

# Memory-R1: Enhancing Large Language Model Agents to Manage and Utilize Memories via Reinforcement Learning

Sikuan Yan<sup>\*1,2</sup>, Xiufeng Yang<sup>\*3</sup>, Zuchao Huang<sup>1</sup>, Ercong Nie<sup>1,2</sup>,  
Zifeng Ding<sup>2,4</sup>, Zonggen Li<sup>5</sup>, Xiaowen Ma<sup>1</sup>, Jinhe Bi<sup>1</sup>, Kristian Kersting<sup>6</sup>,  
Jeff Z. Pan<sup>7</sup>, Hinrich Schuetze<sup>1,2</sup>, Volker Tresp<sup>1,2</sup>, Yunpu Ma<sup>†1,2</sup>

<sup>1</sup>Ludwig Maximilian University of Munich, <sup>2</sup>Munich Center for Machine Learning,  
<sup>3</sup>Technical University of Munich, <sup>4</sup>University of Cambridge, <sup>5</sup>University of Hong Kong,  
<sup>6</sup>Technical University of Darmstadt, <sup>7</sup>University of Edinburgh

s.yan@campus.lmu.de, cognitive.yunpu@gmail.com

## Abstract

Large Language Models (LLMs) have demonstrated impressive capabilities across a wide range of NLP tasks, but they remain fundamentally *stateless*, constrained by limited context windows that hinder long-horizon reasoning. Recent efforts to address this limitation often augment LLMs with an external memory bank, yet most existing pipelines are static and heuristic-driven, lacking a learned mechanism for deciding what to store, update, or retrieve. We present Memory-R1, a reinforcement learning (RL) framework that equips LLMs with the ability to actively manage and utilize external memory through two specialized agents: a *Memory Manager* that learns structured operations, including ADD, UPDATE, DELETE, and NOOP; and an *Answer Agent* that pre-selects and reasons over relevant entries. Both agents are fine-tuned with outcome-driven RL (PPO and GRPO), enabling adaptive memory management with minimal supervision. With only 152 training QA pairs, Memory-R1 outperforms strong baselines and generalizes across diverse question types, three benchmarks (LoCoMo, MSC, LongMemEval), and multiple model scales (3B–14B).

## 1 Introduction

Large Language Models (LLMs) have shown remarkable ability in understanding and generating natural language, making them central to recent advances in AI (OpenAI et al., 2024; Qwen et al., 2025). Yet, they remain fundamentally *stateless* (Yu et al., 2025; Fan et al., 2025; Goodyear et al., 2025): their memory is bounded by a finite context window and any information that falls outside this window is forgotten, preventing them from maintaining knowledge across long conversations or evolving tasks (Wang et al., 2024; Fei et al., 2023).

One early effort is the Tensor Brain framework, which uses a bilayer tensor network with index and representation layers to model episodic, semantic, and working memory (Tresp et al., 2023). Recent studies augment LLMs with explicit external memory modules (Zhang et al., 2024), most of which adopt the retrieval-augmented generation (RAG) paradigm (Pan et al., 2025; Salama et al., 2025), appending retrieved memory entries to the model’s input prompt. While this extends access to past information, it also creates a fundamental retrieval challenge: heuristics may return too few entries, omitting crucial context, or too many, flooding the model with irrelevant information and degrading performance (Liu et al., 2023). In this paradigm, retrieved memories are passed to the LLM without meaningful filtering or prioritization, forcing the model to reason over both relevant and irrelevant content, which makes it prone to distraction by noise. Humans, by contrast, retrieve broadly but then filter, integrating only the most useful pieces to maintain coherent, evolving knowledge.

Equally important is the challenge of *memory management*: deciding what to remember, update, or discard. Some systems (Packer et al., 2023; Modarressi et al., 2024; Xiong et al., 2025) adopt CRUD-style operations, namely create, read, update, and delete, which are adapted from databases (Martin, 1983). A more recent work (AIOS Foundation, 2024) augments this paradigm with a search operator, while Mem0 (Chhikara et al., 2025) investigates the operator set {ADD, UPDATE, DELETE, NOOP}. We adopt this setting, as it provides a minimal yet expressive framework for modeling memory dynamics. Existing approaches mainly rely on vanilla LLMs to choose operations from in-context instructions without any learning signal tied to correctness (Packer et al., 2023; Chhikara et al., 2025). Even simple cases can fail. Figure 1, a simplified example drawn from a LoCoMo conversation (Ma-

\*Equal contribution.

†Corresponding author.

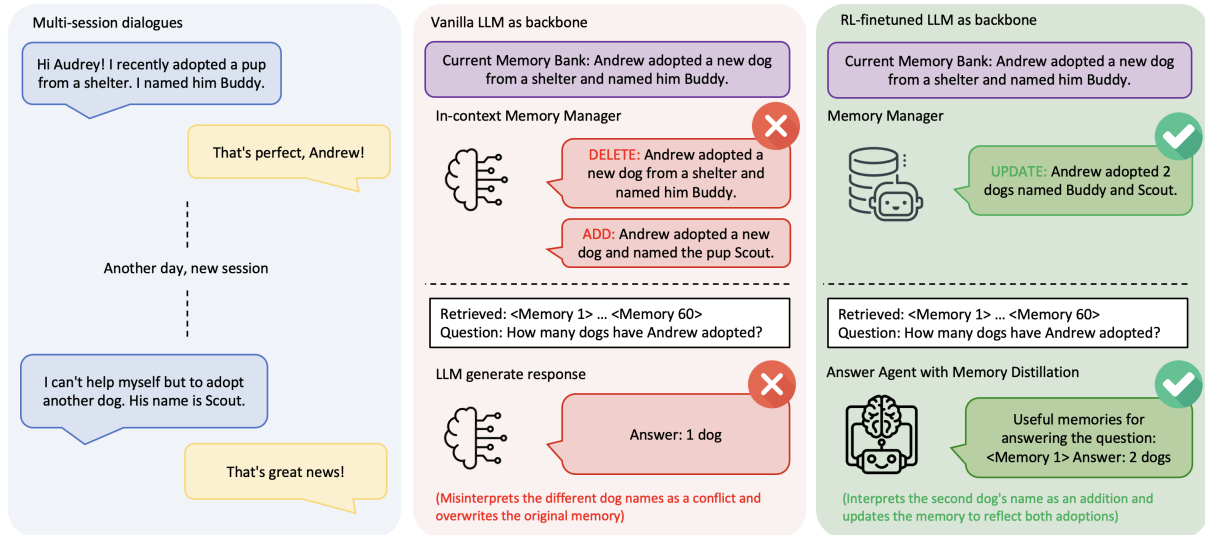


Figure 1: Comparison of Memory-R1 and a vanilla LLM memory system. (Left) In a multi-session dialogue, the user mentions adopting two dogs across sessions. (Middle) The vanilla Memory Manager misinterprets this as a contradiction and issues DELETE+ADD, fragmenting memory. (Right) The RL-trained Memory Manager issues a single UPDATE to consolidate the fact, while the Answer Agent distills 60 retrieved memories down to the relevant one (“Andrew adopted 2 dogs named Buddy and Scout”) and correctly answers “2 dogs.”

harana et al., 2024), shows how a user says “I adopted a dog named Buddy” and later adds “I adopted another dog named Scout”. A vanilla system misinterprets this as a contradiction, issuing DELETE+ADD and overwriting the original memory. A trained agent instead consolidates with an UPDATE: “Andrew adopted two dogs, Buddy and Scout.” Appendix A.1 provides a real dialogue trace illustrating this case in practice.

These challenges of retrieving and managing memory remain largely unsolved. Supervised fine-tuning provides limited help because it is impractical to label every memory operation or retrieval decision. Reinforcement learning (RL), in contrast, has recently shown strong potential for aligning LLM behavior with high-level objectives, including tool use (Qian et al., 2025; Wang et al., 2025), web navigation (Wei et al., 2025), and search optimization (Jin et al., 2025; Song et al., 2025). Building on this success, we argue that RL is the missing ingredient for adaptive memory in LLM agents. By optimizing outcome-based rewards, models can learn when to add, update, delete, or retain information and how to use retrieved memories for reasoning.

In this paper, we present Memory-R1, an RL fine-tuned, memory-augmented LLM framework with two specialized agents: (1) a *Memory Manager* that performs structured memory operations to maintain and evolve the memory bank, and

(2) an *Answer Agent* that applies a *Memory Distillation* policy to filter memories retrieved via Retrieval-Augmented Generation (RAG) and reason over the selected entries to produce answers. Both agents are fine-tuned using PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024), achieving strong performance with as few as 152 question–answer pairs. On the LoCoMo benchmark (Maharana et al., 2024), Memory-R1 delivers substantial gains over the most competitive baseline, Mem0 (Chhikara et al., 2025). Using the LLaMA-3.1-8B-Instruct backbone, Memory-R1-GRPO achieves relative improvements of 28% in F1, 34% in BLEU-1, and 30% in LLM-as-a-Judge. These improvements set a new state of the art on LoCoMo and underscore Memory-R1’s ability to achieve large performance gains with minimal supervision, highlighting its efficiency.

Our contributions are summarized as follows: (1) We introduce Memory-R1, the first RL framework for memory-augmented LLMs, consisting of a *Memory Manager* to perform structured memory operations and an *Answer Agent* to filter and reason over memories retrieved via RAG. (2) We develop a data-efficient fine-tuning strategy using PPO and GRPO that enables Memory-R1 to achieve strong performance with as few as 152 question–answer pairs, demonstrating that large memory improvements can be achieved with minimal supervision. (3) We provide in-depth analysis of RL choices,

model size, and memory design, offering actionable insights for building the next generation of memory-aware, reasoning-capable LLM agents.

## 2 Related Work

### 2.1 Memory Augmented LLM-based Agents

LLMs have emerged as powerful general-purpose reasoners, capable of engaging in multi-turn dialogues, decomposing tasks into actionable steps, and leveraging prior context to guide decision making (Brown et al., 2020; Chowdhery et al., 2022; OpenAI et al., 2024). However, their reliance on fixed-length context windows limits their ability to retain information over extended interactions. To overcome this, recent work augments LLM agents with external memory modules, enabling long-horizon reasoning and persistent knowledge accumulation through selective storage, retrieval, and updating of information. Several approaches illustrate this trend. LoCoMo (Maharana et al., 2024) introduces a benchmark to evaluate agents’ ability to retrieve and reason over temporally distant conversational history. ReadAgent (Lee et al., 2024) proposes a human-inspired reading agent that uses gist-based memory for reasoning over very long contexts. MemoryBank (Zhong et al., 2024) proposes a compositional memory controller for lifelong agent memory. MemGPT (Packer et al., 2023) introduces working and long-term buffers with scheduling policies. For a broader perspective, we refer readers to the recent survey on memory systems in AI agents (Du et al., 2025). While most existing approaches rely on static memory designs, our work instead develops a learnable memory system trained with reinforcement learning.

### 2.2 LLM and Reinforcement Learning

The intersection of LLM and RL has received increasing attention as researchers seek to move beyond static supervised fine-tuning and enable models to learn from dynamic, interactive feedback. Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022) is a foundational method used to align LLM outputs with human preferences. Recent works extend RL to more structured decision-making tasks for LLMs. For instance, Toolformer (Schick et al., 2023) and ReAct-style agents (Yao et al., 2023) frame tool use as an RL problem, where the LLM learns when to query external tools or APIs. Search-R1 (Jin et al., 2025) trains LLMs to issue web search queries

using RL to maximize final answer correctness. Similarly, the Trial and Error approach (Song et al., 2024) optimizes agents to select better reasoning paths. These approaches demonstrate that RL can improve complex behavior sequences in LLMs. However, memory management and utilization in LLMs remain underexplored in the RL setting. Existing memory-augmented LLM systems (Chhikara et al., 2025; Packer et al., 2023) typically rely on heuristics to control memory operations, lacking adaptability and long-term optimization. Our work, Memory-R1, is among the first to frame memory operation selection, and the utilization of relevant memories as an RL problem.

## 3 Method

We present Memory-R1, a reinforcement learning framework for multi-session dialogue tasks, where each dialogue contains multiple *sessions* (separate interactions occurring at different times) and each session consists of several *turns* (a back-and-forth exchange between two users). Answering a question always requires synthesizing information spread across sessions, posing a strong challenge for long-horizon memory management and reasoning. Figure 2 illustrates the overall pipeline. At each dialogue turn, the LLM extracts and summarizes information worth remembering, then retrieves related entries from the memory bank as part of the Retrieval-Augmented Generation (RAG) framework. The *Memory Manager* decides whether to ADD, UPDATE, DELETE, or NOOP, thereby maintaining and evolving the memory state. For question answering, the *Answer Agent* applies a memory distillation policy over retrieved memories to filter noise and reason over the most relevant content. Both agents are fine-tuned with PPO or GRPO, enabling outcome-driven learning of memory operations and selective utilization. Further implementation details, such as model hyperparameters, optimization schedule, and training setup, are provided in Appendix D.

### 3.1 RL Fine-tuning for Memory Manager

**Task Formulation** The *Memory Manager* maintains the memory bank by selecting one of ADD, UPDATE, DELETE, NOOP for each new piece of information extracted from a dialogue, outputting both the operation and updated content  $m'$ . Training uses (i) a partially constructed memory bank and (ii) a new dialogue turn with information relevant

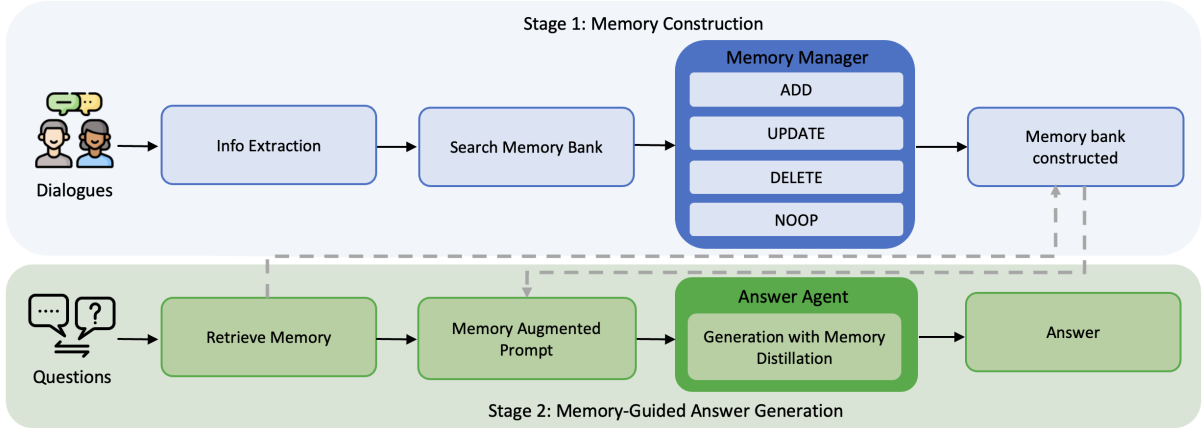


Figure 2: Overview of the Memory-R1 framework. Stage 1 (blue) constructs and updates the memory bank via the RL-fine-tuned Memory Manager, which chooses operations  $\{\text{ADD}, \text{UPDATE}, \text{DELETE}, \text{NOOP}\}$  for each new dialogue turn. Stage 2 (green) answers user questions via the Answer Agent, which applies a Memory Distillation policy to reason over retrieved memories.

to downstream QA. The goal is to learn which operation produces a memory state that enables the Answer Agent to answer correctly. Formally, the Memory Manager is modeled as a policy  $\pi_\theta$  that takes extracted information  $x$  and retrieved memories  $\mathcal{M}_{\text{old}}$  as input, and outputs an operation  $o$  with content  $m'$ :

$$(o, m') \sim \pi_\theta(\cdot | x, \mathcal{M}_{\text{old}}), \quad (1)$$

where  $x$  is the new information and  $\mathcal{M}_{\text{old}}$  the current memory bank. The data construction details are provided in Appendix B.2.

**PPO for Memory Manager** We fine-tune the Memory Manager with *Proximal Policy Optimization* (PPO; Schulman et al., 2017). Given candidate memory  $x$  and memory bank  $\mathcal{M}_{\text{old}}$ , the manager samples an operation  $o$  and updated content  $m'$  from policy  $\pi_\theta$ , applies it to the memory bank, and forwards the result to the frozen Answer Agent. Answer correctness provides a scalar reward  $r$ , from which we estimate an advantage  $A$ . The clipped PPO objective is:

$$\mathcal{J}(\theta) = \mathbb{E} \left[ \min(\rho_\theta A, \text{clip}(\rho_\theta, 1 - \epsilon, 1 + \epsilon)A) \right], \quad (2)$$

where  $\rho_\theta = \frac{\pi_\theta(o, m' | x, \mathcal{M}_{\text{old}})}{\pi_{\text{old}}(o, m' | x, \mathcal{M}_{\text{old}})}$  is the importance ratio,  $A$  is the advantage estimated from the answer-based reward  $r$ , and  $\epsilon$  is the clipping threshold for stable updates.

**GRPO for Memory Manager** We also train the Memory Manager with *Group Relative Policy Optimization* (GRPO; Shao et al., 2024), which samples

a group of  $G$  candidate actions per state and computes their relative advantages. This formulation avoids an explicit value function while maintaining PPO-style stability. For a state  $s = (x, \mathcal{M}_{\text{old}})$ , the GRPO objective is:

$$\mathcal{J}(\theta) = \mathbb{E} \left[ \frac{1}{G} \sum_{i=1}^G \rho_\theta^{(i)} A_i - \beta \mathbb{D}_{\text{KL}}[\pi_\theta \| \pi_{\text{ref}}] \right], \quad (3)$$

where each candidate  $i$  yields reward  $r_i$ ,  $A_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$ ,  $\mathbf{r} = \{r_1, \dots, r_G\}$ , is its standardized group-relative advantage, and  $\rho_\theta^{(i)}$  is the per-action importance ratio. The KL term regularizes updates to prevent policy drift away from the reference  $\pi_{\text{ref}}$ .

**Reward Design for Memory Manager** We use an outcome-driven reward: the Memory Manager’s operations are judged by their effect on downstream QA. After applying operation  $o$  with proposed content  $m'$ , the updated memory bank is passed to the frozen Answer Agent, and the reward is based on answer correctness:

$$R_{\text{answer}} = \text{EM}(y_{\text{pred}}, y_{\text{gold}}) \quad (4)$$

where  $y_{\text{pred}}$  is the predicted answer and  $y_{\text{gold}}$  the ground truth. This exact-match signal requires no manual labels, remains scalable, and is sufficient to teach effective memory operations.

## 3.2 RL Fine-Tuning for Answer Agent

**Task Formulation** The Answer Agent leverages the memory bank maintained by the Memory Man-

ager to answer questions in multi-session dialogues. Following (Chhikara et al., 2025), 60 candidate memories are retrieved for each question via similarity-based RAG, and the agent performs *memory distillation* to select the most relevant entries before generating an answer.

We model the agent as a policy  $\pi_\theta$  mapping the question  $q$  and retrieved set  $\mathcal{M}_{\text{ret}}$  to an answer  $y$ :

$$y \sim \pi_\theta(\cdot | q, \mathcal{M}_{\text{ret}}). \quad (5)$$

**PPO for Answer Agent** We fine-tune the Answer Agent using the same PPO algorithm as in Section 3.1. The agent takes the question  $q$  and retrieved memories  $\mathcal{M}_{\text{ret}}$  and generates an answer  $y$ . The objective mirrors Equation (2), applied to the generated sequence. The importance ratio is:

$$\rho_\theta(q, \mathcal{M}_{\text{ret}}) = \frac{\pi_\theta(y | q, \mathcal{M}_{\text{ret}})}{\pi_{\text{old}}(y | q, \mathcal{M}_{\text{ret}})}, \quad (6)$$

where  $y$  is the generated answer. Advantages derive from final answer quality (e.g., exact match), and clipping ensures stable updates.

**GRPO for Answer Agent** We also fine-tune the Answer Agent with GRPO, following the formulation in Section 3.1. For each  $(q, \mathcal{M}_{\text{ret}})$ , the policy samples  $G$  candidate answers  $\{y_i\}_{i=1}^G$ . Their exact-match rewards against  $y_{\text{gt}}$  are normalized into group-relative advantages. GRPO uses the same importance ratio as PPO, computed per candidate, and stabilizes training without a value function by comparing candidates within each group.

**Reward Design for Answer Agent** We use the Exact Match (EM) score between the generated answer  $y_{\text{pred}}$  and ground truth  $y_{\text{gold}}$  as the reward. This design directly ties the reward to the correctness of the final answer, encouraging the agent to select and reason over memories in a way that yields accurate outputs rather than optimizing for intermediate steps.

## 4 Experiments

### 4.1 Experimental Setup

**Dataset and Model** We evaluate Memory-R1 on three benchmarks: LoCoMo (Maharana et al., 2024), MSC (Packer et al., 2023), and LongMemEval (Wu et al., 2024). LoCoMo contains long multi-session dialogues (about 600 turns, 26k tokens) with QA pairs covering single-hop, multi-hop, open-domain, and temporal reasoning. Fol-

lowing prior work (Chhikara et al., 2025), we exclude the adversarial subset and use a 1:1:8 train/validation/test split (152/81/1307 questions). Models are trained only on LoCoMo and evaluated zero-shot on MSC and LongMemEval. We use LLaMA-3.1-8B-Instruct and Qwen-2.5 Instruct backbones (3B, 7B, 14B). Dataset construction details are provided in Appendix B.

**Evaluation Metrics** We evaluate performance using three metrics: token-level F1 (F1), BLEU-1 (B1), and LLM-as-a-Judge (J). F1 and B1 measure lexical overlap with ground-truth answers, while J uses a separate LLM to assess semantic correctness, relevance, completeness, and contextual appropriateness. Implementation details for LLM-as-a-Judge are provided in Appendix C.

**Baselines** To evaluate the effectiveness of MEMORY-R1, we compare it against several established baselines for multi-session dialogue reasoning: (1) LoCoMo (Maharana et al., 2024), a RAG-style framework that converts entire dialogues into chunks and retrieves relevant segments for answering questions, serving as the benchmark baseline for long-range, multi-session conversation reasoning; (2) A-Mem (Xu et al., 2025), a dynamic agentic memory system that creates, links, and updates structured memories to enhance reasoning across sessions; (3) Mem0 (Chhikara et al., 2025), a modular memory system with explicit in context memory operations designed for scalable deployment; (4) MemoryOS (Kang et al., 2025), a system-level framework that treats memory as an operating system abstraction for LLMs, providing unified mechanisms for memory read, write, and management across sessions to support long-horizon reasoning; (5) Memory-SFT. To isolate the effect of RL, we implement a supervised fine-tuning variant of our framework. Memory-SFT uses the same architecture and training data as Memory-R1 but replaces RL optimization with behavior cloning from GPT-5-generated trajectories.

For a fair comparison, we re-implemented all baselines using both the LLaMA-3.1-8B-Instruct and Qwen-2.5-7B-Instruct models as backbones, with temperature set to 0 and a maximum token limit of 2048. This consistent setup ensures reproducibility and allows us to assess how each method performs across different model architectures.

Model	Method	Multi-Hop			Temporal			Open Domain			Single-Hop			Overall		
		F1↑	B1↑	J↑	F1↑	B1↑	J↑	F1↑	B1↑	J↑	F1↑	B1↑	J↑	F1↑	B1↑	J↑
LLaMA-3.1-8B Instruct	LoCoMo (RAG)	12.25	9.77	13.81	13.69	10.96	20.48	11.59	8.30	15.96	9.38	8.15	4.65	11.41	8.71	13.62
	A-Mem	21.62	16.93	44.76	13.82	11.45	34.93	34.67	29.13	49.38	25.77	22.14	36.43	29.20	24.40	44.76
	Mem0	27.29	18.63	43.93	18.59	13.86	37.35	34.03	24.77	52.27	26.90	21.06	31.40	30.41	22.22	45.68
	MemoryOS	31.89	23.05	52.72	13.80	12.78	31.33	40.74	33.67	57.36	28.74	21.44	23.64	35.04	27.99	48.20
	Memory-SFT	34.64	23.73	56.90	20.80	16.26	37.35	46.47	37.35	63.27	47.18	34.58	<b>54.65</b>	42.81	32.98	58.76
	Memory-R1-PPO	32.52	24.47	53.56	26.86	23.47	42.17	45.30	39.18	64.10	41.57	26.11	47.67	41.05	32.91	57.54
Memory-R1-GRPO	<b>35.73</b>	<b>27.70</b>	<b>59.83</b>	<b>35.65</b>	<b>30.77</b>	<b>53.01</b>	<b>47.42</b>	<b>41.24</b>	<b>68.78</b>	<b>49.86</b>	<b>38.27</b>	51.55	<b>45.02</b>	<b>37.51</b>	<b>62.74</b>	
Qwen-2.5-7B Instruct	LoCoMo (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	18.96	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	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	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
	Memory-SFT	27.81	20.25	57.74	24.62	22.28	46.99	43.33	34.06	66.85	44.41	34.32	<b>52.71</b>	39.51	30.84	61.13
	Memory-R1-PPO	<b>34.22</b>	23.61	57.74	<b>32.87</b>	<b>29.48</b>	<b>53.01</b>	44.78	38.72	66.99	42.88	30.30	42.25	41.72	33.70	59.53
Memory-R1-GRPO	33.64	<b>26.06</b>	<b>62.34</b>	23.55	20.71	40.96	<b>46.86</b>	<b>40.92</b>	<b>67.81</b>	<b>47.75</b>	<b>38.49</b>	49.61	<b>43.14</b>	<b>36.44</b>	<b>61.51</b>	

Table 1: Evaluation results of Memory-R1 and baselines across LLaMA-3.1-8B-Instruct and Qwen-2.5-7B-Instruct on the LoCoMo benchmark dataset. Models are evaluated on F1, BLEU-1 (B1), and LLM-as-a-Judge (J) across *Single Hop*, *Multi-Hop*, *Open Domain*, and *Temporal* questions. Higher values indicate better performance. The best results are marked in bold.

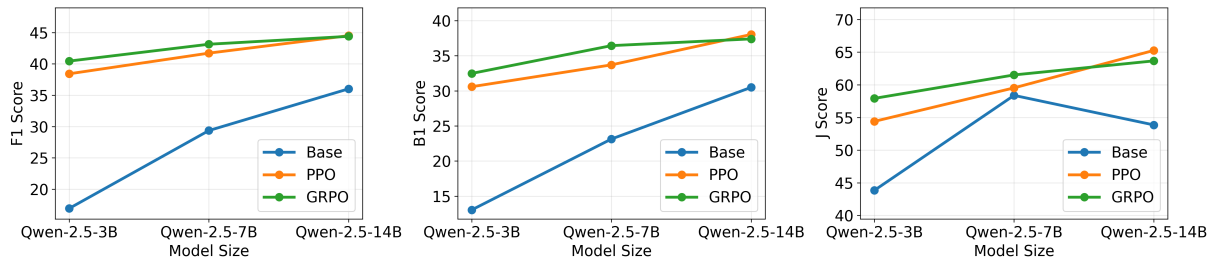


Figure 3: Scalability of Memory-R1 across model sizes (Qwen-2.5-3B, 7B, 14B-Instruct). Both PPO- and GRPO-tuned variants consistently outperform the base models across F1, BLEU-1 (B1), and LLM-as-a-Judge (J) metrics, showing strong scaling behavior.

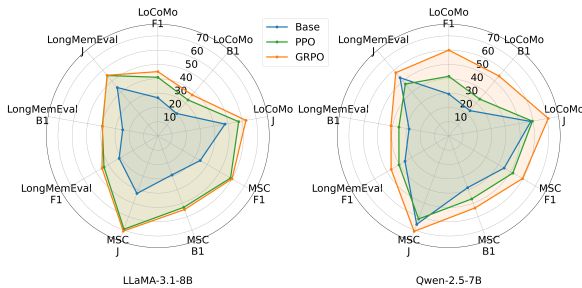


Figure 4: Generalization analysis of Memory-R1 across three benchmarks (LoCoMo, MSC, and LongMemEval), using LLaMA-3.1-8B-Instruct (left) and Qwen-2.5-7B-Instruct (right) as backbones.

## 4.2 Main Results

Table 1 reports the performance of Memory-R1 across LLaMA-3.1-8B-Instruct and Qwen-2.5-7B-Instruct models on the LoCoMo benchmark, covering diverse question types including single-hop, multi-hop, open-domain, and temporal reasoning. We evaluate two variants of Memory-R1, one fine-tuned with PPO and another with GRPO, and benchmark them against leading memory-

augmented baselines, including LoCoMo (RAG), A-Mem, Mem0, MemoryOS and Memory-SFT.

Across both model families, Memory-R1 consistently achieves new state-of-the-art performance. On LLaMA-3.1-8B, Memory-R1-GRPO delivers the strongest overall performance, improving F1 by 28.5%, B1 by 34.0%, and J by 30.2% relatively over the strongest baseline MemoryOS. Similarly, Memory-R1-PPO also yields substantial improvements, raising overall F1, B1, and J scores by 17.2%, 17.6%, and 19.4%, respectively. When applied to Qwen-2.5-7B-Instruct, Memory-R1-GRPO again emerges as the top performer, surpassing MemoryOS by margins of 24.5% (F1), 24.1% (B1), and 20.0% (J). PPO remains competitive, delivering strong gains over all non-RL baselines. Notably, while Memory-SFT benefits from guidance by a powerful teacher model (GPT-5), our reinforcement learning approach still outperforms it, highlighting the effectiveness of outcome-driven optimization over purely supervised imitation.

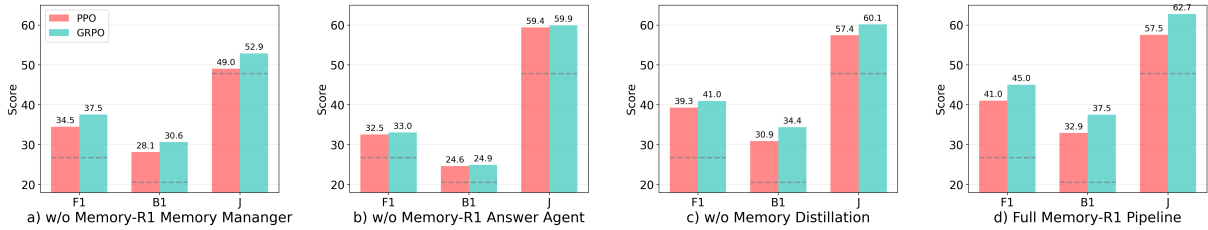


Figure 5: Ablation analysis of Memory-R1. Each subfigure shows the effect of removing one component: (a) Memory Manager, (b) Answer Agent, (c) Memory Distillation, and (d) the full pipeline. Performance drops in all ablations, demonstrating that each component contributes to the final results. Grey dashed lines indicate the baseline pipeline without RL fine-tuning.

### 4.3 Generalization and Scalability

We further investigate the robustness of Memory-R1 across model scales and datasets. Figure 3 shows results on the Qwen-2.5 family (3B, 7B, 14B). Memory-R1 consistently outperforms the base model at every scale, with PPO and GRPO delivering clear gains in F1, BLEU-1, and J scores. These improvements persist as models scale, demonstrating that reinforcement learning remains effective in teaching LLMs memory management regardless of backbone capacity.

To evaluate cross-task generalization, we apply the pipeline fine-tuned only on LoCoMo directly to two additional benchmarks: MSC and LongMemEval. As shown in Figure 4, Memory-R1 with both PPO and GRPO continues to achieve consistent improvements across all three datasets and metrics, despite never being trained on MSC or LongMemEval. This zero-shot transfer highlights the robustness of Memory-R1 and shows its ability to generalize beyond its training distribution. The gains extend across single-hop, multi-hop, open-domain, and temporal questions, demonstrating Memory-R1 as a generalizable framework for adaptive, memory-augmented LLMs capable of long-horizon reasoning. Detailed results on LoCoMo, MSC, and LongMemEval, with type-level breakdowns, are provided in Appendix F.

### 4.4 Ablation Studies

We conduct ablation studies to assess the contribution of each component in Memory-R1, isolating the effects of the Memory Manager, the Answer Agent, and the Memory Distillation mechanism. We also compare the training dynamics of PPO and GRPO.

**Effect of Memory Manager** We compare the full Memory-R1 pipeline with an ablated variant without RL fine-tuning of the Memory Manager,

both using LLaMA-3.1-8B-Instruct. As shown in Figure 5 (a,d), removing the RL-fine-tuned Memory Manager consistently degrades performance. Under PPO, F1, BLEU-1, and LLM-as-a-Judge drop from 41.0, 32.9, and 57.5 to 34.5, 28.1, and 49.0, respectively. Under GRPO, the corresponding scores decrease to 37.5, 30.6, and 52.9. These results confirm that outcome-driven RL enables more effective memory operations than scripted control.

**Effect of Answer Agent** Figure 5 (b,d) shows that RL fine-tuning the Answer Agent substantially improves answer quality. Without the Memory-R1 Answer Agent, PPO achieves F1, BLEU-1, and J scores of 32.5, 24.6, and 59.4, while GRPO reaches 33.0, 24.9, and 59.9. With the full pipeline, PPO improves to 41.0, 32.9, and 57.5, and GRPO further increases performance to 45.0, 37.5, and 62.7. This demonstrates that reward-driven fine-tuning enhances answer quality beyond static retrieval. A case study is provided in Appendix A.2.

**Effect of Memory Distillation** We evaluate memory distillation by comparing Answer Agents trained with and without distillation (Figure 5 (c,d)). With distillation enabled, PPO improves from 39.3, 30.9, and 57.4 to 41.0, 32.9, and 57.5 on F1, BLEU-1, and J, respectively. GRPO shows larger gains, increasing from 41.0, 34.4, and 60.1 to 45.0, 37.5, and 62.7. These results indicate that filtering irrelevant memories reduces noise and improves reasoning.

**RL-Fine-Tuned Answer Agent Gains More with Stronger Memory Manager** We test whether Answer Agent gains depend on Memory Manager quality. Figure 6 compares PPO/GRPO agents with a LLaMA-3.1-8B manager versus a stronger GPT-4o-mini manager. Improvements are larger with the stronger manager (F1: +10.10 vs. +19.72; BLEU-1:

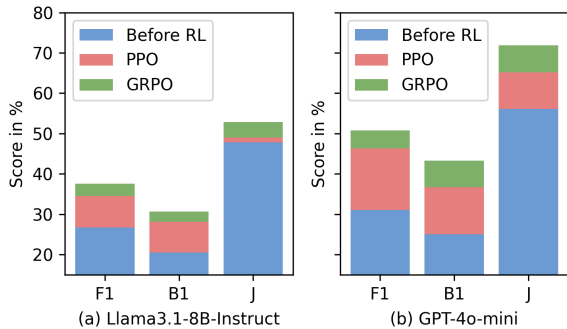


Figure 6: Performance gains of Answer Agent increase when paired with stronger Memory Managers, showing compounding benefits from higher memory quality.

Method	F1↑	B1↑	J↑
PPO (J-based reward model)	33.69	23.36	<b>63.58</b>
PPO (EM-based reward model)	<b>41.05</b>	<b>32.91</b>	57.54

Table 2: Reward Design Choice Comparison. PPO with J-based reward achieves higher J scores but lower F1 and B1 due to verbose outputs, while the EM-based reward yields balanced performance across metrics.

+10.81 vs. +18.19; J: +5.05 vs. +15.76), showing that Memory-R1 compounds benefits and the Answer Agent scales with memory quality.

**Comparison of RL Policies** We compare PPO and GRPO for training the Answer Agent, using exact match against ground-truth answers as the reward signal. As shown in Figure 7, GRPO exhibits faster initial convergence, likely due to its grouped return normalization providing stronger early guidance. However, as training progresses, both methods steadily improve and ultimately reach comparable final reward levels.

**Reward Design Analysis** We experimented with different reward models for fine-tuning the Answer Agent. As shown in Table 2, using the LLM-as-a-Judge value as reward leads to the highest J score (63.58), but performs poorly on F1 and BLEU-1. This is because the reward encourages longer, descriptive answers, which misaligns with string-overlap metrics. For example, when asked “Did John and James study together?”, the EM-based model outputs “Yes”, while the LLM-as-a-Judge-based model produces “Yes, John and James studied together, as they were part of the same on-line programming group, as implied by the memories above.” Although both are semantically correct, the latter is penalized under F1 and BLEU-1. This makes direct comparison with baselines difficult,

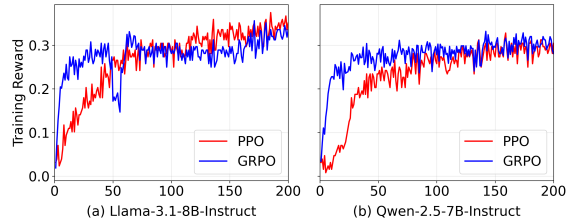


Figure 7: Training reward curves for PPO and GRPO on the Answer Agent using exact match as the reward. GRPO converges faster initially, and both reach similar final rewards.

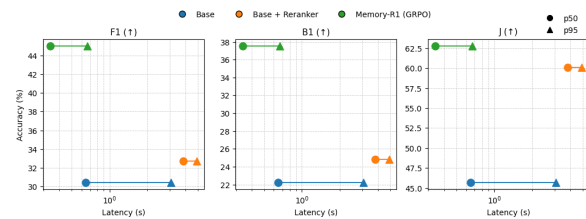


Figure 8: Accuracy and latency comparison across different inference pipelines: Base, Base + Reranker, and Memory-R1 (GRPO).

since responses are no longer length-controlled. To avoid bias from relying on a single metric, we adopt the EM reward, which yields balanced improvements across all three metrics.

**Comparison of Learned Memory Distillation and Reranking** We compare learned memory distillation in Memory-R1 with reranker-based pipelines in terms of accuracy and inference latency across three settings: Base, Base + Reranker, and Memory-R1 with a GRPO-trained Answer Agent (Figure 8). While reranking provides modest accuracy gains, it incurs substantial latency overhead. In contrast, Memory-R1 achieves higher accuracy with lower median and tail latency, demonstrating a more favorable accuracy–latency trade-off. Additional analyses are provided in Appendix G.

## 5 Conclusion

We presented Memory-R1, a reinforcement learning framework that enables LLM-based agents to effectively manage and utilize external memory. Unlike heuristic pipelines, Memory-R1 learns memory operations as well as memory distillation and usage for answering. With only 152 training examples, it achieves state-of-the-art results on Lo-CoMo, scales across model sizes, and generalizes to MSC and LongMemEval without retraining. Ablation studies confirm that reinforcement learning

improves every component of the system. Overall, Memory-R1 highlights reinforcement learning as a promising direction for adaptive and agentic memory in LLMs.

## Limitations

Our evaluation focuses on dialogue-centric datasets. While these benchmarks cover a wide range of reasoning types, extending Memory-R1 to multimodal data may introduce challenges beyond the scope of this work. Additionally, we train the Memory Manager and Answer Agent separately to ensure stability under sparse rewards. This separation is necessary but makes the process less straightforward. An end-to-end multi-agent reinforcement learning approach could simplify training and enable richer coordination, which we view as a promising direction for future work.

## Acknowledgements

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 projects 22189 and 22560.

## References

- AIOS Foundation. 2024. Aios agent sdk: Memory api. <https://docs.aios.foundation/aios-docs/aios-agent-sdk/memory-api>. Accessed: 2025-08-29.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. [Language models are few-shot learners](#). *Preprint*, arXiv:2005.14165.
- Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, and 48 others. 2022. [Palm](#): Scaling language modeling with pathways. *Preprint*, arXiv:2204.02311.
- Yiming Du, Wenyu Huang, Danna Zheng, Zhaowei Wang, Sebastien Montella, Mirella Lapata, Kam-Fai Wong, and Jeff Z. Pan. 2025. [Rethinking memory in ai: Taxonomy, operations, topics, and future directions](#). *Preprint*, arXiv:2505.00675.
- Siqi Fan, Xiusheng Huang, Yiqun Yao, Xuezhi Fang, Kang Liu, Peng Han, Shuo Shang, Aixin Sun, and Yequan Wang. 2025. If an llm were a character, would it know its own story? evaluating lifelong learning in llms. *arXiv preprint arXiv:2503.23514*.
- Weizhi Fei, Xueyan Niu, Pingyi Zhou, Lu Hou, Bo Bai, Lei Deng, and Wei Han. 2023. [Extending context window of large language models via semantic compression](#). *Preprint*, arXiv:2312.09571.
- Lyle Goodyear, Rachel Guo, and Ramesh Johari. 2025. The effect of state representation on llm agent behavior in dynamic routing games. *arXiv preprint arXiv:2506.15624*.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.
- Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. Memory os of ai agent. *arXiv preprint arXiv:2506.06326*.
- Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. 2024. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. [Lost in the middle: How language models use long contexts](#). *Preprint*, arXiv:2307.03172.
- Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. 2024. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*.
- James Martin. 1983. *Managing the Data-base Environment*. Prentice-Hall, Englewood Cliffs, New Jersey.
- Ali Modarressi, Abdullatif Köksal, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. 2024. Mem-llm: Finetuning llms to use an explicit read-write memory. *arXiv preprint arXiv:2404.11672*.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 224 others. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.

- Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. [Training language models to follow instructions with human feedback](#). *Preprint*, arXiv:2203.02155.
- Charles Packer, Vivian Fang, Shishir\_G Patil, Kevin Lin, Sarah Wooders, and Joseph\_E Gonzalez. 2023. Memgpt: Towards llms as operating systems. *ArXiv*.
- Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Xufang Luo, Hao Cheng, Dongsheng Li, Yuqing Yang, Chinyew Lin, H Vicky Zhao, Lili Qiu, and 1 others. 2025. On memory construction and retrieval for personalized conversational agents. *arXiv preprint arXiv:2502.05589*.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. [Toolrl: Reward is all tool learning needs](#). *Preprint*, arXiv:2504.13958.
- Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, Junyang Lin, and 24 others. 2025. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. 2025. [Meminsight: Autonomous memory augmentation for llm agents](#). *Preprint*, arXiv:2503.21760.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. [Toolformer: Language models can teach themselves to use tools](#). *Preprint*, arXiv:2302.04761.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv: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*, pages 1279–1297.
- Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*.
- Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. [Trial and error: Exploration-based trajectory optimization for llm agents](#). *Preprint*, arXiv:2403.02502.
- Volker Tresp, Sahand Sharifzadeh, Hang Li, Dario Konopatzki, and Yunpu Ma. 2023. The tensor brain: A unified theory of perception, memory, and semantic decoding. *Neural Computation*, 35(2):156–227.
- Cangqing Wang, Yutian Yang, Ruisi Li, Dan Sun, Ruicong Cai, Yuzhu Zhang, Chengqian Fu, and Lillian Floyd. 2024. [Adapting llms for efficient context processing through soft prompt compression](#). *Preprint*, arXiv:2404.04997.
- Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. 2025. [Acting less is reasoning more! teaching model to act efficiently](#). *Preprint*, arXiv:2504.14870.
- 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. 2025. [Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning](#). *Preprint*, arXiv:2505.16421.
- Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*.
- Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. 2025. How memory management impacts llm agents: An empirical study of experience-following behavior. *arXiv preprint arXiv:2505.16067*.
- Jing Xu, Arthur Szlam, and Jason Weston. 2021. Beyond goldfish memory: Long-term open-domain conversation. *arXiv preprint arXiv:2107.07567*.
- Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. 2025. [A-mem: Agentic memory for llm agents](#). *Preprint*, arXiv:2502.12110.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [React: Synergizing reasoning and acting in language models](#). *Preprint*, arXiv:2210.03629.
- Lingfan Yu, Jinkun Lin, and Jinyang Li. 2025. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 144–158.
- Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024. A survey on the memory mechanism of large language model based agents. *ACM Transactions on Information Systems*.

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

## A Case Study of Behavior of Agents before and after Fine-tuning

### A.1 From In-context Memory Manager to RL fine-tuned Memory Manager

To demonstrate how RL fine-tuning improves memory operations, we present two real representative examples. In the first case, the user initially mentions adopting a dog named Buddy, and later states that they have adopted another dog named Scout.

#### Dialogue Excerpts:

- Audrey: That's great news! What's the pup's name?
- Andrew: It took us a while to decide, but we ended up going with 'Scout' for our pup - it seemed perfect for their adventurous spirit.

#### Vanilla Memory Manager Actions (failed):

- \*\*Operations Issued:\*\***
- DELETE: "Andrew adopted a new dog from a shelter and named him Buddy because he is his buddy."
  - DELETE: "Andrew enjoys taking walks with his newly adopted dog, Buddy, as a way to spend quality time together and get fresh air."
  - ADD: "Andrew adopted a new dog and named the pup 'Scout' because the name seemed perfect for their adventurous spirit."

#### Memory-R1 Memory Manager Actions (successful):

- \*\*Operations Issued:\*\***
- UPDATE: "Andrew adopted a new dog from a shelter and named him Buddy because he is his buddy."
- > "Andrew adopted a dog from a shelter and named him Buddy because he is his buddy, and later adopted another dog named Scout."

**Discussion:** The vanilla LLM-based memory manager misinterpreted the second adoption as a contradiction because it saw two separate statements:

"adopted a dog named Buddy" and "adopted another dog named Scout", and assumed that the differing dog names implied the new statement replaced the old one. As a result, it deleted the first dog ("Buddy") and added the second ("Scout"), leaving the memory fragmented and inaccurate. In contrast, the RL fine-tuned Memory Manager recognized the events as complementary and consolidated them with a single UPDATE, maintaining a coherent memory bank and enabling the system to answer correctly. This case illustrates how reinforcement learning enhances the Agent to understand the nuance between ADD, UPDATE, and DELETE, allowing the model to handle overlapping or related information more intelligently.

To further demonstrate how RL fine-tuning improves memory operations, we present another real-world example from a user-agent dialogue involving Joanna's feelings about pets and her allergies, particularly toward turtles.

#### Dialogue Excerpts:

- Joanna: I wish I wasn't allergic! I would get two turtles today if I could! I found out recently I'm allergic to cockroaches as well, so who knows if I'll ever get a pet.
- Nate: Sorry! Maybe there are other animals you could consider! In the meantime though, I'll be sure to send you pics of my turtles so you can still watch them grow without getting too close.

#### Vanilla Memory Manager Actions (failed):

- \*\*Operations Issued:\*\***
- DELETE: "Joanna is allergic to most reptiles and animals with fur, which causes her face to get puffy and itchy. Despite this, she finds other ways to be happy."
  - DELETE: "Joanna finds turtles peaceful and believes they bring calm and joy."
  - DELETE: "Joanna thinks having pets must be a wonderful experience."
  - ADD: "Joanna is allergic to turtles and cockroaches, which affects her ability to have pets."

## Memory-R1 Memory Manager Actions (successful):

```
**Operations Issued:**
· UPDATE: "Joanna is allergic to most reptiles and animals with fur, which causes her face to get puffy and itchy. Despite this, she finds other ways to be happy." -> "Joanna is allergic to most reptiles, including turtles and cockroaches, and animals with fur, which causes her face to get puffy and itchy. Additionally, she is also allergic to turtles and cockroaches, which affects her ability to have pets."
· UPDATE: "Joanna finds turtles peaceful and believes they bring calm and joy." -> "Joanna finds turtles peaceful and believes they bring calm and joy, but she is allergic to them."
```

**Discussion:** The vanilla Memory Manager misinterpreted the new allergy information in two aspects. First, it viewed the difference between “allergic to most reptiles and animals with fur” and “allergic to turtles and cockroaches” as a contradiction in allergy type—failing to recognize the latter as a more specific elaboration rather than a conflicting statement. Second, it interpreted Joanna’s expressed fondness for turtles as incompatible with her allergy to them, incorrectly assuming that emotional attachment and physical limitations cannot coexist. As a result, it issued a series of DELETE operations followed by a single ADD, discarding valuable emotional context such as Joanna’s admiration for turtles and her general enthusiasm toward pets. In contrast, the RL fine-tuned Memory Manager recognized that these pieces of information were complementary: Joanna likes turtles but cannot keep them due to her allergies. It updated the relevant memories accordingly using targeted UPDATE operations, preserving both factual accuracy and emotional nuance. This case demonstrates how reinforcement learning equips the model to reason about overlapping and evolving information more intelligently, favoring memory consolidation over fragmentation.

## A.2 From Vanilla LLM to Memory-Distilled RL Answer Agent

To illustrate how RL fine-tuned Answer Agent with *Memory Distillation* improves answer accuracy, we compare the original model’s output with the RL fine-tuned model on a representative example from LoCoMo. The prompt provided to the model is shown in Figure 11.

**Question:** *Does John live close to a beach or the mountains?*

**Original Model Output (Before Fine-Tuning):**

```
**Answer:** He lives near mountains.
```

**Memory-R1 Output (After RL Fine-Tuning with Memory Distillation):**

```
**Memories selected as relevant:**
· 8:30 pm on 1 January, 2023: John has a nostalgic memory of having a film camera as a kid and taking many pictures at the beach.
· 1:24 pm on 25 May, 2023: John shared a picture of his family at the beach, expressing a sense of making a difference and commitment to continue their efforts.
**Answer:** beach
```

**Discussion:** The original model consumed all retrieved memories indiscriminately and defaulted to “mountains,” likely influenced by irrelevant mentions of mountaineering. In contrast, Memory-R1 filtered out distractors, surfaced only beach-related memories, and generated the correct answer. This case highlights how *Memory Distillation* helps the model discard noise, focus on true signals, and improve factual accuracy.

## B Dataset Details

### B.1 Test Data

**LoCoMo.** LoCoMo (Maharana et al., 2024) is a benchmark of long-term multi-session dialogues, with conversations averaging 300 turns and 9k tokens, spanning up to 35 sessions. It serves as our primary experimental dataset, on which we conduct and report detailed results.

**MSC.** We further evaluate on the Multi-Session Chat (MSC) dataset (Xu et al., 2021), which contains open-domain dialogues spanning multiple sessions. Following MemGPT (Packer et al., 2023), we use a modified version of MSC tailored to the memory-augmented evaluation setting, where questions depend on information distributed across earlier sessions. This dataset tests whether models can maintain continuity across temporally separated interactions.

**LongMemEval** We also evaluate on LongMemEval (Wu et al., 2024), a benchmark designed to test long-term memory capabilities of LLMs. It covers diverse tasks including factual recall, temporal reasoning, and entity tracking, with questions requiring integration of information from long and sparse contexts. LongMemEval complements LoCoMo and MSC by emphasizing broader generalization beyond dialogue-centric settings. In our experiments, we adopt the oracle version of LongMemEval.

## B.2 Training Data

We construct separate training datasets for the *Memory Manager* and the *Answer Agent* from the LoCoMo multi-turn dialogues. The LoCoMo dataset is publicly released under the **CC BY-NC 4.0** license. We slightly modify it for dialogue segmentation to fit our reinforcement learning pipeline, while preserving its original license terms and using it solely for non-commercial research purposes. All other datasets used in this paper (MSC and LongMemEval) are publicly available research benchmarks and are used in accordance with their respective licenses.

**Memory Manager Training Data.** For each dialogue turn  $t$ , GPT-4o-mini builds a temporal memory bank from the preceding 24 turns. The current turn  $t$  is fused with this snapshot to form the input. Unlike supervised annotation of memory operations, we do not provide explicit labels (ADD, UPDATE, DELETE, NOOP). Instead, the Memory Manager is optimized via reinforcement learning, where the correctness of the downstream Answer Agent’s answer provides the learning signal. The full procedure is given in Algorithm 1.

**Answer Agent Training Data.** For each question  $q$  in LoCoMo, we retrieve 60 candidate memories using retrieval-augmented search (RAG) over the temporal memory bank. The retrieved set, paired

---

### Algorithm 1 Data Construction for Memory-R1 Training

---

```

1: Input: LoCoMo multi-turn dialogues  $\mathcal{D}$ 
2: Output: Training tuples for the Memory Manager (dialogue turn, temporal memory bank, QA)
3: for each dialogue  $d \in \mathcal{D}$  do
4:   for each turn  $t$  in  $d$  do
5:     Build a temporal memory bank using the previous 50 turns with GPT-4o-mini
6:     Combine (i) the temporal memory bank, (ii) the current turn  $t$ , and (iii) any QA pairs linked to  $t$ 
7:     Store the combined package as a single training tuple
8:   end for
9: end for

```

---

with the question and gold answer, serves as the training input for the Answer Agent, which learns to distill the relevant entries and generate concise, correct responses.

## C Prompts

In developing our Memory Manager Prompt, answer generation agent prompt, and LLM-as-a-Judge prompt, we adapt elements from the prompt released by prior work (Packer et al., 2023; Chhikara et al., 2025)

### C.1 Memory Manager Prompt

For training the Memory Manager, we use a detailed prompt that instructs the model how to perform four memory operations: ADD, UPDATE, DELETE, and NOOP. The full prompt spans multiple figures for readability.

### C.2 Answer Agent Prompt

We provide the full prompt used to instruct the Answer Agent in our case study. This prompt defines the reasoning process, memory selection criteria, and formatting requirements for the model’s responses. Figure 11 shows the complete instructions, context, and representative retrieved memories.

### C.3 LLM-as-a-Judge (J) Prompt

For evaluating the correctness of generated answers, we employ an LLM-as-a-Judge prompt. The judge model is asked to label each answer as CORRECT or WRONG based on comparison with

### Memory Manager Prompt (Part 1): Overview and ADD/UPDATE Instruction

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

Based on the above four operations, the memory will change.

Compare newly retrieved facts with the existing memory. For each new fact, decide whether to:

- ADD: Add it to the memory as a new element
- UPDATE: Update an existing memory element
- DELETE: Delete an existing memory element
- NONE: Make no change (if the fact is already present or irrelevant)

1. **\*\*Add\*\***: If the retrieved facts contain new information not present in the memory, then you have to add it by generating a new ID in the id field.

- Example:

Old Memory:

```
[
  {"id" : "0", "text" : "User is a software engineer"}
]
```

Retrieved facts: ["Name is John"]

New Memory:

```
{
  "memory" : [
    {"id" : "0", "text" : "User is a software engineer", "event" : "NONE"},
    {"id" : "1", "text" : "Name is John", "event" : "ADD"}
  ]
}
```

2. **\*\*Update\*\***: If the retrieved facts contain information that is already present in the memory but the information is totally different, then you have to update it.

If the retrieved fact contains information that conveys the same thing as the memory, keep the version with more detail.

Example (a) - if the memory contains "User likes to play cricket" and the retrieved fact is "Loves to play cricket with friends", then update the memory with the retrieved fact.

Example (b) - if the memory contains "Likes cheese pizza" and the retrieved fact is "Loves cheese pizza", then do NOT update it because they convey the same information.

Important: When updating, keep the same ID and preserve old\_memory.

- Example:

Old Memory:

```
[
  {"id" : "0", "text" : "I really like cheese pizza"},
  {"id" : "2", "text" : "User likes to play cricket"}
]
```

Retrieved facts: ["Loves chicken pizza", "Loves to play cricket with friends"]

New Memory:

```
{
  "memory" : [
    {"id" : "0", "text" : "Loves cheese and chicken pizza", "event" : "UPDATE",
      "old_memory" : "I really like cheese pizza"},
    {"id" : "2", "text" : "Loves to play cricket with friends", "event" : "UPDATE",
      "old_memory" : "User likes to play cricket"}
  ]
}
```

Figure 9: Memory Manager Prompt (Part 1): Overview and ADD/UPDATE operation instruction.

```

Memory Manager Prompt (Part 2): DELETE/NO_OPERATION Instructions

3. **Delete**: If the retrieved facts contain information that contradicts
the memory, delete it. When deleting, return the same IDs – do not generate new IDs.

- Example:
  Old Memory:
  [
    {"id" : "1", "text" : "Loves cheese pizza"}
  ]
  Retrieved facts: ["Dislikes cheese pizza"]

  New Memory:
  {
    "memory" : [
      {"id" : "1", "text" : "Loves cheese pizza", "event" : "DELETE"}
    ]
  }

4. **No Change**: If the retrieved facts are already present, make no change.

- Example:
  Old Memory:
  [
    {"id" : "0", "text" : "Name is John"}
  ]
  Retrieved facts: ["Name is John"]

  New Memory:
  {
    "memory" : [
      {"id" : "0", "text" : "Name is John", "event" : "NONE"}
    ]
  }

```

Figure 10: Memory Manager Prompt (Part 2): DELETE/NO\_OPERATION instructions.

---

### Algorithm 2 Data Construction for Answer Agent Training

---

- 1: **Input**: LoCoMo multi-turn dialogues  $\mathcal{D}$ , trained Memory Manager
  - 2: **Output**: Training tuples for the Answer Agent (question, retrieved memories, gold answer)
  - 3: **for** each dialogue  $d \in \mathcal{D}$  **do**
  - 4:   Use the Memory Manager to maintain an up-to-date memory bank across turns
  - 5: **end for**
  - 6: **for** each question  $q$  in  $d$  **do**
  - 7:   Use the question  $q$  as a query to retrieve the top 30 most relevant candidate memories for each participant from the memory bank
  - 8:   Pair (i) the question  $q$ , (ii) the 60 retrieved memories, and (iii) the gold answer  $a_{\text{gold}}$
  - 9:   Store the triplet as a single training tuple for Answer Agent fine-tuning
  - 10: **end for**
- 

the gold answer. The complete prompt template is shown in Figure 12.

## D Implementation Details

We fine-tune MEMORY-R1 on LLaMA-3.1-8B-Instruct and Qwen-2.5-3B, 7B, and 14B-Instruct

models to evaluate robustness across architectures. Experiments are primarily conducted on 4 NVIDIA H100 GPUs (80GB each), except for Qwen-2.5-14B, which requires 8 GPUs. The total batch size is 128 with a micro-batch size of 2 per GPU. The maximum prompt and response lengths are set to 4096 and 2048 tokens, respectively.

Prompts for memory operations and memory-augmented answer generation are adapted from Chhikara et al. (2025). Reinforcement learning fine-tuning is performed using PPO and GRPO within the VERL framework (Sheng et al., 2025). For PPO, actor and critic networks are jointly trained with learning rates of  $1 \times 10^{-6}$  and  $1 \times 10^{-5}$ , respectively, using a constant warmup schedule. GRPO updates only the actor via grouped return normalization.

During RL training, we use a decoding temperature of  $\tau = 1.0$  to encourage exploration and collect diverse reward signals, which helps stabilize policy learning. For validation and testing, greedy decoding ( $\tau = 0$ ) is applied to ensure deterministic outputs and consistent metric evaluation.

## E Algorithm

The overall Memory-R1 pipeline contains two complementary procedures, outlined in Algorithm 3

```

Full Prompt and Retrieved Memories

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
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"),
   calculate the actual date based on the memory timestamp.
6. Always convert relative time references to specific dates, months, or years.
7. Focus only on the content of the memories. Do not confuse character names
8. The answer should be less than 5-6 words.
9. IMPORTANT: Select memories you found that are useful for answering the questions,
   and output it before you answer questions.
10. IMPORTANT: Output the final answer after **Answer:**

# APPROACH (Think step by step):
1. Examine all relevant memories
2. Examine the timestamps carefully
3. Look for explicit mentions that answer the question
4. Convert relative references if needed
5. Formulate a concise answer
6. Double-check the answer correctness
7. Ensure the final answer is specific
8. First output the memories that you found are important before you answer questions

Memories for user John:
- 7:20 pm on 16 June, 2023: John has a special memory of a vacation to California where he experienced a
gorgeous sunset and an enjoyable night strolling the shore, creating meaningful memories with loved ones.
- 6:13 pm on 10 April, 2023: John explored the coast in the Pacific Northwest and visited some national
parks, finding the beauty of nature absolutely breathtaking.
- 3:14 pm on 13 August, 2023: John enjoys spending time outdoors with his family, including activities
such as hiking, hanging out at the park, and having picnics. He also values indoor family activities like
playing board games and having movie nights at home.
... (In total 30 most relevant memories from John's Memory Bank are provided) ...

Memories for user Maria:
- 6:29 pm on 7 July, 2023: John experienced a severe flood in his old area last week, which caused
significant damage to homes due to poor infrastructure.
- 1:24 pm on 25 May, 2023: Maria appreciates the beauty of small, meaningful moments in life, as reflected
in her reaction to a family beach photo shared by John.
- 3:14 pm on 13 August, 2023: Maria appreciates family bonding and is interested in the activities that
John and his family enjoy doing together.
... (In total 30 most relevant memories from Maria's Memory Bank are provided) ...

Question: Does John live close to a beach or the mountains?

```

Figure 11: Prompt and retrieved memories used in the case study, showing all instructions, context, and memory entries provided to the model.

and Algorithm 4. Algorithm 3 (Memory Bank Construction) governs how the system incrementally builds and refines the external memory bank as new dialogue turns arrive. For each dialogue input, an LLM extracts key information, retrieves semantically related entries from the memory bank via retrieval-augmented generation (RAG), and invokes the RL fine-tuned Memory Manager to classify the update action as one of {ADD, UPDATE, DELETE, NOOP}. Depending on the chosen action, the memory store is updated accordingly—either inserting a new entry, merging information into an existing one, pruning contradictory content, or leaving the memory unchanged.

Algorithm 4 (Memory-augmented Answer Gen-

eration) describes how the system leverages the constructed memory bank to generate answers. Given an incoming question, the model retrieves the top-k relevant memory candidates, concatenates them with the question to form a memory-augmented prompt, and applies the Answer Agent’s Memory Distillation policy to filter for the most relevant facts. The distilled memory context, along with the query, is then passed to the Answer Agent to produce the final response, which is added to the answer set. Together, these algorithms enable Memory-R1 to jointly manage memory and generate memory augmented answers.

Training in Memory-R1 is performed in two stages, with the Memory Manager and Answer

**LLM-as-a-Judge Prompt Template**

Your task is to label an answer to a question as 'CORRECT' or 'WRONG'.  
 You will be given the following data:  
 (1) a question (posed by one user to another user),  
 (2) a 'gold' (ground truth) answer,  
 (3) a generated answer,  
 which you will score as CORRECT or 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 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, or year. The generated answer might include relative references (e.g., "last Tuesday"), but you should be generous – if it refers to the same time period as the gold answer, mark it CORRECT, even if the format differs (e.g., "May 7th" vs. "7 May").

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.

Return the label in JSON format with the key as "label".

Figure 12: LLM-as-a-Judge prompt used to evaluate model answers. The judge model labels each generated answer as CORRECT or WRONG based on comparison with the gold answer, with explicit instructions for handling time references and topic matching.

Agent optimized separately. When training the Memory Manager, the Answer Agent is frozen and used only to provide outcome-based rewards: the Manager’s operations are reinforced if the resulting memory state improves the Answer Agent’s ability to answer correctly. Conversely, when training the Answer Agent, the Memory Manager is fixed to ensure a stable memory input. Algorithm 5 illustrates this process for the Memory Manager, where dialogue turns are processed sequentially, candidate operations are sampled, the memory bank is updated, and policy gradients (via PPO or GRPO) are applied based on downstream answer correctness. This decoupled setup avoids attribution ambiguity while still allowing both components to co-adapt over alternating training phases.

## F Extended Results and Type-Level Analysis

Tables 3 and 4 provide detailed type-level evaluation on the LoCoMo and LongMemEval benchmarks. On LoCoMo (Table 3), Memory-R1 achieves consistent improvements across all reasoning types, with the largest gains on multi-hop and temporal questions, confirming its ability to maintain and integrate long-range information.

On LongMemEval (Table 4), improvements are most pronounced in multi-session scenarios where continuity across temporally distant interactions is critical. Memory-R1 shows substantial gains on tasks requiring factual recall (SSU) and temporal reasoning (TR), while also yielding steady improvements in knowledge update (KU) and open-domain QA. Across reasoning types, GRPO generally outperforms PPO, particularly in scenarios involving reasoning over multiple or noisy memory entries.

In addition to type-level analysis, Table 5 reports overall performance on the LongMemEval benchmark, including all baseline methods as well as Memory-R1 variants. Importantly, Memory-R1 is fine-tuned only on the LoCoMo dataset and evaluated on LongMemEval without any additional training. Despite this zero-shot transfer setting, Memory-R1-GRPO outperforms all baseline systems across both LLaMA-3.1-8B and Qwen-2.5-7B backbones. Together, these results complement the main findings in Section 4, further reinforcing that Memory-R1 generalizes robustly across reasoning types, model families, and benchmark tasks.

Model	Method	Single Hop			Multi-Hop			Open Domain			Temporal			Overall		
		F1↑	B1↑	J↑	F1↑	B1↑	J↑	F1↑	B1↑	J↑	F1↑	B1↑	J↑	F1↑	B1↑	J↑
Qwen 2.5-3B	BASE	19.82	15.78	46.44	11.57	10.22	24.10	25.37	20.04	45.67	28.94	24.19	29.46	24.18	19.46	41.24
	PPO	28.60	19.02	<b>50.63</b>	<b>26.57</b>	<b>22.72</b>	40.96	41.06	35.60	58.73	43.92	29.73	50.00	38.42	30.59	54.40
	GRPO	<b>30.10</b>	<b>20.00</b>	49.37	25.29	22.69	<b>44.58</b>	<b>43.01</b>	<b>37.39</b>	<b>65.06</b>	<b>47.69</b>	<b>33.36</b>	<b>50.00</b>	<b>40.45</b>	<b>32.48</b>	<b>57.92</b>
Qwen 2.5-7B	BASE	23.61	17.78	60.67	17.86	14.40	43.37	31.39	24.34	<b>65.75</b>	32.66	27.51	40.31	29.36	23.14	<b>58.38</b>
	PPO	<b>34.92</b>	<b>25.69</b>	<b>59.00</b>	25.30	<b>22.66</b>	42.17	43.51	38.00	65.34	42.52	30.10	<b>41.86</b>	40.59	33.21	58.07
	GRPO	33.98	25.54	58.16	<b>25.50</b>	21.63	<b>46.99</b>	<b>44.72</b>	<b>48.99</b>	64.65	<b>43.54</b>	<b>35.52</b>	39.92	<b>41.31</b>	<b>34.74</b>	57.46
Qwen-2.5-14B	BASE	34.60	26.82	55.23	24.24	21.45	38.55	39.79	34.53	56.95	34.98	29.39	33.33	36.91	31.28	50.80
	PPO	37.59	<b>31.92</b>	63.18	<b>28.21</b>	<b>24.57</b>	<b>50.60</b>	<b>48.46</b>	<b>42.44</b>	<b>72.76</b>	43.78	34.73	50.78	<b>44.26</b>	<b>37.86</b>	<b>65.26</b>
	GRPO	<b>38.32</b>	30.64	<b>63.18</b>	22.71	20.40	42.17	46.70	41.70	67.13	<b>50.50</b>	<b>36.60</b>	<b>60.47</b>	44.40	37.32	63.50

Table 3: Extended evaluation of Memory-R1 with Qwen-2.5 model family as backbones on the LoCoMo benchmark. Results are reported across question types (Single-Hop, Multi-Hop, Open-Domain, Temporal) and overall performance. Best scores are highlighted in bold.

Task	LLaMA-3.1-8B			Qwen-2.5-7B		
	BASE	PPO	GRPO	BASE	PPO	GRPO
SSU (F1/B1/J)	61.9/53.2/80.0	<b>78.9/75.6/87.1</b>	76.0/70.3/87.1	64.4/54.6/90.0	70.8/65.5/80.0	<b>80.9/76.3/91.4</b>
SSP (F1/B1/J)	7.4/0.1/46.7	9.6/1.6/50.0	<b>11.5/3.8/63.3</b>	13.9/2.5/53.3	<b>14.9/2.2/66.7</b>	12.6/2.0/66.7
OD (F1/B1/J)	17.9/16.6/19.6	30.6/24.6/33.9	<b>31.2/25.3/33.9</b>	14.1/14.4/16.1	23.5/21.1/19.6	<b>26.8/23.0/26.8</b>
MS (F1/B1/J)	20.8/19.6/33.1	43.1/43.6/54.1	<b>50.0/48.1/57.9</b>	30.2/26.9/54.9	32.4/35.1/36.1	<b>51.7/48.5/63.2</b>
KU (F1/B1/J)	36.0/27.9/51.3	<b>46.4/43.1/55.1</b>	38.5/35.5/52.6	40.5/33.5/59.0	52.3/48.2/65.4	<b>54.4/51.3/65.4</b>
TR (F1/B1/J)	34.0/23.1/42.1	37.0/29.2/49.6	<b>41.5/30.3/45.1</b>	36.5/24.5/44.4	<b>38.1/26.3/38.4</b>	35.1/25.8/41.4
O (F1/B1/J)	31.3/25.0/44.2	43.6/39.5/55.2	<b>45.2/39.3/55.4</b>	35.5/28.3/53.2	40.3/35.5/47.4	<b>46.7/41.1/57.8</b>

Table 4: Extended evaluation of Memory-R1 on the LongMemEval benchmark using LLaMA and Qwen backbones. Each cell shows F1/B1/J for a given model-method combination, reported with one decimal precision. Task types are abbreviated as: SSU = Single-Session-User, SSP = Single-Session-Preference, OD = Open Domain, MS = Multi-Session, KU = Knowledge Update, TR = Temporal Reasoning, and O = Overall. The best value for each metric (F1, B1, J) within a task row is highlighted in bold.

Base Model	Method	Overall F1↑	Overall B1↑	Overall J↑
LLaMA-3.1-8B	LoCoMo (RAG)	20.55	15.17	21.00
	A-Mem	38.36	33.30	54.20
	Mem0	31.41	21.69	41.20
	Memory-SFT	43.89	36.72	54.80
	Memory-R1-PPO	43.60	<b>39.50</b>	55.20
	Memory-R1-GRPO	<b>45.20</b>	39.30	<b>55.40</b>
Qwen-2.5-7B	LoCoMo (RAG)	18.27	14.57	22.20
	A-Mem	41.55	36.58	54.80
	Mem0	38.44	34.53	46.80
	Memory-SFT	43.16	35.04	54.80
	Memory-R1-PPO	40.30	35.50	47.40
	Memory-R1-GRPO	<b>46.70</b>	<b>41.10</b>	<b>57.80</b>

Table 5: Overall results on the LongMemEval benchmark. We report the mean scores across all six evaluation dimensions. The best results are marked in bold.

## G Latency Analysis

We provide a detailed latency analysis to better understand the efficiency characteristics of Memory-R1 and its individual components. All latency results are reported using median (p50) and tail (p95) inference time, measured across three components of the pipeline: the Memory Manager, Memory Search, and the Answer Agent. We compare the base model, PPO-trained variants, and GRPO-trained variants on both LLaMA-3.1-8B and Qwen-

2.5-7B backbones.

**Overall Trends** Across both model families, Memory-R1 does not introduce prohibitive latency overhead despite incorporating explicit memory management and reasoning components. In many cases, GRPO-trained variants achieve lower tail latency than both the base model and PPO variants, indicating that reinforcement learning can improve not only accuracy but also inference efficiency.

**Memory Manager Latency** For the Memory Manager component, latency remains relatively stable across Base, PPO, and GRPO variants. On LLaMA-3.1-8B, median latency ranges narrowly between 1.98 s and 2.17 s, with p95 latency around 3.4-3.6 s. Similar behavior is observed on Qwen-2.5-7B, where p50 latency stays below 1.4 s across all variants. These results suggest that RL fine-tuning does not materially increase the computational cost of memory operation selection.

**Memory Search Latency** Memory Search exhibits consistently low latency across all settings. On both backbones, median latency remains below

---

**Algorithm 3** Memory Bank Construction via Memory Manager

---

```
1: Input: Multi-turn dialogue  $D = \{t_1, t_2, \dots, t_n\}$ ; Initial empty memory bank  $M$ 
2: Output: Updated memory bank  $M$ 
3: procedure CONSTRUCTMEMORYBANK( $D, M$ )
4:   for each dialogue turn  $t_i \in D$  do
5:     Extract key info:  $f_i \leftarrow \text{LLMExtract}(t_i)$ 
6:     Retrieve memories:  $M_{old} \leftarrow \text{TopK}(f_i, M)$ 
7:     Determine operation:
8:      $o_i \leftarrow \text{MemoryManager}(f_i, M_{old})$  where  $o_i \in \{\text{ADD}, \text{UPDATE}, \text{DELETE}, \text{NOOP}\}$ 
9:     if  $o_i = \text{ADD}$  then
10:       $M \leftarrow M \cup \{f_i\}$ 
11:     else if  $o_i = \text{UPDATE}$  then
12:       $M_{tmp} \leftarrow \text{Merge}(M_{old}, f_i)$ 
13:       $M \leftarrow M \setminus M_{old} \cup M_{tmp}$ 
14:     else if  $o_i = \text{DELETE}$  then
15:       $M \leftarrow M \setminus M_{old}$ 
16:     else if  $o_i = \text{NOOP}$  then
17:       $M \leftarrow M$ 
18:     end if
19:   end for
20:   return  $M$ 
21: end procedure
```

---

---

**Algorithm 4** Memory-augmented Generation via Answer Agent

---

```
1: Input: Question set  $Q = \{q_1, q_2, \dots, q_m\}$ ; Memory bank  $M$ ; Generation instruction text  $t$ 
2: Output: Answer set  $\hat{A}$ 
3: procedure GENERATEANSWERS( $Q, M, t$ )
4:    $\hat{A} \leftarrow \{\}$ 
5:   for each question  $q_i \in Q$  do
6:      $M_{ret} \leftarrow \text{TopK}(q_i, M)$ 
7:      $p_i \leftarrow \text{Concat}(t, q_i, M_{ret})$   $\triangleright p_i$  is the memory augmented prompt
8:      $M_{distill}, \hat{a}_i \leftarrow \text{AnswerAgent}(p_i)$ 
9:      $\hat{A} \leftarrow \hat{A} \cup \{\hat{a}_i\}$ 
10:   end for
11:   return  $\hat{A}$ 
12: end procedure
```

---

0.35 s, and p95 latency remains under 0.65 s. Differences between Base, PPO, and GRPO variants are minimal, indicating that improvements in downstream accuracy are not driven by more expensive retrieval operations.

**Answer Agent Latency** The Answer Agent shows the most pronounced latency differences across methods. On LLaMA-3.1-8B, the GRPO-trained Answer Agent achieves substantially lower median and tail latency, with p50 and p95 of 0.34 s and 0.67 s, compared to 0.65 s and 3.07 s for the base model and 0.91 s and 4.67 s for PPO. A similar pattern holds on Qwen-2.5-7B, where GRPO reduces p95 latency to 0.83 s, compared to 1.06 s for the base model and 2.60 s for PPO. This reduction suggests that GRPO encourages more concise and

---

**Algorithm 5** Memory-R1 Pipeline for Memory Manager

---

```
1: Input: Dataset  $D$  of tuples: dialogue turns  $ds$ , question-answer pairs  $(q_i, a_i)$ ; Temp memory bank  $M$ ; Memory Manager LLM  $\mathcal{L}_m$ ; Answer LLM  $\mathcal{L}_a$ ; Reward Function  $\mathcal{F}$ ; Generation instruction text  $t$ 
2: Output: Fine-tuned Memory Manager LLM  $\mathcal{L}_m$ 
3: procedure TRAINMEMORYMANAGER( $D, \mathcal{L}_m, \mathcal{L}_a, \mathcal{F}$ )
4:   for each tuple  $(ds, q_i, a_i) \in D$  do
5:      $M \leftarrow \{\}$ 
6:     for  $d_i \in ds$  do
7:       Facts Extraction:  $f_i \leftarrow \text{LLMExtract}(d_i)$ 
8:       Memory Retrieval:  $M_{ret} \leftarrow \text{TopK}(f_i, M)$ 
9:       Determine operation:  $o_i \sim \mathcal{L}_m(f_i, M_{ret})$ 
10:      if  $o_i = \text{ADD}$  then
11:         $M \leftarrow M \cup \{f_i\}$ 
12:      else if  $o_i = \text{UPDATE}$  then
13:         $M_{tmp} \leftarrow \text{Merge}(M_{ret}, f_i)$ 
14:         $M \leftarrow M \cup M_{tmp}$ 
15:      else if  $o_i = \text{DELETE}$  then
16:         $M \leftarrow M \setminus M_{ret}$ 
17:      else if  $o_i = \text{NOOP}$  then
18:         $M \leftarrow M$ 
19:      end if
20:    end for
21:    Get Context:  $C_{ret} \leftarrow \text{TopK}(q_i, M)$ 
22:    Update Prompt:  $p_i \leftarrow \text{Concat}(t, q_i, C_{ret})$ 
23:    Get Response:  $r_i \sim \mathcal{L}_a(p_i)$ 
24:    Policy Update:  $\mathcal{L}_m \leftarrow \text{RL}_{step}(\mathcal{L}_m, \mathcal{F}, a_i, r_i)$ , where  $\text{RL} \in \{\text{PPO}, \text{GRPO}\}$ 
25:  end for
26:  return  $\mathcal{L}_m$ 
27: end procedure
```

---

efficient reasoning paths during answer generation.

**Accuracy-Latency Relationship** Figures 13 and 14 further illustrate the relationship between accuracy and latency across components. In contrast to retrieval-heavy pipelines, Memory-R1 achieves higher accuracy while simultaneously reducing both median and tail latency. This behavior indicates a Pareto improvement rather than a trade-off, where learned memory distillation and policy optimization enable the model to reason more efficiently without sacrificing correctness.

Overall, these results demonstrate that Memory-R1 improves inference efficiency in addition to accuracy, especially in the Answer Agent component, and that reinforcement learning can lead to more streamlined reasoning behavior rather than increased computational cost.

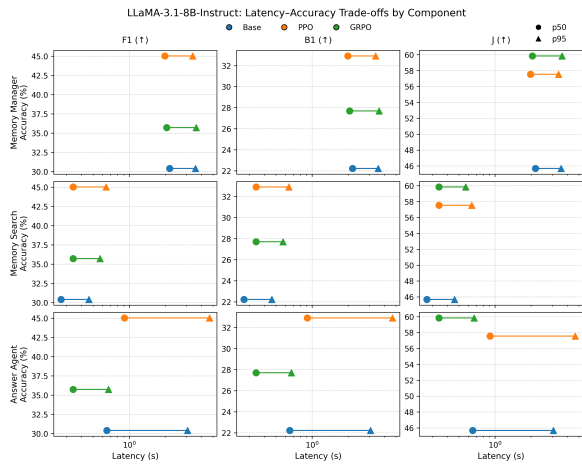


Figure 13: Latency-accuracy comparison across pipeline components on LLaMA-3.1-8B-Instruct. Points show median (p50) and tail (p95) latency versus accuracy (F1, BLEU-1, and LLM-as-a-Judge) for the base model and RL-trained variants (PPO, GRPO).

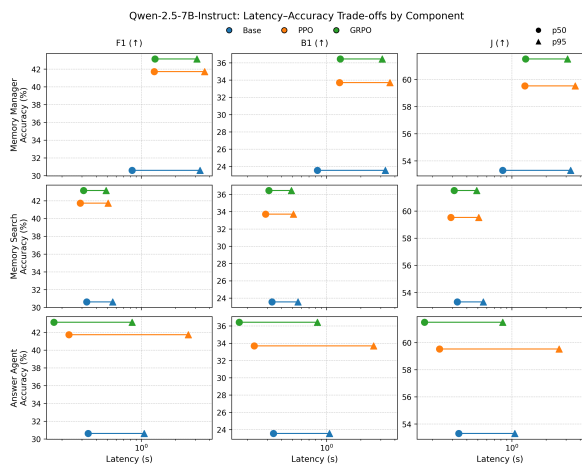


Figure 14: Latency-accuracy comparison across pipeline components on Qwen-2.5-7B-Instruct. Points represent median (p50) and tail (p95) latency versus accuracy for the base model and RL-trained variants (PPO, GRPO).