyoto."

Base Model. Adds a disconnected entry with no reference to the planning phase:

```
{"operations": [{"action": "ADD", "memory": "
User went to Japan. Took photos at bamboo forest
."}]}
```

SFT Model. Overwrites the planning record with completion information:

```
{"operations": [{"action": "UPDATE", "old_memory
": "2024-02-15: User planned trip to Japan...",
"new_memory": "2024-03-15: User completed trip
to Japan | Details: Took trip to Japan last
month. Visited Kyoto and Tokyo. Took photos at
bamboo forest in Arashiyama. Uploaded all photos
."}]}
```

RL Model. Creates a new event entry and updates the original with status annotation:

```
{"operations": [{"action": "ADD", "memory":
"2024-03-10: User visited bamboo forest in
Arashiyama, Kyoto | Details: Took amazing photos
 during trip to Japan; shared photos
highlighting scenic beauty and personal
enjoyment"}, {"action": "UPDATE", "old_memory":
"2024-02-15: User planned trip to Japan...", "
new_memory": "2024-02-15: User planned trip to
Japan [COMPLETED] | Details: Planning vacation
to Japan -> Trip completed March 2024. Visited
Kyoto (bamboo forest) and Tokyo. Photos uploaded
."}]}
```

Analysis. The Base model creates isolated entries without temporal connections. SFT's UPDATE operation overwrites planning details, losing information about the user's original interests (cherry blossom season research). The RL model constructs a temporal chain: preserving the planning phase, adding the completion event as a separate entry, and marking status transitions.

G.3 Semantic Memory: Comprehensive Fact Capture

Scenario. The user expresses interest in art after visiting a museum.

User Message. "I've been really into impressionist art lately. Went to see the Monet exhibition at the museum last weekend. His water lilies series is absolutely stunning."

Base Model. Adds a simple entry with minimal detail:

```
{"operations": [{"action": "ADD", "memory": "
User likes impressionist art. Saw Monet
exhibition."}]}
```

SFT Model. Updates an existing hobby entry:

```
{"operations": [{"action": "UPDATE", "old_memory
": "Hobbies: Enjoys visiting museums | Details:
Regular museum visitor...", "new_memory": "
Hobbies: Enjoys visiting museums | Details:
Regular museum visitor, interested in various
exhibitions, particularly impressionist art.
Recently saw Monet’s water lilies series
exhibition at a museum last weekend."}]}
```

RL Model. Creates multiple entries capturing both user preference and conceptual knowledge:

```
{"operations": [{"action": "ADD", "memory": "
User - Art Interest: Impressionist art | Details:
 Recently developed strong interest in
impressionist art, particularly Monet’s water
lilies series. Describes them as ’absolutely
stunning’."}, {"action": "ADD", "memory": "
Concept - Monet’s Water Lilies | Details: A
series of paintings by Claude Monet depicting
water lilies, displayed at Musee de l’Orangerie
in Paris."}]}
```

Analysis. The Base model captures only surface-level facts. SFT consolidates new information into existing entries, compressing the user's art interest into a general hobby record. The RL model learns to output multiple distinct entries: one for user-specific preferences (interest in impressionist art) and another for conceptual knowledge (Monet's Water Lilies). This separation enables more precise retrieval when answering questions about the user's artistic taste versus factual questions about specific artworks.

G.4 Procedural Memory: Version Preservation

Scenario. The user describes a refined coffee routine. The retrieved memory contains:

```
[PROCEDURAL] User’s coffee brewing method |
Steps: 1. Grind beans. 2. Add hot water. 3. Wait
 4 minutes. 4. Press and pour. | Context: French
 press method.
```

User Message. "I finally perfected my morning coffee routine! I grind 18g of beans, use 92°C water, bloom for 30 seconds, then do a slow pour over 3 minutes."

Base Model. Adds a simple entry without procedural structure:

```
{"operations": [{"action": "ADD", "memory": "
User has new coffee method: 18g beans, 92C water,
 30s bloom, 3min pour."}]}
```

SFT Model. Overwrites the French press method entirely:

```
{"operations": [{"action": "UPDATE", "old_memory
": "User’s coffee brewing method | Steps: 1.
Grind...", "new_memory": "User’s pour-over
coffee brewing method | Steps: 1. Grind 18g of
beans. 2. Use 92C water. 3. Bloom for 30 seconds.
 4. Slow pour over 3 minutes. | Context: Pour-
over method, refined morning routine."}]}
```

RL Model. Adds the new method as a versioned entry, preserving the original:

```
{"operations": [{"action": "ADD", "memory": "
Pour-over coffee method (v2, current) | Steps: 1.
 Grind 18g beans to medium-fine. 2. Heat water
to 92C. 3. Bloom for 30 seconds. 4. Slow pour in
 circular motion for 3 minutes. 5. Allow to drip.
 | Context: Refined morning routine, replaces
previous French press method"}]}
```

Analysis. This case exhibits the starkest contrast. SFT's UPDATE completely erases the French press procedure—if the user later asks "How did I use to make coffee with my French press?", the system cannot answer. The RL model learns through downstream QA feedback that procedural knowledge should be versioned rather than overwritten. The new entry explicitly references the previous method while the original remains retrievable.

H Prompt Templates

This section presents the prompt templates used in our framework. For readability, we have made minor formatting adjustments (e.g., line breaks and indentation) to the original prompts.

Core Memory Prompt

You are the Core Memory Manager. Your role is to analyze user messages and extract fundamental information about the user that will be beneficial in future conversations.
Current Core Memory (Human Block): {{current_core_memory}}
Character Usage: {{core_usage}}%
New Messages: {{messages}}

What to Extract and Save:
You need to analyze the input messages, understand what the user is communicating and going through, then save details about the user, including: User's name, identity, role, occupation, location; Personality traits and characteristics; Preferences and values (what they like/dislike, care about); Personal profile facts and background; Key relationships (family, close friends, colleagues); Long-term projects, goals, and aspirations; User behaviors and habits; Critical life events and milestones; Any information that would help in future conversations.

Examples of Good Core Memory Entries:
- "Is a software engineer at Google, specializing in machine learning"
- "Loves to play Cyberpunk 2077, prefers RPG games over shooters"
- "Has publications: 1. Paper on NLP transformers 2. Book on AI Ethics"
- "Close friend: Emma (marathon runner), meets weekly for coffee"
- "Working on long-term project: Building a personal knowledge management system"
- "Personality: Introverted, analytical, values deep conversations over small talk"

Instructions:
1. Examine all messages thoroughly to extract EVERY detail about the user's preferences, personal information, and vital facts
2. Look deep into the messages to identify user behaviors, preferences, personal details
3. Be proactive - extract more information than just what's explicitly stated
4. The core memory can be as detailed as possible - capture context and nuance
5. Decide on ONE operation: APPEND: Add new information to existing block (if <90% full); REPLACE: Update specific outdated or incorrect information; REWRITE: Reorganize and consolidate the entire block (if >90% full or major updates needed)

Return JSON with ONE of these operations:
{"operation": "APPEND", "content": "Additional text to append"}
OR
{"operation": "REPLACE", "old_text": "Text to replace", "new_text": "Replacement text"}
OR
{"operation": "REWRITE", "content": "Complete rewritten human block (keep under 5000 chars)"}

Focus on user identity, preferences, personality traits, and vital facts that would improve future interactions.
CRITICAL: Return ONLY the JSON object. Do NOT add any explanations, analysis, or additional text after the JSON.

Figure 8: Prompt template for Core Memory.

Episodic Memory Prompt

You are the Episodic Memory Manager. Manage time-ordered event memories.
Episodic Memory stores time-ordered, event-based information from interactions—essentially, the "diary" of user events.
Each episodic memory MUST include: (a) summary: Short textual summary of the event (concise and informative); (b) timestamp: When the event occurred (format: "YYYY-MM-DD HH:MM", "YYYY-MM-DD", "YYYY-MM", or "YYYY" depending on precision available); (c) details: Detailed description capturing AS MANY DETAILS AS POSSIBLE - who, what, when, where, why, specific objects mentioned, colors, numbers, names, emotions, context, etc.; (d) event_type: Type of event (e.g., "conversation", "activity", "observation", "plan").
CRITICAL: Each event must clearly identify whose event this is (who experienced or performed it).

IMPORTANT TIMESTAMP RULES:
Conversation Timestamp: {{conversation_timestamp}}
1. One Event Per Timestamp: Each memory = ONE specific event at ONE point in time; Multiple events in one message → create SEPARATE memories
2. Timestamp Format (Use ABSOLUTE time only): Use ONLY absolute dates: "YYYY-MM-DD", "YYYY-MM", or "YYYY"; "yesterday" → calculate and use YYYY-MM-DD; "last week" / "last month" → calculate and use YYYY-MM; "this past weekend" → calculate and use YYYY-MM-DD; No time mentioned → use conversation timestamp; Unclear → use YYYY-MM or YYYY (do NOT guess specific dates); NEVER include relative expressions like "(last month)" or "(yesterday)" in the timestamp
3. Preserve Original Time Expression in Details (REQUIRED): ALWAYS start Details with time context; User says "last month" → Details starts with "Last month from conversation date of {{conversation_timestamp}} (calculated as YYYY-MM), ..."; User says "yesterday" → Details starts with "Yesterday from conversation date of {{conversation_timestamp}} (YYYY-MM-DD), ..."; User says "this past weekend" → Details starts with "This past weekend from conversation date of {{conversation_timestamp}} (week of YYYY-MM-DD), ..."; No time mentioned → Details starts with "Mentioned during conversation on {{conversation_timestamp}}, ..."

Existing Recent Episodic Memories: {{existing_episodic}}
New Messages: {{messages}}

Analyze the messages and extract time-ordered events and decide on operations: For each new event, determine if it should be: ADD: Completely new event not in memory; UPDATE: Add new related event that references previous events (old versions remain for history); MERGE: Combine multiple related events into a timeline with timestamp range, drawing well-supported conclusions from the pattern (old versions remain for history).

CRITICAL REQUIREMENTS for the "memory" field:
Start with ABSOLUTE timestamp only (use appropriate precision: "YYYY-MM-DD HH:MM", "YYYY-MM-DD", "YYYY-MM", or "YYYY"); NO relative time expressions in timestamp (no "last month", "yesterday", etc.); Follow with ": " then brief summary, then " | Details: "; Details MUST start with time context if event time differs from conversation time; Record AS MANY DETAILS AS POSSIBLE: names, objects, colors, numbers, sizes, emotions, locations, specific quotes, future plans; Capture visual details (e.g., "black and white bowl", "purple running shoes", "sunset with palm tree"); Include context and background information; For UPDATE: Create new event with its own timestamp that references previous events (old versions remain for history); For MERGE: Create timeline with timestamp range, synthesizing events and drawing well-supported conclusions from patterns. Only include conclusions that are clearly evidenced by the events (old versions remain for history).

Return JSON format:
{"operations": [{"action": "ADD", "memory": "2024-03-15 19:00: Alex attended first Italian cooking class | Details: ..."}, {"action": "UPDATE", "old_memory": "...", "new_memory": "..."}, {"action": "MERGE", "old_memories": ["...", "..."], "new_memory": "..."}]}
CRITICAL: Return ONLY the JSON object. Do NOT add any explanations, analysis, or additional text after the JSON.

Figure 9: Prompt template for Episodic Memory.

Procedural Memory Prompt

You are the Procedural Memory Manager. Manage step-by-step processes, workflows, and instructions.
Procedural Memory contains how-to guides, step-by-step instructions, or processes the user might follow.
Each procedural memory entry MUST include: (a) entry_type: Type of procedure (e.g., "workflow", "guide", "recipe", "troubleshooting", "routine"); (b) description: Short descriptive text explaining what the procedure is for; (c) steps: The procedure in clear, numbered steps (can be text or structured format); (d) context: When/where/why this procedure is used (optional but helpful).
Existing Procedural Memories: {{existing_procedural}}
New Messages: {{messages}}

Analyze the messages and extract procedural knowledge.
CRITICAL REQUIREMENTS for the "memory" field:
Start with description, then " | Steps: " with numbered steps; Number all steps clearly (1, 2, 3...); Include specific details: times, temperatures, quantities, tools, materials; Optionally add " | Context: " at the end; Most conversations won't have procedural content - return empty operations array.

Return JSON:
{"operations": [{"action": "ADD", "memory": "How Ryan brews cold brew coffee | Steps: 1. Grind 1 cup... 2. Add grounds... | Context: Ryan's weekly coffee preparation routine."}, {"action": "UPDATE", "old_memory": "...", "new_memory": "..."}]}
CRITICAL: Return ONLY the JSON object. Do NOT add any explanations, analysis, or additional text after the JSON.

Figure 10: Prompt template for Procedural Memory.

Core Memory Compress Prompt

The Core Memory is too long ({len(content)} chars, limit: {CORE_MEMORY_HUMAN_CHAR_LIMIT}).
Compress it to under 3000 characters, keeping only core identity and critical facts: User's name, role, occupation, key relationships; Personality traits and important preferences; Long-term goals and critical life events; Unique characteristics that define the user.
Remove or compress: Redundant descriptions and verbose explanations; Minor details and conversational context; Detailed examples (keep only key takeaways).
Current content: {content}
Output format: {"content": "compressed version under 3000 chars"}
Respond with ONLY the JSON object, no other text.
WARNING: IMPORTANT (second compression): Be more aggressive in compression this time - the previous attempt still exceeded the limit. Remove all non-essential information while preserving the user's core identity.

Figure 11: Prompt template for Core Memory compression.

Answer Generation Prompt

{context}{time_context}
Question: {question}

Instructions: 1. Carefully analyze the retrieved memories to find relevant information; 2. Consider synonyms and related concepts (e.g., "support group", "activist group" may refer to similar things); 3. If memories mention specific dates/times, use those to answer time-related questions; 4. If memories contain contradictory information, prioritize the most recent memory; 5. Focus on the content of the memories, not just exact word matches.

For factual questions (What/When/Where/Who): Answer based on direct information in the memories; If the specific fact is not mentioned, respond: "Not answerable".
For inference/reasoning questions (Would/Could/Likely): You CAN make reasonable inferences based on related information in the memories; Example: If asked "Would X pursue career Y?" and memories show X wants career Z, you can infer "Likely no, X wants Z instead"; Example: If asked "Would X be considered religious?" and memories show X's interactions with religious topics, you can infer based on those interactions.
When to say "Not answerable": If the question asks about a specific person but the memories are about a DIFFERENT person, respond: "Not answerable"; If the question asks about an event/action that is NOT mentioned in ANY of the memories AND there's no related information to make an inference, respond: "Not answerable"; If you find information about a similar but DIFFERENT event (e.g., question asks about "Caroline's charity race" but memories only mention "Melanie's charity race"), respond: "Not answerable".
IMPORTANT for "Not answerable" responses: Simply state "Not answerable" without lengthy explanations; Do NOT add phrases like "There is no direct record" or "does not appear to be"; Keep it concise: just "Not answerable" is sufficient.

Provide a concise, direct answer based on the available information, or state "Not answerable" if the specific information requested is not present or is about a different person/entity.

Figure 12: Prompt template for QA answering.

LLM Judge Prompt

Your task is to label an answer to a question as 'CORRECT' or 'WRONG'. You will be given: (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.

Handling "Not answerable" cases:

1. If the GOLD answer is "Not answerable" (meaning the information truly doesn't exist in the conversation history): The generated answer should be CORRECT if it clearly indicates unavailability; Accept equivalent expressions: "Not answerable", "There is no information", "There is no direct record", "does not appear to be", "no explicit mention", "cannot be determined", "no specific details available"; As long as the generated answer conveys that the information is unavailable, count it as CORRECT.
2. If the GOLD answer is a SPECIFIC answer (e.g., "7 May 2023", "John", "Paris"): The generated answer saying "Not answerable" should be counted as WRONG; This means the system failed to retrieve information that actually exists in the conversation history; Even if phrased as "no information available" or similar, it's still WRONG when the gold answer is specific; IMPORTANT: Even if the generated answer mentions the correct information but attributes it to a DIFFERENT person/entity than asked in the question, it should be counted as WRONG. For example, if the question asks about "Alice's opinion" but the answer says "Bob thinks X" (even if X matches the gold answer), this is WRONG because it answers about the wrong person.
3. CRITICAL RULE for "Not answerable" responses: When the generated answer indicates "Not answerable" or similar (cannot find, no information, etc.), the ONLY way it can be CORRECT is if the GOLD answer is ALSO "Not answerable"; If the gold answer contains ANY specific information (names, dates, facts, opinions, etc.), then a "Not answerable" response is ALWAYS WRONG, regardless of any explanation or reasoning provided in the generated answer; Do NOT be misled by keywords in the explanation - focus on whether the answer actually provides the requested information.

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".

Figure 13: Prompt template for LLM judge evaluation.

Synthetic QA Generation Prompt

You are an expert at generating precise, specific verification questions for testing memory systems.

EVALUATION SCENARIO:
You are creating questions to test a memory system. Here's how the evaluation works: 1. **Memory Building Phase (Already Done)**: A memory system has processed the conversation history up to this point and stored memories in a vector database. 2. **Question Answering Phase (What You're Preparing For)**: The answering model will receive ONLY your question; The answering model will search the memory database using your question as a query; The answering model will retrieve relevant memories (episodic, semantic, procedural); The answering model will answer based ONLY on retrieved memories; **CRITICAL**: The answering model CANNOT see the original conversation text. 3. **Your Task**: Generate questions that: Test whether the memory system correctly captured information from the current session; Include enough context/anchors so the question itself can retrieve the right memories; Are answerable using only the information stored in the memory database.

Memory State from Previous Steps (All Retrieved Memories):
Core Memory: {core_memory}
Episodic Memories: {episodic_memories}
Semantic Memories: {semantic_memories}
Procedural Memories: {procedural_memories}
Current Session Conversation (Newly Added): {current_session}
Session Timestamp: {session_timestamp}

QUESTION GENERATION GUIDELINES:
Critical Rules: 1. **Use First Person Perspective**: All questions MUST be phrased from the user's perspective using "I/my/me". ✅ CORRECT: "What is my favorite hobby?" ❌ WRONG: "What is the user's favorite hobby?". 2. **Ask About Facts, NOT Opinions**: Questions must have objective, verifiable answers. ✅ CORRECT: "What city did I visit last month?" (factual, verifiable) ❌ WRONG: "How do I feel about my job?" (subjective, opinion-based). 3. **Single Retrievable Answer**: Each question should have ONE clear answer that can be found through memory search. 4. **Natural Question Phrasing**: Use conversational, natural language. 5. **Be Specific with Anchors**: Each question MUST include specific anchoring information (names, dates, places, events, products, activities) to help retrieve the correct memories. 6. **Avoid Vague Reasoning**: Do NOT ask abstract relationship questions like "How does X relate to Y?". 7. **Concrete Facts Only**: Focus on verifiable, concrete facts that have clear, unambiguous answers.

Question Types: current_session: Ask about NEW information from the current session; cross_session: Connect current session mentions with historical details.
QUESTION TYPES AND DISTRIBUTION: Generate exactly {num_questions} questions with the following distribution: 1. **single-session** (Target: 50% = 2-3 questions): Tests memory retention from current session ONLY; Information found ONLY in the current session. 2. **multi-session** (Target: 30% = 1-2 questions): Requires information from MULTIPLE sessions; Needs to aggregate/count/compare across sessions. 3. **temporal-reasoning** (Target: 20% = 1 question): Involves time calculation, date comparison, or event ordering; Requires reasoning about temporal relationships.

OUTPUT FORMAT:
Return a JSON object with this EXACT structure:
{"questions": [{"question": "What is my favorite hobby?", "answer": "Photography", "type": "single-session|multi-session|temporal-reasoning|knowledge-update", "source": "current_session|cross_session"}, ... (exactly {num_questions} questions total)]}
Return ONLY the JSON object, no additional text.

Figure 14: Prompt template for synthetic question generation.