# Memory-R2: Fair Credit Assignment for Long-Horizon Memory-Augmented LLM Agents

**Sikuan Yan**[*1,2,3], **Ahmed Bahloul**[*4], **Ercong Nie**[1], **Susanna Schwarzmann**[3], **Riccardo Trivisonno**[3], **Volker Tresp**[1,2], **Yunpu Ma**[† 1,2]
[1]Ludwig Maximilian University of Munich, [2]Munich Center for Machine Learning,
[3]Huawei Heisenberg Research Center (Munich), [4]Technical University of Munich
s.yan@campus.lmu.de, cognitive.yunpu@gmail.com

## Abstract

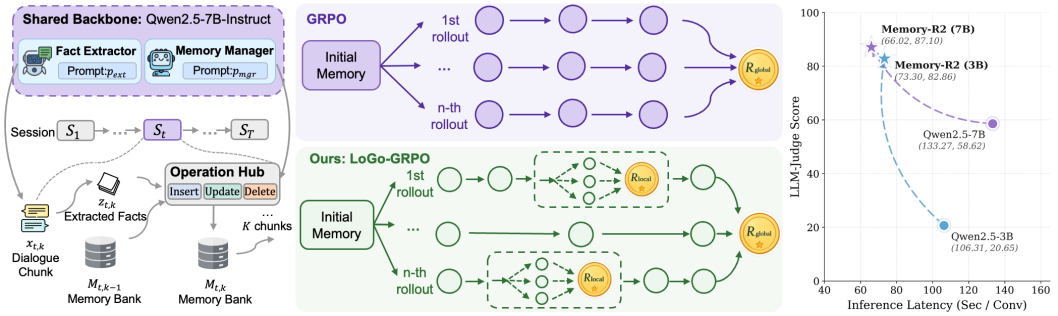
Memory-augmented LLM agents enable interactions that extend beyond finite context windows by storing, updating, and reusing information across sessions. However, training such agents with reinforcement learning in multi-session environments is challenging because memory turns the agent's past actions into part of its future environment. Once different rollouts write, update, or delete different memories, they no longer share the same intermediate memory state, making trajectory-level comparisons fundamentally unfair. This violates a key assumption behind group-relative methods such as GRPO, where rollouts are compared as if they were sampled from the same effective environment. Consequently, trajectory-level rewards provide noisy or biased credit signals for long-horizon memory operations. To address this challenge, we introduce **Memory-R2**, a training framework for long-horizon memory-augmented LLM agents. Its core algorithm, **LoGo-GRPO**, combines local and global group-relative optimization. The global objective preserves end-to-end learning from long-horizon trajectory-level rewards, while local rerollouts compare different memory-operation outcomes from the same intermediate memory state, yielding fairer group comparisons and more precise supervision for memory construction. Beyond credit assignment, Memory-R2 jointly optimizes memory formation and memory evolution with a shared-parameter co-learning design, where a fact extractor and a memory manager are instantiated from the same LLM backbone through role-specific prompts. To stabilize multi-step RL over long memory horizons, we adopt a progressive curriculum that increases the training horizon from 8 to 16 to 32 sessions. Together, these components provide an effective training paradigm for memory-augmented LLM agents in long-horizon multi-session settings.

## 1 Introduction

Large language models (LLMs) have rapidly evolved from standalone text generators into agentic systems that can plan [24], use tools [12, 4], and interact over long horizons [18]. A central requirement for such agents is the ability to accumulate, update, and reuse information across interactions. However, despite strong in-context reasoning ability, LLM agents remain fundamentally constrained by finite context windows and the lack of persistent state, making it difficult to retain salient user information, track long-term goals, or maintain consistency over extended multi-session interactions [7, 9].

[*]Equal contribution.
[†]Corresponding author. The code is available for access via this repository.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).




(a) Memory-R2 Architecture (b) Fair Credit Assignment with LoGo-GRPO (c) Accuracy–Efficiency Gains

Figure 1: Overview of Memory-R2. (a) Memory-R2 uses a shared-backbone extractor–manager architecture for chunk-wise memory construction. (b) LoGo-GRPO contrasts with standard GRPO by introducing local rerollouts from shared intermediate memory states for fairer credit assignment while preserving global trajectory-level optimization. (c) Memory-R2 improves accuracy and inference latency across backbones.

To address this limitation, a growing body of work augments LLM agents with explicit memory systems [27, 21]. Existing research broadly follows two directions. The first focuses on memory infrastructure, including graph-structured memory, structured memory schemas, and system-inspired memory organization [13, 1, 6, 23, 26, 8, 5]. The second focuses on memory policy learning, where reinforcement learning (RL) is used to decide what to extract, how to update memory, and how to use retrieved memory [22, 17]. While these efforts have substantially improved long-horizon agent behavior, training memory agents in multi-session environments remains fundamentally challenging.

The core difficulty is that memory makes the environment non-stationary. In multi-session agent training, memory turns the agent's past actions into part of its future environment: what the agent writes, updates, or deletes in one session becomes the state inherited by subsequent sessions. This creates a fundamental challenge for trajectory-level RL, especially for group-relative methods such as GRPO [2], which rely on comparing rollouts sampled from the same effective environment. Once rollouts modify memory differently, they no longer share the same intermediate memory state, yet GRPO still normalizes their rewards within a single comparison group, leading to unfair comparisons and biased credit assignment. The problem is further amplified by trajectory-level rewards: when a downstream failure occurs, it is difficult to determine whether it comes from the current session's memory operation, corrupted memory inherited from earlier sessions, or later updates that overwrite useful information. This raises a simple but important question:

*How can we design a training paradigm for memory-augmented agents that provides more accurate and fair credit assignment across sessions?*

In this work, we present **Memory-R2**, a training framework for long-horizon memory-augmented LLM agents, as illustrated in Figure 1. At its core is **LoGo-GRPO**, a credit-assignment algorithm that combines *global* and *local* group-relative optimization. LoGo-GRPO preserves a trajectory-level global reward for end-to-end long-horizon optimization, while additionally introducing session-wise attribution signals and local rerollouts that compare trajectories starting from identical intermediate memory states. This yields fairer group comparison and cleaner supervision for memory operations.

Beyond fair credit assignment, Memory-R2 is designed to optimize the whole memory lifecycle. Recent analyses decompose agentic memory into memory formation, memory evolution, and memory retrieval [3], whereas prior RL-based memory work has focused primarily on evolution and retrieval [22]. Our framework targets memory formation and evolution through two cooperative roles: a *fact extractor*, which identifies salient information from the interaction context, and a *memory manager*, which decides whether to insert, update, or delete memory entries. Inspired by shared-policy multi-agent RL [16], we instantiate both roles with a shared LLM backbone and role-specific prompts, enabling parameter-efficient co-learning and tighter coordination between extraction and memory editing.

We further formulate memory construction as a multi-step decision process within each session. Rather than treating a session as a single monolithic transition, we divide it into chunks and allow the fact extractor and memory manager to alternate over them, turning memory construction into a temporally extended process that can be refined as more evidence becomes available. To stabilize long-horizon optimization, we also introduce a curriculum over session horizon, progressively scaling training from 8 to 16 to 32 sessions so that the model first acquires reliable short-horizon memory behavior before adapting to more challenging long-context settings. Our contributions are summarized as follows:

- We propose **Memory-R2**, a training framework for long-horizon memory-augmented LLM agents, whose core algorithm **LoGo-GRPO** improves fairness and session-level credit assignment through global-local group-relative optimization.
- We introduce a **shared-parameter extractor–manager architecture** and formulate memory construction as a **multi-step decision process** over chunked sessions, enabling joint optimization of memory formation and evolution.
- We develop a **curriculum learning strategy** over session horizon that stabilizes long-horizon RL training, and show that the resulting system is highly **data-efficient**, achieving strong gains over prior memory-agent baselines using only two training conversations while generalizing across benchmarks, model scales, and answer agents.

## 2 Related Work

### 2.1 Memory Agent Architectures

Explicit memory has become a standard way to extend LLM agents beyond finite context windows and support long-horizon interaction [21, 27]. Prior work mainly differs in how memory is represented and managed. Representative examples include graph- or structure-based memory systems such as Zep [13], G-Memory [26], A-MEM [21], Mem0 [1], and CAM [6], as well as system-inspired designs such as MemOS [8] and MemoryOS [5]. While these methods propose increasingly expressive memory substrates, they mostly rely on heuristic or prompt-based policies for deciding what to store, update, or discard. In contrast, our work retains a modular extractor–manager architecture but optimizes the memory lifecycle directly with reinforcement learning.

### 2.2 Reinforcement Learning for Memory Agents

Reinforcement learning has recently become an effective paradigm for training LLM agents in interactive settings such as tool use, web navigation, and reasoning [12, 4, 18, 2]. This is particularly suitable for memory agents, where the quality of extraction, memory editing, and retrieval decisions is only revealed through downstream task performance. Existing RL-based memory methods, such as Memory-R1 [22] and Mem-$\alpha$ [17], demonstrate the promise of this direction. However, they rely mainly on outcome-level rewards and do not explicitly address cross-session credit assignment under diverging memory states. They also focus primarily on memory evolution and retrieval, leaving joint optimization of formation, evolution, and retrieval underexplored [3]. Our work addresses these gaps by introducing multi-step extractor–manager training, shared-parameter co-learning, and a global-local GRPO objective for fairer credit assignment in long-horizon multi-session settings.

## 3 Method

### 3.1 Problem Formulation: Multi-step Memory Bank Construction

We study memory bank construction for long-horizon multi-session interactions. Let $\mathcal{D} = \{S_t\}_{t=1}^{T}$ denote a dialogue trajectory of $T$ sessions, where each session $S_t = \{x_{t,k}\}_{k=1}^{K}$ is divided into $K$ chunks. The agent maintains an external memory bank $\mathcal{M}$ that evolves across sessions. We formulate memory construction as a chunk-wise multi-step process, illustrated in Figure 1(a): for each chunk $x_{t,k}$, a fact extractor first proposes salient content

$$z_{t,k} \sim \pi_{\text{ext}}(z \mid x_{t,k}) \,, \tag{1}$$

and a memory manager then chooses an operation conditioned on the extracted content and current memory state,

$$a_{t,k} \sim \pi_{\text{mgr}}(a \mid z_{t,k}, \mathcal{M}_{t,k-1}), \tag{2}$$

where $a_{t,k} \in \mathcal{A}$ denotes operations such as `INSERT`, `UPDATE`, and `DELETE`. The memory bank is updated by a deterministic transition operator

$$\mathcal{M}_{t,k} = \mathcal{T}(\mathcal{M}_{t,k-1}, z_{t,k}, a_{t,k}). \tag{3}$$

This yields a chunk-wise memory construction process over session $t$:

$$\mathcal{M}_{t,0} \xrightarrow[\pi_{\text{ext}},\, \pi_{\text{mgr}}]{x_{t,1}} \mathcal{M}_{t,1} \xrightarrow[\pi_{\text{ext}},\, \pi_{\text{mgr}}]{x_{t,2}} \cdots \xrightarrow[\pi_{\text{ext}},\, \pi_{\text{mgr}}]{x_{t,K}} \mathcal{M}_{t,K}, \tag{4}$$

Across the full dialogue trajectory, let $\tau = \{z_{t,k}, a_{t,k}\}_{t=1,k=1}^{T,K}$ denote a memory-construction rollout. Its probability factorizes as

$$p_\theta(\tau \mid \mathcal{D}) = \prod_{t=1}^{T} \prod_{k=1}^{K} \pi_{\text{ext}}(z_{t,k} \mid x_{t,k})\, \pi_{\text{mgr}}(a_{t,k} \mid z_{t,k}, \mathcal{M}_{t,k-1}). \tag{5}$$

In our framework, the extractor and the manager are implemented as two cooperative roles instantiated from a shared LLM backbone with role-specific prompts:

$$\pi_{\text{ext}}(\cdot) = \pi_\theta(\cdot \mid p_{\text{ext}}, \cdot), \qquad \pi_{\text{mgr}}(\cdot) = \pi_\theta(\cdot \mid p_{\text{mgr}}, \cdot), \tag{6}$$

where $\theta$ denotes the shared model parameters, and $p_{\text{ext}}$ and $p_{\text{mgr}}$ are role-specific prompts for fact extraction and memory management, respectively. The resulting memory-construction rollout $\tau$ is evaluated through downstream task performance, yielding a trajectory-level reward $R(\tau)$. We optimize the shared memory policy by maximizing the expected return $\mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$.

### 3.2 Length-Normalized Step-level RL with Shared Extractor–Manager Policy

While Sec. 3.1 defines memory construction as a multi-step process, optimizing it with a shared LLM policy introduces length-induced bias. We instantiate fact extraction and memory management as two roles of a shared policy with role-specific prompts [16]. Since the two roles produce outputs of different lengths, token-level RL assigns more loss terms to longer generations, biasing the shared policy toward verbose outputs and roles with longer outputs. To address this, we use a *length-normalized step-level* objective, treating each extractor or manager call as one generation step. For a generation step $u$ with generated token indices $\mathcal{U}_u$, we aggregate token-level ratios and advantages as

$$\rho_u = \exp\left(\frac{1}{|\mathcal{U}_u|} \sum_{\ell \in \mathcal{U}_u} \log \frac{\pi_\theta(y_\ell \mid h_\ell)}{\pi_{\theta_{\text{old}}}(y_\ell \mid h_\ell)}\right), \qquad \bar{A}_u = \frac{1}{|\mathcal{U}_u|} \sum_{\ell \in \mathcal{U}_u} A_\ell, \tag{7}$$

where $\rho_u$ is the step-level importance ratio, $\bar{A}_u$ is the step-level advantage, $y_\ell$ is a generated token, $h_\ell$ is its autoregressive context, $A_\ell$ is the token-level advantage, and $\pi_{\theta_{\text{old}}}$ is the rollout policy. This gives each generation step comparable weight regardless of output length. The resulting $\rho_u$ and $\bar{A}_u$ are then used in the LoGo-GRPO objective.

### 3.3 LoGo-GRPO for Multi-session Credit Assignment

The formulation in Sec. 3.1 defines memory construction as a chunk-wise multi-step process, but learning still requires fair credit assignment across sessions. Trajectory-level GRPO is problematic in memory-augmented settings because memory turns an agent's past actions into part of its future environment. Once rollouts write, update, or delete different memories, they no longer share the same intermediate memory state, making group-relative comparisons unfair and credit signals noisy or biased. To address this, we propose **LoGo-GRPO**, which combines a global trajectory-level branch with a local rerollout branch. As shown in Figure 1 (b), the global branch preserves end-to-end optimization over the full multi-session trajectory, while the local branch rerolls a stochastically sampled subset of sessions from shared memory states, yielding lower-bias session-level credit assignment at manageable cost.

**Reward function.** Let $\mathcal{Q}$ denote the full set of question-answer pairs $(q, a^*)$ associated with a training conversation, and let $\mathcal{Q}_t \subseteq \mathcal{Q}$ denote the subset whose required evidence is attributed to session $t$. Given a memory bank $\mathcal{M}$ and a question $q$, an answer module retrieves relevant entries from memory and generates an answer $\hat{a}$. We measure QA quality using token-level F1:

$$\mathrm{QA}(\mathcal{M}, \mathcal{Q}_t) = \frac{1}{|\mathcal{Q}_t|} \sum_{(q,a^*)\in\mathcal{Q}_t} \mathrm{F1}(\hat{a}, a^*)\,. \tag{8}$$

To discourage unbounded memory growth, we penalize memory tokens exceeding an $\alpha$ fraction of the cumulative session tokens up to session $t$, where $\mathrm{Tok}(\cdot)$ denotes token count and $\alpha$ is a fixed memory budget ratio:

$$\mathrm{Comp}(\mathcal{M}, t) = \begin{cases} 0, & \mathrm{Tok}(\mathcal{M}) \le \alpha \sum_{s=1}^{t} \mathrm{Tok}(S_s), \\ \dfrac{\mathrm{Tok}(\mathcal{M}) - \alpha \sum_{s=1}^{t} \mathrm{Tok}(S_s)}{\sum_{s=1}^{t} \mathrm{Tok}(S_s)}, & \mathrm{Tok}(\mathcal{M}) > \alpha \sum_{s=1}^{t} \mathrm{Tok}(S_s). \end{cases} \tag{9}$$

The session-level reward is

$$R(\mathcal{M}, \mathcal{Q}_t, t) = \mathrm{QA}(\mathcal{M}, \mathcal{Q}_t) - \lambda_{\mathrm{comp}}\, \mathrm{Comp}(\mathcal{M}, t), \tag{10}$$

where $\lambda_{\mathrm{comp}}$ controls the compression penalty.

**Global branch.** For rollout $i$, let $\mathcal{M}_t^{(i)} \equiv \mathcal{M}_{t,K}^{(i)}$ denote the memory state after session $t$. The global branch evaluates the terminal memory $\mathcal{M}_T^{(i)}$ and attributes the reward to session $t$ according to the location of the required evidence:

$$r_{t,i}^{\mathrm{G}} = R\Big(\mathcal{M}_T^{(i)},\, \mathcal{Q}_t,\, T\Big)\,. \tag{11}$$

Following GRPO, we compute group-relative advantages across the $n$ global rollouts:

$$\hat{A}_{t,i}^{\mathrm{G}} = \frac{r_{t,i}^{\mathrm{G}} - \mu_t^{\mathrm{G}}}{\sigma_t^{\mathrm{G}} + \varepsilon}, \qquad \mu_t^{\mathrm{G}} = \frac{1}{n}\sum_{j=1}^{n} r_{t,j}^{\mathrm{G}}, \qquad \sigma_t^{\mathrm{G}} = \mathrm{std}_j\big(r_{t,j}^{\mathrm{G}}\big)\,. \tag{12}$$

While this branch provides full-horizon supervision, it still suffers from reward contamination: at session $t$, different rollouts induce different intermediate memory states as their effective environments, yet GRPO normalizes their rewards within the same comparison group.

**Local branch with stochastic rerollout.** To reduce this contamination, the local branch performs rerollouts from shared intermediate memory states. After the global rollout phase, each session is independently selected with probability $p_{\mathrm{local}}$:

$$b_t \sim \mathrm{Bernoulli}(p_{\mathrm{local}}), \qquad \mathcal{B} = \{t \mid b_t = 1\}. \tag{13}$$

For each selected session $t \in \mathcal{B}$, we choose an anchor rollout $i_0 \in \{1, \ldots, n\}$, retrieve the cached memory state immediately before session $t$, and sample $m$ local rerollouts of session $t$ only. Since these rerollouts share the same starting memory state $\mathcal{M}_{t-1}^{(i_0)}$, their comparison is not confounded by divergence from earlier sessions. Let $\mathcal{M}_t^{(i_0,j)}$ denote the memory state after the $j$-th local rerollout from this anchor state. The local reward is

$$r_{t,j}^{\mathrm{L}} = R\Big(\mathcal{M}_t^{(i_0,j)},\, \mathcal{Q}_t,\, t\Big)\,, \qquad j = 1, \ldots, m. \tag{14}$$

The corresponding local advantages are computed within the rerollout group:

$$\hat{A}_{t,j}^{\mathrm{L}} = \frac{r_{t,j}^{\mathrm{L}} - \mu_t^{\mathrm{L}}}{\sigma_t^{\mathrm{L}} + \varepsilon}, \qquad \mu_t^{\mathrm{L}} = \frac{1}{m}\sum_{j=1}^{m} r_{t,j}^{\mathrm{L}}, \qquad \sigma_t^{\mathrm{L}} = \mathrm{std}_j\big(r_{t,j}^{\mathrm{L}}\big)\,. \tag{15}$$

Because local advantages are computed among rerollouts from the same anchor memory state $\mathcal{M}_{t-1}^{(i_0)}$, the comparison is fairer than global normalization across already-diverged trajectories.

**Unified training objective.** We optimize the shared memory policy using both global rollouts and local rerollouts. For each generation step $u$, we assign the normalized advantage associated with its corresponding rollout: $\hat{A}^{\mathrm{G}}_{t,i}$ for a step from global rollout $i$ at session $t$, and $\hat{A}^{\mathrm{L}}_{t,j}$ for a step from local rerollout $j$ at session $t$. The same assigned advantage is used as the token-level advantage $A_\ell$ for all tokens in step $u$. Let $\mathcal{K}_{\mathrm{step}}$ denote the set of valid generation steps from both branches. Using the step-level ratio $\rho_u$ and advantage $\bar{A}_u$ from Eq. 7, we optimize the dual-clipped surrogate

$$\ell_u = \begin{cases} \min\big(-c\,\bar{A}_u,\ \max\big(-\rho_u\bar{A}_u,\ -\mathrm{clip}(\rho_u, 1-\epsilon, 1+\epsilon)\bar{A}_u\big)\big), & \bar{A}_u < 0, \\ \max\big(-\rho_u\bar{A}_u,\ -\mathrm{clip}(\rho_u, 1-\epsilon, 1+\epsilon)\bar{A}_u\big), & \bar{A}_u \geq 0, \end{cases} \tag{16}$$

where $c > 1$ is the dual-clipping constant and $\epsilon$ is the clipping threshold. The final actor objective is

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{K}_{\mathrm{step}}|} \sum_{u \in \mathcal{K}_{\mathrm{step}}} \ell_u - \beta_{\mathrm{ent}}\,\overline{H}_{\mathrm{token}} + \beta_{\mathrm{kl}}\,\overline{D}_{\mathrm{KL,token}}, \tag{17}$$

where $\overline{H}_{\mathrm{token}}$ and $\overline{D}_{\mathrm{KL,token}}$ denote the mean token-level entropy and KL divergence, respectively. The proportion of local rerollouts controls the strength of local supervision, allowing LoGo-GRPO to balance end-to-end long-horizon learning with lower-bias session-level credit assignment.

### 3.4 Curriculum Learning for Long-Horizon Credit Assignment

Directly training on long multi-session trajectories is unstable before the model acquires reliable memory manipulation skills. Because memory operations shape the future environment, early insert, update, or delete errors can propagate across sessions and make long-horizon credit assignment increasingly noisy. We therefore adopt a curriculum over session horizon: training starts from shorter sessions, where memory effects are easier to observe and attribute, and gradually increases the horizon as the policy stabilizes. Concretely, we train in three stages with the maximum number of sessions increasing from 8 to 16 to 32. The 8-session stage learns basic memory operations under limited error propagation, the 16-session stage introduces stronger inter-session dependencies, and the 32-session stage enables full long-horizon optimization. For each stage, we select the best validation checkpoint as the initialization for the next stage, providing a stable starting point for longer-horizon training.

## 4 Experiments

### 4.1 Experiment Setup

**Datasets and Evaluation Metrics.** We train on LoCoMo [10], a long-term persona-grounded conversation benchmark, using a 2:1:7 train/validation/test split. For out-of-distribution evaluation, we additionally test on LongMemEval [19], MSC-Self-Instruct [11, 20], and MemBench [15]. We report token-level F1, BLEU-1 (B1), and LLM-as-a-Judge (J) as the primary metrics, and additionally use **M-Fail**, the percentage of required evidence location IDs that are missing from the memory bank, as a diagnostic measure of memory-construction quality. Further details on the M-Fail metric can be found in Appendix C.

**Baselines and Implementation Details.** We compare against A-MEM [21], Mem0 [1], MemoryOS [5], a RAG variant implemented within the Mem0 framework, MEM1 [28], MemAgent [25], and Memory-R1 [22]. Our work primarily targets the memory construction stage: the memory extractor and memory manager share a Qwen2.5-7B-Instruct backbone and are jointly trained, while the answer agent is held fixed during training to provide stable reward signals. We use GPT-OSS-120B as this fixed answer agent, since a weaker answer model would yield noisy reward signals that conflate memory-construction quality with answer-generation errors. To remain consistent with this training pipeline, all reported results in our ablation and analysis experiments use the same GPT-OSS-120B answer agent. In Table 1, however, we additionally train a Qwen2.5-7B-Instruct answer agent and report a backbone-controlled variant of Memory-R2 in which all components share the same 7B backbone, enabling a fair comparison against the baselines. Unless otherwise noted, all baselines also use Qwen2.5-7B-Instruct as the backbone. Additional details are provided in Appendix A.

Table 1: Main results on LoCoMo. We report token-level F1 (F1), BLEU-1 (B1), and LLM-as-a-Judge (J), with the best per column in **bold**. For fair comparison, all baselines and Memory-R2 use Qwen2.5-7B-Instruct as the base model. We additionally report Memory-R2 (GPT-OSS), which swaps the answer agent for GPT-OSS-120B; notably, our RL-finetuned 7B Memory-R2 surpasses this 120B variant on F1 and B1, showing that targeted training outweighs raw model scale. Results are averaged over three runs; standard deviations are in Table 3. †: as reported in [22].

| | Multi-hop | | | Open Domain | | | Single Hop | | | Temporal | | | Overall | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model** | F1 | B1 | J | F1 | B1 | J | F1 | B1 | J | F1 | B1 | J | F1 | B1 | J |
| *Training-free Methods* | | | | | | | | | | | | | | | |
| RAG† | 9.57 | 7.00 | 15.06 | 11.84 | 10.02 | 19.28 | 8.67 | 6.52 | 12.79 | 8.35 | 8.74 | 5.43 | 8.97 | 7.27 | 12.17 |
| A-MEM [21]† | 18.92 | 12.86 | 40.78 | 14.73 | 12.66 | 31.32 | 30.58 | 26.14 | 46.90 | 23.67 | 20.67 | 28.68 | 26.08 | 21.78 | 40.78 |
| Mem0 [1]† | 24.96 | 18.05 | 61.92 | 20.31 | 15.82 | 48.19 | 32.74 | 25.27 | 65.20 | 33.16 | 26.28 | 38.76 | 30.61 | 23.55 | 53.30 |
| MemoryOS [5]† | 29.55 | 22.59 | 48.12 | 21.03 | 18.41 | 38.55 | 40.85 | 36.26 | 63.14 | 26.26 | 19.70 | 24.81 | 34.64 | 29.36 | 51.26 |
| *Trained Methods* | | | | | | | | | | | | | | | |
| MEM1 [28] | 17.15 | 12.72 | 41.29 | 22.67 | 14.72 | 43.10 | 28.05 | 22.96 | 57.22 | 31.77 | 26.40 | 44.73 | 26.55 | 21.38 | 50.78 |
| MemAgent [25] | 35.95 | 24.56 | 70.65 | 29.20 | 25.48 | 62.07 | 50.55 | 43.89 | 78.61 | 23.15 | 16.55 | 56.96 | 40.72 | 33.36 | 71.52 |
| Memory-R1 [22]† | 33.64 | 26.06 | 62.34 | 23.55 | 20.71 | 40.96 | 46.86 | 40.92 | 67.81 | 47.75 | 38.49 | 49.61 | 43.14 | 36.44 | 61.51 |
| Memory-R2(OSS) | 36.37 | 29.79 | **85.07** | **30.22** | **24.50** | **74.14** | 50.98 | 45.79 | **90.58** | **62.48** | **55.33** | **83.33** | 49.67 | 43.77 | **87.10** |
| Memory-R2 | **38.41** | **30.90** | 80.93 | 20.76 | 16.78 | 67.53 | **54.06** | **48.73** | 86.80 | 59.65 | 50.05 | 69.90 | **50.60** | **44.01** | 80.99 |

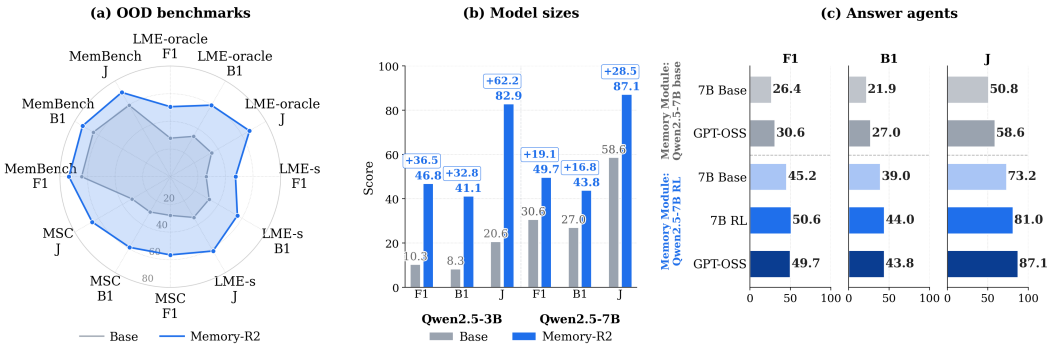



Figure 2: Generalization of Memory-R2 across (a) OOD benchmarks, (b) backbone sizes, and (c) answer agents.

## 4.2 Main Results

**Fair Comparison.** Table 1 reports the main results on LoCoMo. Under the backbone-controlled setting, Memory-R2 achieves the best overall F1 and BLEU-1 among all training-free and trained baselines, including MEM1, MemAgent, and Memory-R1. Compared with the closely related RL baseline Memory-R1, Memory-R2 improves overall F1 from 43.14 to 50.60 and B1 from 36.44 to 44.01, while also reaching a strong judge score of 80.99. These gains are obtained with a simple memory-agent pipeline, suggesting that the improvement mainly comes from the proposed training algorithm rather than additional system complexity. We additionally report Memory-R2 (GPT-OSS), which uses the same memory construction module but replaces the answer agent with GPT-OSS-120B. Memory-R2 with the 7B answer agent achieves higher F1 and BLEU-1 than the GPT-OSS-120B variant, demonstrating that a task-aligned small model can rival a much larger frozen one when paired with a well-trained memory module.

**Strong Generalization.** Figure 2 further demonstrates the strong generalization ability of Memory-R2 from three complementary perspectives. Notably, these gains are achieved even though the model is trained on only two LoCoMo conversations, suggesting that the proposed training paradigm is highly data-efficient. First, Figure 2(a) shows strong transfer to out-of-distribution benchmarks. When evaluated zero-shot on LongMemEval-oracle, LongMemEval-s, MSC-Self-Instruct, and MemBench, Memory-R2 consistently improves over the base model across all reported metrics. For example, on LongMemEval-oracle, the F1 score improves from 27.88 to 50.60, and similar gains are observed on the other benchmarks, indicating that the learned memory-construction policy does not simply overfit to the training benchmark. Second, Figure 2(b) shows that the gains also transfer across model scales. The improvement is especially pronounced for Qwen2.5-3B, where F1 increases from

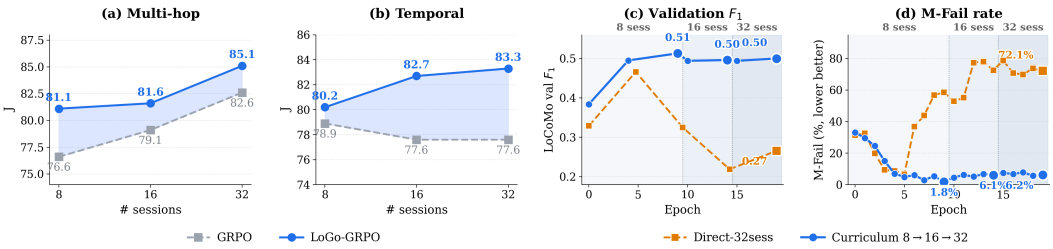



Figure 3: **LoGo-GRPO and curriculum learning are both essential. (a,b)** LoGo-GRPO consistently outperforms GRPO across curriculum stages. **(c,d)** Curriculum training remains stable under equal compute, whereas direct 32-session training collapses validation F1 from 0.47 to 0.27 and increases M-Fail to 72.1%.

10.3 to 46.8, suggesting that our training paradigm is particularly beneficial for smaller-capacity models, for which effective long-horizon memory construction is otherwise difficult to learn. Third, Figure 2(c) decomposes the contribution of training the memory module versus the answer agent. The dominant gain comes from training the memory module (e.g., F1 from 26.4 to 45.2 with a 7B-Base answer agent; F1 from 30.6 to 49.7 with a GPT-OSS answer agent), while varying the answer agent at fixed RL-trained memory yields comparably high scores. This indicates that the benefits of Memory-R2 transfer across diverse downstream answer agents. Taken together, these results indicate that Memory-R2 learns a robust and transferable memory-construction policy rather than overfitting to a specific benchmark, model scale, or answer agent.

### 4.3 Ablation Studies

Table 2 summarizes ablations on the major components of our method, with **M-Fail** reported as a diagnostic measure of memory quality.

Table 2: Ablation studies on components of LoGo-GRPO.

| Variants | F1 ↑ | B1 ↑ | J ↑ | M-Fail ↓ |
|---|---|---|---|---|
| **LoGo-GRPO (full)** | 49.67 | 43.77 | 87.10 | 6.72 |
| *RL Algorithm Ablation* | | | | |
| GRPO | 46.62 (-3.05) | 40.97 (-2.80) | 82.76 (-4.34) | 10.20 (+3.48) |
| -curriculum | 24.12 (-25.55) | 20.67 (-23.10) | 45.99 (-41.11) | 46.50 (+39.78) |
| -length norm. | 43.53 (-6.14) | 38.10 (-5.67) | 77.20 (-9.90) | 8.30 (+1.58) |
| *Architecture Ablation* | | | | |
| Single agent | 39.14 (-10.53) | 33.93 (-9.84) | 72.63 (-14.47) | 18.00 (+11.28) |
| Separate params | 44.31 (-5.36) | 38.59 (-5.18) | 78.89 (-8.21) | 33.00 (+26.28) |
| *Chunks Ablation* | | | | |
| N = 4 | 40.39 (-9.28) | 35.16 (-8.61) | 73.92 (-13.18) | 14.50 (+7.78) |
| N = 8 | 41.37 (-8.30) | 36.15 (-7.62) | 75.58 (-11.52) | 22.70 (+15.98) |
| N = 10 | 37.61 (-12.06) | 32.61 (-11.16) | 70.05 (-17.05) | 37.20 (+30.48) |
| *Training Target Ablation* | | | | |
| memory manager | 45.34 (-4.33) | 39.99 (-3.78) | 81.20 (-5.90) | 16.10 (+9.38) |
| fact extractor | 28.30 (-21.37) | 24.41 (-19.36) | 52.90 (-34.20) | 56.50 (+49.78) |

Replacing LoGo-GRPO with standard GRPO degrades F1 from 49.67 to 46.62 and B1 from 43.77 to 40.97, confirming the benefit of global-local credit assignment. Figure 3(a, b) shows this gap holds at every stage across question types, indicating that local rerollouts consistently mitigate credit-assignment bias. Removing curriculum learning (−curriculum) causes a much larger drop—F1 falls to 24.12 and M-Fail rises to 46.5%. Figure 3(c, d) traces this collapse: direct 32-session training peaks at $F_1$=0.47 before falling to 0.27, while M-Fail explodes from below 10% to over 70%; the curriculum instead stabilizes around $F_1$=0.50 with M-Fail held under 7%. This confirms that early errors propagate across sessions and corrupt memory, and that progressive horizon expansion is essential for stable long-horizon training. We further ablate the length normalization in our step-level objective: switching to a token-level loss (−length norm.) drops F1 to 43.53 and B1 to 38.10, confirming that length-normalized step weighting is necessary to prevent output length bias under the shared extractor–manager policy.

For the memory-construction architecture, a single-agent variant merging extraction and editing into one role drops to 39.14 F1, and a separate-params variant where the extractor and manager use disjoint parameters also underper-

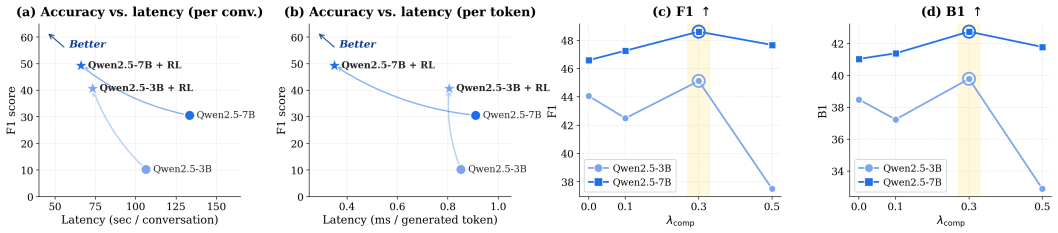



Figure 4: **Inference efficiency and compression penalty analysis. (a,b)** Accuracy–latency trade-off measured by F1 vs. time per conversation and per generated token. **(c,d)** Effect of $\lambda_{\text{comp}} \in \{0, 0.1, 0.3, 0.5\}$ on F1 and BLEU-1; the yellow band marks $\lambda_{\text{comp}} = 0.3$, and rings mark the best value.

forms (44.31 F1), supporting both explicit role decomposition and parameter sharing. Alternative interaction depths likewise underperform the full multi-step design (40.39 / 41.37 / 37.61 F1 for $N{=}4/8/10$), showing that moderate iterative refinement is optimal. Too few chunks limit refinement, while overly long interaction chains hurt optimization. Finally, training only the memory manager (45.34 F1) or only the fact extractor (28.30 F1) also degrades performance, especially the latter, confirming that both components benefit from joint RL training and that fact extraction is the more brittle of the two roles when left untrained.

Overall, the gains of our method arise from the combination of fair credit assignment, curriculum learning, multi-step memory construction, and shared extractor–manager co-learning. Additional ablations are reported in Appendix E.

### 4.4 More Analysis: Latency and Compression

**Latency.** Figure 4(a,b) compares F1 and inference latency before and after Memory-R2 training. Memory-R2 improves F1 while reducing latency for both Qwen2.5-3B and Qwen2.5-7B under the per-conversation measurement, moving both models toward a better quality–efficiency regime. The source of this latency reduction differs across scales. For Qwen2.5-3B, the gain is mainly driven by more concise generations: the trained policy emits fewer tokens per memory-construction turn. For Qwen2.5-7B, the gain comes not only from shorter generations, but also from a more stable generation-length distribution: the untrained policy occasionally produces overly long memory-management outputs, whereas Memory-R2 suppresses these unnecessary generations, reducing decoding work and making memory construction more stable. We provide a diagnostic breakdown of these scale-dependent mechanisms in Figure 12. These results suggest that better-trained memory policies can improve answer quality without incurring additional inference overhead, and can even reduce latency by making memory construction more concise and stable.

**Compression.** Figure 4(c,d) studies the effect of the compression penalty $\lambda_{\text{comp}}$. Across both Qwen2.5-3B and Qwen2.5-7B, $\lambda_{\text{comp}} = 0.3$ achieves the best F1 and BLEU-1, as highlighted by the yellow band. Smaller penalties may retain redundant or noisy memories, while overly strong compression can remove useful evidence. We therefore use $\lambda_{\text{comp}} = 0.3$ as the default setting.

## 5 Conclusion

In this paper, we present Memory-R2, a training framework for long-horizon memory-augmented LLM agents that addresses a fundamental challenge in multi-session reinforcement learning: fair credit assignment under diverging memory states. Our method, LoGo-GRPO, combines global trajectory-level optimization with local rerollouts from shared intermediate memory states, enabling fairer session-level comparisons while preserving end-to-end long-horizon learning. Beyond credit assignment, Memory-R2 jointly optimizes memory formation and memory evolution through a shared extractor–manager policy, formulates memory construction as a multi-step decision process over chunked sessions, and stabilizes training with a curriculum over session horizon. Experiments show that Memory-R2 consistently outperforms prior memory-agent baselines on LoCoMo and generalizes well across out-of-distribution benchmarks, model scales, and answer agents. These results suggest

that improving credit assignment is a key ingredient for training robust long-horizon memory agents, and we hope this work provides a useful foundation for future research on memory-centric RL for LLM agents.

## Acknowledgments

We thank the anonymous reviewers for their valuable feedback, and gratefully acknowledge the scientific support and HPC resources provided by the Karlsruhe Institute of Technology National High Performance Computing Center (NHR@KIT) under NHR project 22560 and 24767.

## References

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

[2] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

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

[4] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.09516.

[5] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent, 2025. URL https://arxiv.org/abs/2506.06326.

[6] Rui Li, Zeyu Zhang, Xiaohe Bo, Zihang Tian, Xu Chen, Quanyu Dai, Zhenhua Dong, and Ruiming Tang. Cam: A constructivist view of agentic memory for llm-based reading comprehension, 2025. URL https://arxiv.org/abs/2510.05520.

[7] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning, 2024. URL https://arxiv.org/abs/2404.02060.

[8] Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, Junpeng Ren, Zehao Lin, Jiahao Huo, Tianyi Chen, Kai Chen, Kehang Li, Zhiqiang Yin, Qingchen Yu, Bo Tang, Hongkang Yang, Zhi-Qin John Xu, and Feiyu Xiong. Memos: An operating system for memory-augmented generation (mag) in large language models, 2025. URL https://arxiv.org/abs/2505.22101.

[9] Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, Yuanxing Zhang, Zhuo Chen, Hangyu Guo, Shilong Li, Ziqiang Liu, Yong Shan, Yifan Song, Jiayi Tian, Wenhao Wu, Zhejian Zhou, Ruijie Zhu, Junlan Feng, Yang Gao, Shizhu He, Zhoujun Li, Tianyu Liu, Fanyu Meng, Wenbo Su, Yingshui Tan, Zili Wang, Jian Yang, Wei Ye, Bo Zheng, Wangchunshu Zhou, Wenhao Huang, Sujian Li, and Zhaoxiang Zhang. A comprehensive survey on long context language modeling, 2025. URL https://arxiv.org/abs/2503.17407.

[10] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents, 2024. URL https://arxiv.org/abs/2402.17753.

[11] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

[12] Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs, 2025. URL https://arxiv.org/abs/2504.13958.

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

[14] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

[15] Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. Membench: Towards more comprehensive evaluation on the memory of llm-based agents, 2025. URL https://arxiv.org/abs/2506.21605.

[16] Ziyu Wan, Yunxiang Li, Xiaoyu Wen, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. Rema: Learning to meta-think for llms with multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2503.09501.

[17] Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning. *arXiv preprint arXiv:2509.25911*, 2025.

[18] Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2505.16421.

[19] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

[20] Jing Xu, Arthur Szlam, and Jason Weston. Beyond goldfish memory: Long-term open-domain conversation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5180–5197, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.356. URL https://aclanthology.org/2022.acl-long.356/.

[21] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025. URL https://arxiv.org/abs/2502.12110.

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

[23] Sikuan Yan, Sicheng Dong, Haotong Wang, Ercong Nie, Yilun Liu, Jinhe Bi, Yingjie Xu, Susanna Schwarzmann, Riccardo Trivisonno, Volker Tresp, et al. Pyravid: Hierarchical multimodal memory for long-horizon video reasoning. *arXiv preprint arXiv:2605.17065*, 2026.

[24] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

[25] Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent, 2025. URL https://arxiv.org/abs/2507.02259.

[26] Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-memory: Tracing hierarchical memory for multi-agent systems, 2025. URL https://arxiv.org/abs/2506.07398.

[27] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory, 2023. URL https://arxiv.org/abs/2305.10250.

[28] Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

# A Additional Implementation Details

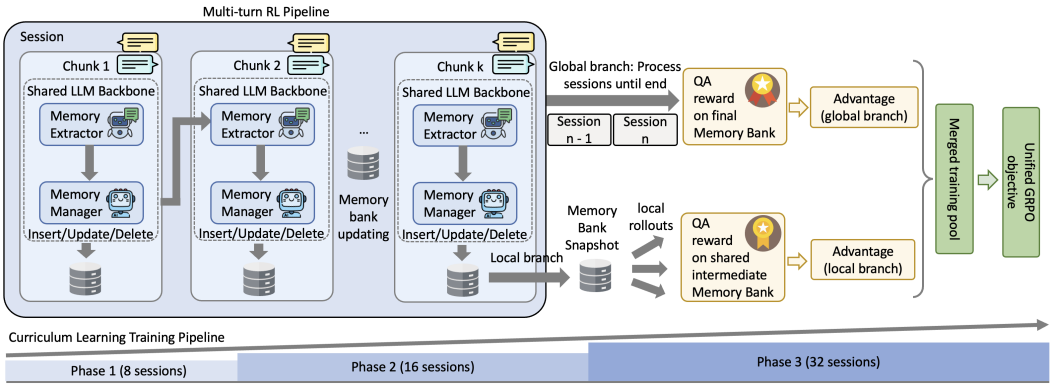



Figure 5: LoGo-GRPO training pipeline for memory manager. Memory bank construction via alternating extraction and management steps over chunked sessions. Global rollouts optimize end-to-end performance using rewards from the final memory, while local rerollouts from shared memory states provide low-bias credit assignment. Both signals are unified in a single GRPO-style objective, with curriculum learning enabling stable long-horizon training.

## A.1 RL Training for Answer Agent

*Training pipeline.* We fine-tune Qwen2.5-7B-Instruct as the answer agent using GRPO, implemented with the VERL framework [14]. Each training example is a single-turn QA prompt. Given a natural-language question $q$ and a constructed memory bank $\mathcal{M}$, we retrieve, for each speaker, the top-30 memory entries by text-embedding similarity to $q$ using a similarity threshold of $0.3$, and insert them into the QA template (Appendix B.3). The template instructs the model to reason step by step over the timestamped memories and output the final answer inside an `<answer>...</answer>` tag.

*Training data.* We construct the training set by running the full memory-construction pipeline (fact extraction, memory operations, and QA answering) on only two LoCoMo training conversations with GPT-4o, $T{=}0$. For each QA pair $(q, a^\star)$, we store (i) the rendered QA prompt and (ii) the answer extracted from the generated `<answer>` tag. Because some QA instances in the original data are noisy or weakly aligned with the available memory evidence, we retain only samples whose generated answer achieves token-level F1 $\geq 0.25$ against the gold answer $a^\star$. This filtering removes clearly problematic QA instances while preserving a sufficiently diverse training distribution. The remaining samples are randomly split into $90\%$ training and $10\%$ validation.

*Reward.* We use a judge-free rule-based reward

$$R(\hat{y}, a^\star) = \mathrm{F1}\big(\mathrm{extract}(\hat{y}),\, a^\star\big) \in [0, 1],$$

where $\mathrm{extract}(\cdot)$ returns the substring inside the first `<answer>...</answer>` tag. The F1 score is computed using the standard SQuAD-style token F1 after lowercasing, removing punctuation and articles (`a`, `an`, `the`), and whitespace tokenization. This keeps the training reward aligned with the answer-level metric used in evaluation.

*Optimization.* We train with GRPO using $n{=}8$ rollouts per prompt, sampling temperature $1.0$, the vLLM backend, GPU memory utilization $0.8$, and tensor parallelism $\mathrm{TP}{=}1$. We set the maximum prompt length to $12{,}288$ tokens and the maximum response length to $1{,}024$ tokens, with left truncation applied on the prompt side. The train batch size is $64$, the PPO mini-batch size is $16$, and the micro-batch size is $1$ per GPU. We use the standard GRPO advantage estimator and apply a token-level KL penalty in the actor loss with coefficient $0.001$, without adding KL to the reward. For runs with at least two GPUs, both parameters and optimizer states are kept on device; optimizer offloading is enabled only in the single-GPU setting. We train for $5$ epochs, and perform evaluation and checkpointing every $5$ optimizer steps.

### A.2 RL Training for Fact Extraction and Memory Management

We train the joint fact-extractor and memory-manager agent using a curriculum RL recipe with VERL framework [14]. A single `Qwen2.5-7B-Instruct` backbone is shared across both roles: the fact extractor produces atomic facts, while the memory manager predicts `INSERT`/`UPDATE`/`DELETE` operations. Parameter sharing is realized through alternating role-conditioned rollouts within each session chunk.

*Training Data.* We use LoCoMo with a conversation-level 2:1:7 train/validation/test split. The memory-construction policy is trained using only the two conversations in the training split, which contain 328 associated QA pairs in total. These QA pairs are used to compute downstream rewards for memory construction, while the held-out validation conversation is used for checkpoint selection and the remaining seven conversations are reserved for test evaluation.

*Optimization.* We optimize the policy with GRPO [2], using $N_{\text{rollout}}=16$ global trajectories per prompt and a local GRPO sampling fraction of $0.5$ with $N_{\text{local}}=4$ resampled turns to reduce gradient variance for memory operations. The actor is updated with $E_{\text{ppo}}=2$ epochs per batch, PPO mini-batch size $16$, micro-batch size $1$ per GPU, learning rate $\eta=2 \times 10^{-6}$, clipping ratio $\epsilon=0.2$, and entropy coefficient $0.001$. We use a KL penalty with coefficient $\beta_{\text{KL}}=10^{-3}$, without adding KL into the reward. We apply *turn-level* importance-ratio clipping (`clip_mode=turn`) and *turn-level* loss aggregation.

*Sequence and Rollout Budgets.* Each turn uses a prompt budget of $L_{\text{prompt}}=28{,}672$ tokens and a response budget of $L_{\text{resp}}=4{,}096$ tokens. We cap the number of memory turns per session at $T_{\max}=4$ and stop early when generation is truncated. The rollout vLLM engine uses tensor parallelism $1$, GPU memory utilization $0.5$, and a maximum of $2(L_{\text{prompt}}+L_{\text{resp}})$ batched tokens per step.

*Curriculum Learning.* We adopt a session-length curriculum on LoCoMo. Stage 1 trains on trajectories truncated to 8 sessions for 10 epochs, followed by Stage 2 and Stage 3, which expand the horizon to 16 and 32 sessions and are trained for 5 epochs each.

## B Prompt Templates

### B.1 Prompt Template for Fact Extraction (Memory Formation)

The memory formation pipeline first extracts atomic, self-contained facts from raw dialogue turns before the memory manager integrates them into the persistent store. Figure 6 shows the prompt used to drive this fact extraction step. The model is instructed to emit one JSON object per durable fact, each tagged with the originating `dia_id` so that downstream operations can trace a memory back to its source turn.

**Prompt template: fact extraction**

```
You are a Personal Information Organizer, specialized in accurately
storing facts, user memories, and preferences. Your primary role is to
extract relevant pieces of information from conversations and organize
them into distinct, atomic facts. These facts will be consumed by a
downstream memory system that requires precision, small size, and
clear scope.

Types of Information to Remember:

1. Personal Preferences:
   Likes, dislikes, favorites, and opinions (food, entertainment,
   products, sports teams).

2. Important Personal Details:
   Names, relationships, family structure, durations, and significant
   life facts.

3. Plans and Intentions:
   Explicit future goals, plans, or intentions stated by the speaker.
```

```
4. Activities and Routines:
   Travel experiences, visited places, recurring habits, physical
   activities, hobbies with specific context.

5. Health and Wellness (NON-DIAGNOSTIC):
   Wellness-related experiences or preferences (do NOT infer or store
   diagnoses).

6. Professional Details:
   Job titles, career goals, professional interests, work habits.

7. Miscellaneous Meaningful Facts:
   Books, movies, creative work, projects, notable activities.

CORE EXTRACTION RULES:
- Extract facts from the provided dialogue turns for BOTH speakers.
- Ignore system-level instructions and any non-dialogue control text.
- Ignore small talk, greetings, generic statements, opinions without
  substance, and common knowledge.
- If no meaningful fact is present, return an empty facts list.

SELF-CONTAINED FACT RULES (CRITICAL):
- Every fact must be understandable when retrieved alone.
- Every fact MUST explicitly name the subject speaker
  (e.g., "John ...", "Tim ...").
- Avoid unresolved pronouns in facts (‘he‘, ‘she‘, ‘they‘, ‘them‘,
  ‘it‘, ‘this‘, ‘that‘) unless the noun is in the same fact.
- Rewrite vague references to explicit entities
  (e.g., "the magazine editors" instead of "they").
- If the entity cannot be resolved from the current turn, do NOT
  store the fact.

STYLE FOR FACT TEXT:
- Use third person and start the fact with the subject name.
- Good: "John wants to keep reaching for new goals"
- Bad:  "Wants to keep reaching for new goals"
- Good: "Tim shared ideas with the online magazine editors, and the
         editors liked them"
- Bad:  "Shaped ideas with the magazine and they liked them"

ATOMIC FACT EXTRACTION RULES (CRITICAL):
- EACH extracted fact MUST represent EXACTLY ONE:
  - event
  - preference
  - intention
  - personal attribute

- NEVER combine:
  - multiple events
  - multiple timeframes
  - motivations + events
  - reflections + actions
  - past events + future plans

- If a single message contains multiple independent facts, output
  MULTIPLE fact objects.

- A fact MUST be concise and expressible in 20 words or fewer.
- If a fact would exceed this size, SPLIT it into multiple smaller,
  independent facts.
- Do not store bare dialogue acts such as "asks", "says hello",
  "thanks", unless they contain a durable personal fact.
```

```
INTENT VS EVENT RULE:
- Past events (what happened) and intentions or goals (what the
  speaker wants or plans) MUST ALWAYS be extracted as SEPARATE facts.

TEMPORAL INFORMATION RULE:
- If a fact includes temporal information (dates, durations, relative
  times), always include it explicitly.
- Prefer absolute time when resolvable from context
  (e.g., "in 2018" not "five years ago" if the session year is known).
- Do NOT drop temporal details - they are often critical
  for answering questions correctly.

Important Notes on dia_id:
- ‘dia_id‘ uniquely identifies the dialogue turn.
- EACH fact must include the ‘dia_id‘ of its source message.
- Do NOT attach multiple dia_ids to a single fact.

Here are some few-shot examples:

Input: [{"speaker": "John", "text": "Hi, how are you?",
         "dia_id": "D1:1"}]
Output: {"facts": []}

Input: [{"speaker": "John", "text": "There are branches in trees.",
         "dia_id": "D2:3"}]
Output: {"facts": []}

Input:
[
  {"speaker": "Maria", "text": "What’s your favorite sport?",
   "dia_id": "D3:1"},
  {"speaker": "John",  "text": "I love playing basketball with
                                friends.", "dia_id": "D3:2"}
]
Output:
{"facts": [{"speaker": "John", "dia_id": "D3:2",
             "fact": "John loves playing basketball with friends"}]}

Input:
[
  {"speaker": "Maria", "text": "What did you do yesterday?",
   "dia_id": "D3:5"},
  {"speaker": "John",  "text": "Yesterday, I had a meeting at 3pm.
                                We discussed a new project.",
   "dia_id": "D3:6"}
]
Output:
{"facts": [{"speaker": "John", "dia_id": "D3:6",
             "fact": "John had a meeting at 3pm yesterday about a
                      new project"}]}

Input: [{"speaker": "John", "text": "I am a software engineer.",
         "dia_id": "D4:2"}]
Output:
{"facts": [{"speaker": "John", "dia_id": "D4:2",
             "fact": "John is a software engineer"}]}

Input: [{"speaker": "John", "text": "My favorite movies are
         Inception and Interstellar.", "dia_id": "D4:3"}]
Output:
{"facts": [{"speaker": "John", "dia_id": "D4:3",
             "fact": "John’s favorite movies are Inception and
                      Interstellar"}]}
```

```
Input: [{"speaker": "John", "text": "I attended an LGBTQ workshop
         last Friday and it inspired me to pursue counseling.",
         "dia_id": "D5:1"}]
Output:
{"facts": [
  {"speaker": "John", "dia_id": "D5:1",
   "fact": "John attended an LGBTQ workshop last Friday"},
  {"speaker": "John", "dia_id": "D5:1",
   "fact": "John feels inspired to pursue a counseling career"}
]}

Input:
[
  {"speaker": "Maria", "text": "That sounds wonderful!",
   "dia_id": "D6:3"},
  {"speaker": "John",  "text": "Thanks! I really appreciate your
                                support.", "dia_id": "D6:4"}
]
Output: {"facts": []}

Input: [{"speaker": "John", "text": "So what do you think about
         that?", "dia_id": "D7:2"}]
Output: {"facts": []}

Return the facts in JSON format exactly as shown above.

Remember the following:
- If multiple statements describe the SAME EVENT at the SAME TIME
  and PLACE within the SAME dialogue turn, they MAY be merged into
  a SINGLE fact.
- If they differ by time, place, motivation, outcome, or reflection
  -> extract SEPARATE facts.
- Do NOT output standalone facts that depend on another fact for
  context unless merged into a complete event.
- Do not return anything from the custom few-shot example prompts
  provided above.
- If no relevant facts are found, return {"facts": []}.
- The response MUST be valid JSON with a single top-level key:
  "facts".
```

**Figure 6:** Prompt template for atomic fact extraction. Each extracted fact is a self-contained, third-person statement tagged with the originating `dia_id`, and is then passed to the memory manager (Figure 7) for integration into the persistent memory store.

### B.2 Prompt Template for the Memory Manager (Memory Evolution)

The memory manager is the second stage of our pipeline. It takes the atomic facts produced by the fact-extraction prompt (Section B.1) together with the current memory store, and decides for each new fact whether to insert, update, delete, or take no operation on the store. Compared with a naive memory writer, our prompt enforces three properties that proved important in practice: (i) *atomicity*, so that each memory entry encodes exactly one fact; (ii) *monotonicity*, so that prior factual claims are never silently dropped during an UPDATE; and (iii) *noise tolerance* over the embedding-based candidates, which can be topically unrelated. Figure 7 shows the full prompt.

**Prompt template: memory bank management**

```
You are a smart memory manager which controls the memory of a system.
You can perform four operations: (1) insert into the memory, (2) update
the memory, (3) delete from the memory, and (4) no change.

Your primary goal is to preserve accurate factual evidence over time.
```

```
Memory updates must be SAFE, NON-DESTRUCTIVE, and FACT-PRESERVING.

CONTENT LENGTH RULE (enforced before all other rules):
- Every ‘content‘ field you write (INSERT or UPDATE) MUST be at most
  20 words.
- ONE fact per memory item -- never combine multiple independent
  facts into one entry.
- If you cannot express a fact in 20 words, write the most essential
  part only.

INPUT FORMAT: The input contains two sections:
- "memories": a flat list of existing memory entries (each appears
  exactly once, identified by memory_id).
- "facts": new facts to process, each with a "related_memory_ids"
  list -- IDs pointing into "memories" as candidates for UPDATE or
  DELETE for that specific fact.

To find candidates for a fact: look up its "related_memory_ids" in
the "memories" list by memory_id.
WARNING: "related_memory_ids" are retrieved by embedding similarity
and MAY CONTAIN NOISE -- some IDs may point to memories that are
topically unrelated to the fact. Do NOT blindly UPDATE or DELETE
just because an ID appears in "related_memory_ids". Always verify
the memory content actually refers to the same entity and topic
before acting on it. If no entry is genuinely relevant, treat
"related_memory_ids" as empty.

For each new fact, decide whether to:
- INSERT: The fact is new and not captured by any entry in its
  "related_memory_ids" (or list is empty or has no genuinely relevant
  entry).
- UPDATE: The fact refers to the SAME entity or event as a
  "related_memory_ids" entry and enriches, refines, or corrects it
  WITHOUT removing prior factual information.
- DELETE: The fact explicitly proves a "related_memory_ids" entry is
  false or invalid (not merely outdated).
- NO OPERATION: The fact is already captured by a "related_memory_ids"
  entry, redundant, irrelevant, or insignificant.

There are specific guidelines to select which operation to perform:

1. INSERT: If the fact contains new information not captured in its
   ‘related_memory_ids‘, then you have to add it.
   - Assign ‘speaker‘ as who the fact is ABOUT.
   - Assign ‘content‘ as a concise summary in third person.
   - Keep tense faithful to the source fact (past events may stay
     past tense).
   - Do NOT assign ‘memory_id‘ for INSERT operations; the system
     will auto-generate it.
   - Always include the ‘dia_id‘ with each inserted fact to ensure
     the memory is accurately linked to the correct dialogue.
   - ‘content‘ must be SELF-CONTAINED and include the subject name
     explicitly (e.g., "John ...", "Tim ...").
   - ‘content‘ must avoid vague pronouns unless the referenced noun
     appears in the same sentence.
   - If a fact cannot be made self-contained without guessing, skip
     it (NO OPERATION).

   Example:
   - Input: {
       "memories": [{"memory_id": "a32b32c1", "speaker": "John",
         "content": "John works as a software engineer",
         "session_time": "6:59 pm on 26 August, 2023",
```

```
        "dia_ids": ["D1:4"]}],
      "facts": [
        {"speaker": "John", "dia_id": "D3:6",
         "fact": "John had a meeting at 3pm",
         "related_memory_ids": ["a32b32c1"]},
        {"speaker": "John", "dia_id": "D3:6",
         "fact": "John discussed a new project",
         "related_memory_ids": ["a32b32c1"]}
      ]
    }
  - Operations:
    {"operations": [
       {"operation": "INSERT", "speaker": "John",
        "content": "John had a meeting at 3pm about a new project",
        "dia_id": "D3:6"}
    ]}

ATOMICITY RULE:
- Each memory item MUST represent a single fact or event.
- Do NOT merge multiple independent facts into one memory item.
- If a new fact represents a genuinely different event, topic, or
  attribute -> INSERT instead of UPDATE.
- Exception: if the new fact is a progression or status change of
  the SAME entity's story (e.g., "exploring a job" -> "accepted the
  job"), UPDATE the existing entry rather than inserting a duplicate.

2. UPDATE: Use UPDATE only when the new fact clearly refers to the
   SAME entity or event as an entry in its 'related_memory_ids' and
   ADDS detail, refinement, or correction WITHOUT removing prior
   facts.
   - NEVER remove existing factual information during an UPDATE.
   - If the new fact is more specific, merge it with the existing
     content.
   - If both convey the same meaning, keep the more informative
     version.
   - If the new fact introduces a completely unrelated event, goal,
     or topic -> INSERT instead.
   - If the new fact is a later development or confirmation of the
     SAME entity's ongoing story (e.g., plan -> outcome, exploring
     -> confirmed) -> UPDATE even if the time is different.
   - Please keep in mind while updating you have to use the same ID.
   - Always include the 'dia_id' with each updated fact to ensure
     the memory is accurately linked to the correct dialogue.
   - Please note to return the IDs in the output from the input IDs
     only and do not generate any new ID.

   Example (refinement -- same entity, added detail):
   - Input: {
       "memories": [{"memory_id": "a0299e69", "speaker": "Emily",
         "content": "Likes to play cricket",
         "session_time": "2:04 pm on 3 September, 2021",
         "dia_ids": ["D5:2"]}],
       "facts": [{"speaker": "Emily",
         "fact": "Emily loves to play cricket with friends",
         "dia_id": "D5:4", "related_memory_ids": ["a0299e69"]}]
     }
   - Operations:
     {"operations": [{"operation": "UPDATE",
       "memory_id": "a0299e69",
       "content": "Emily loves to play cricket with friends",
       "dia_id": "D5:4"}]}

   Example (cross-session fact evolution -- status changed from
```

```
   exploring to confirmed):
   - Input: {
       "memories": [{"memory_id": "f3a91b44", "speaker": "Sarah",
         "content": "Sarah is exploring a job opportunity at a tech
                     company in Seattle",
         "session_time": "3:00 pm on 10 March, 2022",
         "dia_ids": ["D2:5"]}],
       "facts": [{"speaker": "Sarah", "dia_id": "D3:8",
         "fact": "Sarah accepted a senior software engineer role at
                  TechCorp in Seattle",
         "related_memory_ids": ["f3a91b44"]}]
     }
   - Operations:
     {"operations": [{"operation": "UPDATE",
       "memory_id": "f3a91b44",
       "content": "Sarah accepted a senior software engineer role at
                   TechCorp in Seattle",
       "dia_id": "D3:8"}]}
   Explanation: Same entity (Sarah's job in Seattle), status evolved
   from "exploring" to "accepted" -- UPDATE the same memory entry.
   Do NOT insert a duplicate about the Seattle job.

INVALID UPDATE EXAMPLE (DO NOT DO THIS):
- Memory content: "Sarah traveled to Paris and Rome on her European
  trip"
- Wrong UPDATE content: "Sarah traveled to Paris" -- removes Rome,
  destroys stored fact
- Correct action: NO OPERATION (no new information) or INSERT a
  separate fact about Rome if it was new.

3. DELETE: Use DELETE only when a new fact explicitly contradicts
   and invalidates an entry in its 'related_memory_ids'.
   - Do NOT delete memories just because they are old or less
     relevant.
   - Please note to return the IDs in the output from the input IDs
     only and do not generate any new ID.

   Example:
   - Input: {
       "memories": [{"memory_id": "6v0k193d", "speaker": "Samy",
         "content": "I went to Paris last summer",
         "session_time": "8:04 am on 3 February, 2009",
         "dia_ids": ["D6:5"]}],
       "facts": [{"speaker": "Samy",
         "fact": "Samy never went to Paris", "dia_id": "D7:1",
         "related_memory_ids": ["6v0k193d"]}]
     }
   - Operations:
     {"operations": [{"operation": "DELETE",
       "memory_id": "6v0k193d"}]}

4. NO OPERATION: If the new fact is already captured by an entry in
   its 'related_memory_ids' -- even if worded differently -- do NOT
   insert a new entry.
   Before deciding INSERT, look up the fact's 'related_memory_ids'
   in "memories" and check for semantic overlap: same person, same
   topic, same meaning.
   If a semantically equivalent memory already exists -> NO
   OPERATION (not INSERT).
   If 'related_memory_ids' is empty -> INSERT is safe.

   Example (exact match):
   - Input: {
```

```
      "memories": [{"memory_id": "9b3c82e0", "speaker": "Sofia",
        "content": "Sofia loves cheese pizza",
        "session_time": "11:10 am on 18 March, 2020",
        "dia_ids": ["D8:3"]}],
      "facts": [{"speaker": "Sofia",
        "fact": "Sofia loves cheese pizza", "dia_id": "D8:10",
        "related_memory_ids": ["9b3c82e0"]}]
    }
  - Operations: {"operations": []}

  Example (semantic match -- paraphrase is NOT a new fact):
  - Input: {
      "memories": [{"memory_id": "c4184b6a", "speaker": "Alex",
        "content": "Alex is training for a marathon with a local
                    running club",
        "session_time": "9:00 am on 5 January, 2022",
        "dia_ids": ["D1:3"]}],
      "facts": [{"speaker": "Alex", "dia_id": "D1:9",
        "fact": "Alex is preparing for a marathon competition with
                 teammates",
        "related_memory_ids": ["c4184b6a"]}]
    }
  - Operations: {"operations": []}
  Explanation: The new fact describes the same activity already in
  "memories". It is a paraphrase, not new information -> NO
  OPERATION.

  Example (same memory shared by two facts):
  - Input: {
      "memories": [{"memory_id": "a32b32c1", "speaker": "John",
        "content": "John works as a software engineer",
        "session_time": "6:59 pm on 26 August, 2023",
        "dia_ids": ["D1:4"]}],
      "facts": [
        {"speaker": "John", "dia_id": "D3:6",
         "fact": "John changed careers to become a teacher",
         "related_memory_ids": ["a32b32c1"]},
        {"speaker": "John", "dia_id": "D3:7",
         "fact": "John no longer works in tech",
         "related_memory_ids": ["a32b32c1"]}
      ]
    }
  - Operations:
    {"operations": [{"operation": "UPDATE",
      "memory_id": "a32b32c1",
      "content": "John became a teacher, left software engineering",
      "dia_id": "D3:6"}]}
  Explanation: Both facts point to the same memory. Produce ONE
  UPDATE -- do not UPDATE the same memory_id twice.


DECISION ORDER (follow this sequence for EVERY new fact):
1. Does the new fact explicitly contradict a memory entry in its
   'related_memory_ids'? -> DELETE the contradicted entry.
2. Does a semantically equivalent entry already exist in
   'related_memory_ids' (same person, same topic, same meaning)?
   -> NO OPERATION. Stop.
3. Does an entry in 'related_memory_ids' exist and the new fact
   refines, progresses, or confirms the same entity's story?
   -> UPDATE. Stop.
4. No matching entry found -> INSERT.

Follow the instruction mentioned below:
```

```
- Memory is MONOTONIC: factual information must never be lost
  unless explicitly contradicted.
- UPDATE operations MUST preserve all previously stored factual
  claims. An UPDATE must preserve all existing factual claims, but
  may rephrase them concisely within size limits.
- Do not return anything from the custom few shot prompts provided
  above.
- You should return the operations in only JSON format as shown
  above.
- Do not store small talk, greetings, generic questions. Only store
  information that conveys meaningful or significant facts.
- If there is an insert, must include speaker field. must not
  include memory_id, session_time fields because the system
  auto-generates it.
- If there is a deletion or update, must use exact memory_id from
  the "memories" list (looked up via the fact's
  `related_memory_ids`). Do not invent or guess memory IDs.
- If two facts share a related_memory_ids entry, produce at most
  ONE operation on that memory_id -- do not UPDATE or DELETE the
  same memory_id twice.
- Before outputting operations, run a strict self-check:
  1) Every `content` is understandable alone.
  2) Every `content` explicitly names the subject speaker.
  3) No unresolved vague pronouns remain.
  4) No entry is only a conversational act without durable fact
     value.

Do not return anything except the JSON format.
```

**Figure 7:** Prompt template for the memory manager. The model receives the current memory store and a batch of atomic facts (output of the fact-extraction stage, Appendix B.1) and emits a JSON list of `INSERT`/`UPDATE`/`DELETE` edits. A fixed decision order, an atomicity constraint, and explicit non-destructive update semantics together prevent the common failure modes of LLM-based memory writers, namely fact loss, duplicated entries, and noisy retrieval-driven overwrites.

### B.3 Prompt Template for Answer Agent (Memory Usage)

The answer agent is the final stage of our pipeline. It takes a user question together with the memory entries written by the memory manager (Section B.2) and produces a concise, evidence-grounded answer. Two design choices are worth noting. First, each memory entry carries a timestamp, and questions in our benchmark frequently involve relative time expressions (*"last year"*, *"two months ago"*); the prompt therefore instructs the model to resolve such expressions to absolute dates using the timestamp of the supporting memory, rather than the question's own utterance time. Second, the two speakers' memories are presented in separate blocks labeled by speaker name, which prevents the model from confusing third-party names mentioned within a memory with the speaker who owns that memory. The model is required to terminate its response with an `<answer>...</answer>` span, which is then extracted and scored against the gold answer using SQuAD-style token-level F1. Figure 8 shows the full prompt.

**Prompt template: memory-grounded question answering**

```
You are an intelligent memory assistant tasked with retrieving accurate
information from conversation memories.

# CONTEXT:
You have access to memories from two speakers in a conversation. These
memories contain timestamped information that may be relevant to answering
the question.

# INSTRUCTIONS:
1. Carefully analyze all provided memories from both speakers
```

```
2. Pay special attention to the timestamps to determine the answer
3. If the question asks about a specific event or fact, look for direct
   evidence in the memories
4. If the memories contain contradictory information, prioritize the most
   recent memory
5. If there is a question about time references (like "last year", "two
   months ago", etc.), calculate the actual date based on the memory
   timestamp. For example, if a memory from 4 May 2022 mentions "went to
   India last year," then the trip occurred in 2021.
6. Always convert relative time references to specific dates, months, or
   years. For example, convert "last year" to "2022" or "two months ago"
   to "March 2023" based on the memory timestamp. Ignore the reference
   while answering the question.
7. Focus only on the content of the memories from both speakers. Do not
   confuse character names mentioned in memories with the actual users
   who created those memories.
8. If memories are insufficient and the question is about a general world
   fact, you may use reliable general world knowledge.
9. Keep the final answer concise, typically no more than 10-12 words; do
   not omit essential entities or dates.

# APPROACH (Think step by step):
1. First, examine all memories that contain information related to the
   question
2. Examine the timestamps and content of these memories carefully
3. Look for explicit mentions of dates, times, locations, or events that
   answer the question
4. If the answer requires calculation (e.g., converting relative time
   references), show your work
5. Formulate a precise, concise answer based on the evidence in the
   memories, using general world knowledge only if memories are
   insufficient
6. Double-check that your answer directly addresses the question asked
7. Ensure your final answer is specific and avoids vague time references
8. Output the final answer only in this format, with no extra text:
   <answer>YOUR_FINAL_ANSWER</answer>

Memories for user speaker_1:
speaker_1_memories

Memories for user speaker_2:
speaker_2_memories

Question: question

Answer step by step, and output the final answer in this format, with no
extra text: <answer>YOUR_FINAL_ANSWER</answer>
```

**Figure 8:** Prompt template used for memory-based question answering. Double-braced tokens denote runtime placeholders. Model outputs are parsed from the `<answer>...</answer>` span and scored with SQuAD-style token F1.

# C Evaluation Metrics

## C.1 LLM-as-a-Judge

In addition to F1, B1, we report an LLM-as-a-Judge (J) score that captures semantic equivalence between the generated answer and the gold answer, mitigating the well-known brittleness of token-level metrics on free-form generations. We follow the judging protocol established by prior work on memory-augmented dialogue agents [1, 22], and use `gpt-4o-mini` as the judge model for all reported J scores. The judge receives the question, the gold answer, and the generated answer, and is asked to return a binary Correct/Wrong label, with explicit instructions to be lenient toward formatting

differences (e.g., *"May 7"* versus *"7 May"*) and toward the generated answer being more verbose than the gold. The model is required to emit its decision as a JSON object with a single `label` field, which we parse for downstream aggregation. We chose `gpt-4o-mini` as a deliberate cost-quality trade-off: it is strong enough to reliably handle the lenient string-matching judgments required here, while being cheap enough to run across the full evaluation set without distorting our compute budget. Figure 9 shows the full prompt.

**Prompt template: LLM-as-a-Judge**

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

The gold answer is usually concise; the generated answer may be longer.
Be generous: if the generated answer touches on the same topic/date as
the gold, count CORRECT. Different formats for the same date (e.g. "May 7"
vs "7 May") are CORRECT.

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

First give a one-sentence reasoning, then finish with CORRECT or WRONG. Do
NOT include both.

Return a JSON object with key "label" whose value is exactly "CORRECT"
or "WRONG".
```

**Figure 9:** Prompt template for the LLM-as-a-Judge evaluator, instantiated with `gpt-4o-mini`. Single-braced tokens (`{question}`, `{gold_answer}`, `{generated_answer}`) are runtime placeholders.

### C.2 Memory-Failure Rate (M-Fail)

To diagnose memory-bank construction quality beyond answer-level metrics, we define **M-Fail** as the fraction of gold evidence that is missing from the memory bank. For each question $q$, let $\mathcal{E}_q \subseteq \mathcal{D}$ be the set of gold evidence dialogue-turn IDs required to answer $q$, and let $\mathcal{M} \subseteq \mathcal{D}$ denote the set of dialogue-turn IDs currently stored in the memory bank. We compute

$$\text{M-Fail} = \frac{\sum_q |\mathcal{E}_q \setminus \mathcal{M}|}{\sum_q |\mathcal{E}_q|}. \tag{18}$$

Lower is better; $\text{M-Fail} = 0$ means all required evidence is present in memory. This metric isolates *memory-construction* errors (evidence never stored) from downstream retrieval or answer-generation effects.

## D Limitations and Future Work

Our study focuses on long-horizon text-only multi-session dialogue, where memory is constructed from conversational turns and evaluated through downstream QA. Extending the same training paradigm to multimodal settings, such as image-grounded dialogue, video memory, or embodied interaction, remains unexplored. While the proposed credit-assignment framework is in principle agnostic to the modality of memory content, its practical effectiveness in richer environments still requires further investigation.

This work has both potential positive and negative societal impacts. On the positive side, Memory-R2 improves the training of long-horizon memory-augmented LLM agents, which may help make current memory systems more reliable, consistent, and practically useful. More effective memory construction can benefit applications such as personalized assistants, long-term educational support, and tools that require maintaining user context across multiple interactions. More broadly, improving memory quality may reduce failure modes caused by forgetting or inconsistent recall, which are common limitations of current LLM-based agents.

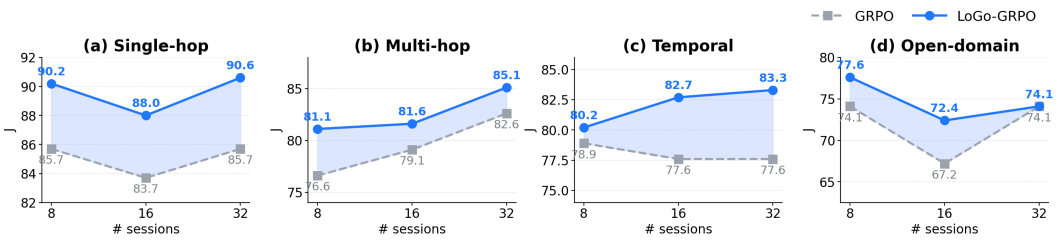



Figure 10: **LoGo-GRPO consistently outperforms GRPO across all question types and curriculum stages.** Judge accuracy (**J**) at curriculum stages $8 \rightarrow 16 \rightarrow 32$ sessions, broken down by question type: **(a)** Single-hop, **(b)** Multi-hop, **(c)** Temporal, **(d)** Open-domain. LoGo-GRPO (blue) dominates GRPO (gray) at every stage and on every category, with the shaded band visualizing the gap, indicating that local rerollouts constantly mitigate credit-assignment bias. Accuracy continues to improve with the session range on long-horizon question types (Multi-hop, Temporal), while remaining stable on the others.

Table 3: Main results per question category, averaged over 3 independent runs with different random seeds. We report mean $\pm$ standard deviation. The J score is computed by an LLM-as-a-Judge with `gpt-4o-mini` as the judge model (Section C.1).

| **Model** | **Category** | **F1** $\uparrow$ | **B1** $\uparrow$ | **J** $\uparrow$ |
|---|---|---|---|---|
| Memory-R2 | Single-hop | $54.06 \pm 2.12$ | $48.73 \pm 2.26$ | $86.80 \pm 0.48$ |
| | Multi-hop | $38.41 \pm 0.91$ | $30.90 \pm 0.46$ | $80.93 \pm 1.52$ |
| | Temporal | $59.65 \pm 1.00$ | $50.05 \pm 1.10$ | $69.90 \pm 1.34$ |
| | Open-domain | $20.76 \pm 1.06$ | $16.78 \pm 0.66$ | $67.53 \pm 2.04$ |
| | Overall | $50.60 \pm 1.34$ | $44.01 \pm 1.37$ | $80.99 \pm 0.28$ |
| Memory-R2 (OSS) | Single-hop | $50.98 \pm 0.45$ | $45.79 \pm 0.48$ | $90.58 \pm 1.56$ |
| | Multi-hop | $36.37 \pm 1.63$ | $29.79 \pm 1.91$ | $85.07 \pm 1.41$ |
| | Temporal | $62.48 \pm 0.52$ | $55.33 \pm 0.42$ | $83.33 \pm 1.49$ |
| | Open-domain | $30.22 \pm 0.71$ | $24.50 \pm 1.11$ | $74.14 \pm 0.00$ |
| | Overall | $49.67 \pm 0.60$ | $43.77 \pm 0.59$ | $87.10 \pm 1.43$ |

At the same time, stronger persistent memory systems also raise important risks. In real-world deployments, memory mechanisms may store sensitive personal information over extended interactions, creating privacy and security concerns if such information is retained unnecessarily, retrieved inappropriately, or exposed to unauthorized parties. As memory becomes more accurate and durable, these risks may become more consequential. In addition, errors in stored memory may persist across sessions and affect later interactions. We therefore view privacy-preserving memory design, secure storage and access control, and safer memory-update mechanisms as important directions for future work.

# E Additional Ablations and Generalization Results

This section provides additional evidence for the effectiveness and stability of Memory-R2.

Figure 10 breaks down LoGo-GRPO versus standard GRPO across question types and curriculum stages, showing consistent gains from local rerollouts.

Figure 11 compares the proposed $8 \rightarrow 16 \rightarrow 32$-session curriculum with direct 32-session training under equal compute, demonstrating that curriculum learning stabilizes validation performance, memory growth, memory quality, and reward signals.

Figure 12 further diagnoses the latency improvements of Memory-R2, showing that the 3B model reduces latency mainly through more concise generations, while the 7B model reduces latency by suppressing long output-length tails and improving batched decoding efficiency.

Table 3 reports mean and standard deviation over three independent runs, confirming that the improvements of Memory-R2 are robust across random seeds.

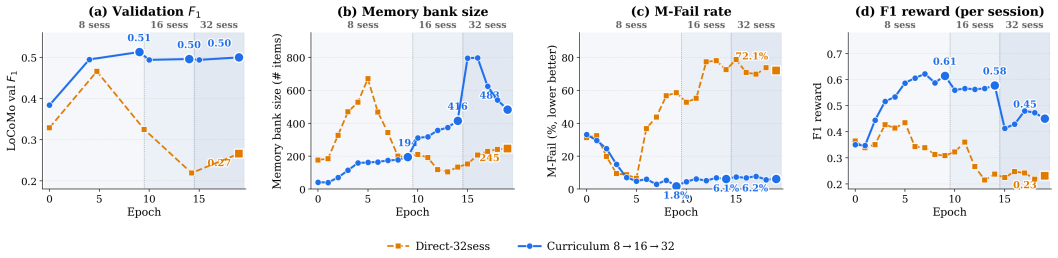



Figure 11: **Curriculum learning is essential for stable long-horizon training.** Training dynamics of curriculum $8 \rightarrow 16 \rightarrow 32$ sessions (blue) vs. direct 32-session training (orange) under equal compute. The x-axis is the cumulative epochs within the curriculum; direct-32sess is linearly stretched onto the same axis for fair comparison. **(a)** Validation $F_1$ on LoCoMo: the curriculum stabilizes around $0.50$, while direct-32sess collapses from a peak of $0.47$ down to $0.27$. **(b)** Memory bank size grows steadily under the curriculum ($194 \rightarrow 416 \rightarrow 483$ items at stage ends), reflecting healthy accumulation; the direct run instead inflates and then truncates erratically. **(c)** Memory failure rate (M-Fail) stays below $7\%$ throughout the curriculum, but explodes to over $70\%$ for direct-32sess once training enters the long-horizon regime. **(d)** Per-session $F_1$ reward remains high across the curriculum ($0.61/0.58/0.45$ at stage ends), while the direct run degrades to $0.23$. Together, these dynamics show that early errors in the long-horizon setting propagate across sessions and corrupt the memory bank, and that the curriculum allows the policy to acquire reliable short-horizon memory behaviors before tackling longer trajectories.

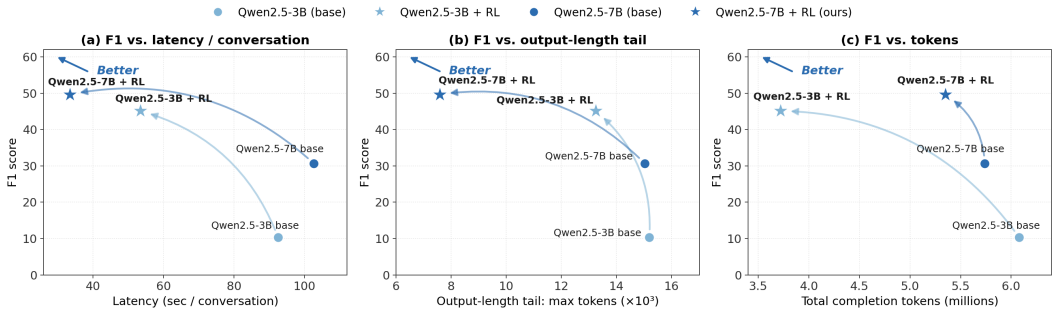



Figure 12: Latency-mechanism diagnostics. (a) Memory-R2 improves F1 while reducing per-conversation latency. (b) Memory-R2 shortens the output-length tail, indicating fewer overly long generations. (c) Memory-R2 reduces total completion tokens, especially for Qwen2.5-3B, indicating a more concise memory-construction policy. Together, these diagnostics suggest that the latency gains come from more controlled and less redundant generation.