JANUSMEM: LEARNING LONG-TERM MEMORY EDITS FROM UTILITY CHANGE ACROSS TIME

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-term memory is essential for LLM agents that interact with an environment across many turns over time, where a memory manager must decide how to add, update, or delete entries as each turn arrives. Existing methods learn these decisions from the task outcomes observed after an edit. This signal overlooks both the utility change the edit causes, which separates its contribution from the utility already in memory, and the edit's effect on past and future queries: an edit that helps now may erase knowledge that earlier queries rely on or miss information that future queries need. To address both limitations, we present JanusMem, a reinforcement learning framework that measures the actual effect of each memory edit across time. On the current query, JanusMem scores an edit by both its post-edit utility and its change from the pre-edit memory, which isolates the edit's own contribution. JanusMem further uses a bidirectional window to credit improvement on nearby future queries and penalize deterioration on nearby past queries. Since utility changes can be subtle, all utilities are measured by a fixed LLM judge with continuous semantic feedback rather than lexical matching. Trained on only 152 QA pairs from a single LoCoMo conversation, JanusMem achieves state-of-the-art performance on the 8 held-out conversations, improving the LLM-Judge score over Memory-R1 by 11.6%. Without further training, the same model transfers zero-shot to LongMemEval and MemoryAgentBench, outperforming all baselines with gains of 7.6% and 2.9%, respectively.

1 INTRODUCTION

LLM agents increasingly interact with users and environments across many turns over time (Zhou et al., 2024; Yao et al., 2022). Such agents rely on a long-term memory in which a memory manager decides whether new information should be added, used to update or delete existing entries, or left out (Park et al., 2023; Packer et al., 2023; Chhikara et al., 2025). Ideally, each edit leaves the memory more useful than before, retaining the valid knowledge that earlier interactions established while capturing the information that later interactions will need. This requires measuring an edit not by whether it appears reasonable for the current turn, but by what it actually changes in the memory and how that change plays out over time.

Existing memory managers fall short of this requirement. Many systems edit memory with prompts, similarity rules, or hand-designed policies (Park et al., 2023; Packer et al., 2023; Zhong et al., 2024; Chhikara et al., 2025), which never assess whether an individual edit is beneficial. Learning-based managers instead optimize edits with reinforcement learning from task outcomes, either rewarding an edit by whether the answer agent answers the associated query correctly (Yan et al., 2026b) or distributing downstream rewards across operations (Yan et al., 2026a; Ma et al., 2026; Yue et al., 2026; Mao et al., 2026). These rewards largely share two blind spots. First, they measure the outcome reached after an edit rather than the change it causes, failing to separate the edit's contribution from the utility already in memory. Second, they measure an edit only on the queries it is meant to support, leaving unchecked whether it degrades queries that the memory could previously answer. This raises a central question: *how can we measure the actual effect of an individual memory edit across time?*

We answer this question with JanusMem, a reinforcement learning framework named after the Roman god who looks both backward and forward. JanusMem considers a memory edit as a transition

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

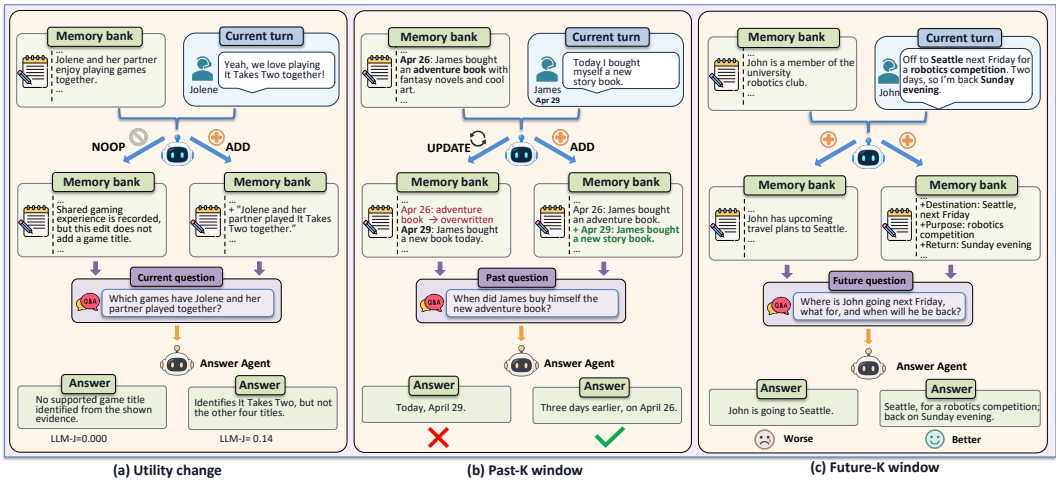

Figure 1: **Motivation for JanusMem.** (a) Utility change captures small improvements even when an answer remains incomplete. (b) The past-K window helps preserve facts needed to answer earlier queries. (c) The future-K window encourages recording specific details that support later queries.

between two memory states and measures it on the current query with two complementary signals: the post-edit utility, *i.e.*, the quality score of the answer produced from the edited memory, and the utility change, *i.e.*, the *difference* in this score between the post-edit and pre-edit memories. Together, they distinguish all 4 editing situations in Table 1, and the utility change credits partial progress that outcome-based rewards miss (Figure 1a).

Beyond the current query, JanusMem further measures each edit on a bidirectional window of nearby past and future queries, which sets a higher standard for the memory manager. To support future queries, the manager must record all relevant facts from the current turn, not only those the current query needs: *"John has upcoming travel plans to Seattle"* (incomplete) tells where John is going, but only *"John goes to Seattle next Friday for a robotics competition, back Sunday evening"* also tells why and when he returns (Figure 1c). To protect past queries, the manager must preserve precise distinctions: overwriting James's earlier purchase with *"James bought a new book on Apr 29"* (imprecise) and adding *"James bought a new story book on Apr 29"* both answer a query about his latest book, but only the latter still tells when he bought his adventure book (Figure 1b). By crediting improvement on future queries and penalizing deterioration on past ones, JanusMem encourages edits that record complete and precise facts.

Since all of these signals depend on subtle utility changes, JanusMem measures utility with a fixed LLM judge rather than lexical metrics such as exact match and F1, which compare surface forms rather than meaning and leave most edits with a zero utility change. The judge first selects a quality band and then assigns a score to two decimal places within it.

Trained on only 152 QA pairs from a single LoCoMo conversation, JanusMem achieves state-of-the-art performance on the 8 held-out conversations, improving the LLM-Judge score over Memory-R1 from 32.5% to 44.1%, and the same model transfers zero-shot to LongMemEval and MemoryAgentBench with absolute gains of 7.6% and 2.9%. Ablations confirm that fine-grained judging, the utility change, and the bidirectional window each improve performance, and the gains hold on a smaller Llama-3.2-3B backbone (+25.9% on LoCoMo).

Our contributions are as follows:

- We propose measuring each memory edit by its post-edit utility and its bounded utility change from the pre-edit memory, isolating what the edit itself contributes.
- We introduce a bidirectional window that credits improvement on nearby future queries and penalizes deterioration on nearby past queries, requiring memory that is both complete and precise.
- We show that JanusMem, trained on only 152 QA pairs, achieves state-of-the-art performance on LoCoMo and transfers zero-shot to LongMemEval and MemoryAgentBench.

2 Related Work

2.1 Dynamic Agent Memory Systems

Early agent memory systems extend limited context by storing and retrieving past interactions. Generative Agents maintain a chronological memory stream, retrieve experiences according to relevance, recency, and importance, and form reflections for planning (Park et al., 2023). MemGPT treats long-running interaction as a memory management problem, moving information between in-context memory and external storage through explicit operations (Packer et al., 2023). MemoryBank updates and forgets stored information according to its importance (Zhong et al., 2024). These systems establish persistent storage and retrieval as foundations for long-term agents.

More recent work focuses on how memory is organized and maintained. A-Mem creates notes with contextual metadata and semantic links, allowing related memories to form an associative structure(Xu et al., 2026). Mem0 extracts salient facts from interactions and maintains a compact memory through structured operations; its graph variant additionally represents relations among entities(Chhikara et al., 2025). LightMem filters and groups information to improve the efficiency of memory-augmented generation (Fang et al., 2026), while MemoryOS organizes memory across components operating at different time scales (Kang et al., 2025). Although these methods support evolving memories, their decisions about what to store, update, or remove are primarily governed by prompts or designed rules. Our work instead learns editing decisions from their effects on answering performance.

2.2 Learning-Based Memory Management

Learning-based approaches optimize memory policies using downstream task outcomes. Memory-R1 trains a Memory Manager and an Answer Agent with reinforcement learning, providing feedback on memory decisions through subsequent question answering (Yan et al., 2026b). MEM1 jointly learns memory consolidation and reasoning while maintaining a compact state under a fixed memory budget (Zhou et al., 2026). AgeMem learns long-term memory writing together with short-term context management using step-wise reinforcement learning (Yu et al., 2026). These methods show that memory management can be effectively learned rather than specified entirely through hand-designed rules.

A related line of work improves credit assignment for individual memory operations. Memory-R2 compares candidate operations using local re-rollouts from an intermediate memory state (Yan et al., 2026a). Fine-Mem associates downstream feedback with operations supported by relevant evidence (Ma et al., 2026). Mem-T and TreeMem propagate feedback across trajectories or trees of memory operations (Yue et al., 2026; Mao et al., 2026). In contrast, JanusMem addresses a different aspect of edit evaluation: it compares the answering utility before and after an edit, while also checking whether the edit preserves support for past questions and improves support for future ones. This combines edit-level change with temporal consequences in the reward used to learn memory management.

3 Method

JanusMem trains a memory manager with a reward that measures the actual effect of each edit across time (Figure 2). Section 3.1 formulates memory editing and the training procedure. The next two subsections address the two blind spots of outcome-based rewards: Section 3.2 measures an edit on the current query by both its post-edit utility and its utility change, separating the edit's contribution from the utility already in memory, and Section 3.3 extends this measurement to nearby past and future queries. Finally, Section 3.4 describes the LLM judge that measures utility finely enough for these comparisons and combines all components into the final reward.

3.1 Problem Formulation

Memory editing. We consider an LLM agent that interacts with an environment over turns $t = 1, \ldots, T$ and maintains an external memory bank M_t of textual entries, each with a unique

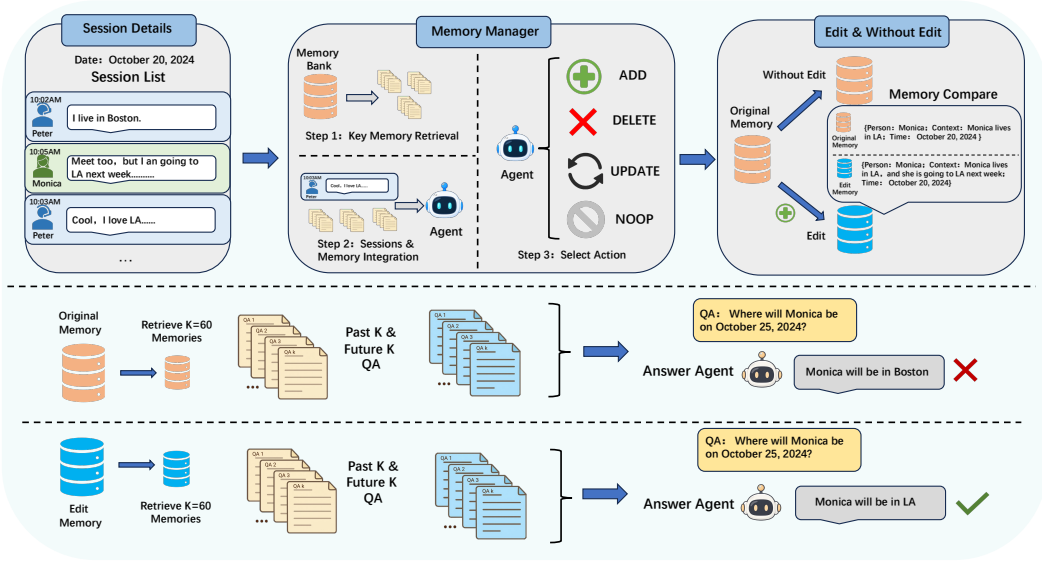

Figure 2: **Overview of JanusMem.** The Memory Manager retrieves relevant memories, integrates the current turn, and selects an ADD, UPDATE, DELETE, or NOOP operation. Each candidate edit is measured by comparing the edited and pre-edit memories on the current query and on nearby past-K and future-K queries, using a fixed Answer Agent and LLM judge.

identifier and a timestamp. Following Memory-R1 (Yan et al., 2026b), the agent consists of a Memory Manager and an Answer Agent. At turn t, the Memory Manager receives a new input x_t, retrieves the relevant entries $C_t \subseteq M_t$, and proposes an edit that adds new information, updates or deletes an existing entry, or leaves the memory unchanged:

$$a \sim \pi_\theta(\cdot \mid x_t, C_t), \qquad M_t^a = \mathcal{T}(M_t, a), \tag{1}$$

where π_θ is the Memory Manager policy and $\mathcal{T}$ applies the edit. The Answer Agent $\mathcal{A}$ answers a query q by retrieving from a memory state M, and we write its answer as $\mathcal{A}(q, M)$. We define the utility of a memory state for a query as the quality score of the answer it supports:

$$s(M, q) = J\big(q,\, y,\, \mathcal{A}(q, M)\big) \in [0, 1], \tag{2}$$

where y is the reference answer and J is a fixed LLM judge (Section 3.4). To measure an edit, we associate each turn with a current query q_t whose evidence lies in the current or nearby turns.

Training. We train the Memory Manager with GRPO (Shao et al., 2024) while keeping $\mathcal{A}$ fixed, so that all edits are measured under the same answering behavior. At turn t, the policy samples a group of G candidate edits $a_1, \ldots, a_G \sim \pi_{\theta_{\text{old}}}(\cdot \mid x_t, C_t)$ from the same memory state M_t. Each candidate receives a reward $R(a_i)$ defined in Equation 10 (we omit the turn index from reward terms), and rewards are normalized within the group:

$$\hat{A}_i = \frac{R(a_i) - \text{mean}\big(\{R(a_j)\}_{j=1}^G\big)}{\text{std}\big(\{R(a_j)\}_{j=1}^G\big)}. \tag{3}$$

Because all candidates share x_t, C_t, and M_t, these advantages compare edits under identical conditions, and any reward term that takes the same value for every candidate cancels. The policy is updated with the clipped objective

$$\mathcal{J}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G} \min\big(\rho_i \hat{A}_i,\ \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)\,\hat{A}_i\big)\right] - \beta\, \mathbb{D}_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big), \tag{4}$$

where $\rho_i = \pi_\theta(a_i \mid x_t, C_t)/\pi_{\theta_{\text{old}}}(a_i \mid x_t, C_t)$, ϵ is the clipping threshold, and β weights the KL penalty toward a reference policy π_{ref}. Following Memory-R1 (Yan et al., 2026b), the memory trajectory during training is prescribed by the dataset: M_{t+1} is given rather than produced by a sampled candidate, so the candidates serve only to compute rewards. At inference, edits sampled from π_θ advance the memory, and y, J, and the evaluation queries are no longer used.

3.2 Utility Change on the Current Query

Post-edit utility and utility change. For a candidate edit a, the post-edit utility $s(M_t^a, q_t)$ measures how useful the edited memory is for q_t, while the utility change

$$\Delta_t(a) = s(M_t^a, q_t) - s(M_t, q_t) \tag{5}$$

measures what the edit itself contributes. Neither signal suffices alone (Table 1). With binary utilities, $s(M_t^a, q_t)$ alone cannot separate $0 \to 1$ from $1 \to 1$, which differ in whether the edit repaired the memory or merely kept an already useful one, nor $0 \to 0$ from $1 \to 0$, which differ in whether the edit left a useless memory unchanged or destroyed a useful one. $\Delta_t(a)$ has the opposite weakness: it assigns 0 to both $0 \to 0$ and $1 \to 1$, although only the latter leaves a memory that answers the query. Only the two signals together separate all four transitions.

Current-query reward. We combine both signals into the reward for the current query:

$$R_{\text{cur}}(a) = \alpha\, s(M_t^a, q_t) + (1-\alpha)\tanh\left(\frac{\Delta_t(a)}{\tau}\right), \tag{6}$$

where $\alpha \in (0,1)$ balances the two terms and $\tau > 0$ controls the sensitivity to small changes (values in Appendix C). The first term favors useful memory states, and the second rewards improvement while penalizing degradation caused by the edit.

Table 1: Values assigned to the four transitions under binary utility. Each signal alone assigns the same value to transitions with different meanings.

Transition	Post-edit	Change
$0 \to 0$	0	0
$0 \to 1$	1	$+1$
$1 \to 0$	0	-1
$1 \to 1$	1	0

Why a bounded transformation. The $\tanh$ is necessary rather than cosmetic. All candidates in a group share M_t, so $s(M_t, q_t)$ is constant within the group. With a linear change term, the reward would reduce to

$$\alpha\, s(M_t^a, q_t) + (1-\alpha)\big(s(M_t^a, q_t) - s(M_t, q_t)\big) = s(M_t^a, q_t) - (1-\alpha) \underbrace{s(M_t, q_t)}_{\text{same for all candidates}}, \tag{7}$$

and the constant would cancel under group normalization (Equation 3), leaving only $s(M_t^a, q_t)$. The $\tanh$ instead makes the reward steepest where $s(M_t^a, q_t)$ crosses $s(M_t, q_t)$, so the advantages emphasize whether an edit improves or degrades the memory rather than its absolute utility alone. It also bounds $\tanh(\Delta_t(a)/\tau)$ by $\tanh(1/\tau)$, so a few large changes cannot dominate training, while staying nearly linear around $\Delta_t(a) = 0$, so that small but real improvements still produce a signal. A smaller τ sharpens this response, whereas a larger τ moves the term back toward the linear case.

3.3 Bidirectional Window over Past and Future Queries

Past and future queries. Equation 6 measures an edit only on q_t, yet an edit that helps q_t may still record incomplete facts that later queries need or overwrite distinctions that earlier queries depend on (Figure 1b,c). We therefore also measure the edited memory on nearby queries in both directions. We select them by the position of their supporting evidence in the interaction sequence: $\mathcal{Q}_{\text{past}}$ contains the K nearest queries whose evidence precedes turn t, and $\mathcal{Q}_{\text{future}}$ the K nearest queries whose evidence follows it, both excluding q_t. When fewer than K queries are available on one side, we use those that exist.

Future credit and past penalty. The future term credits the edited memory for supporting later queries:

$$R_{\text{future}}(a) = \frac{1}{|\mathcal{Q}_{\text{future}}|} \sum_{q \in \mathcal{Q}_{\text{future}}} s(M_t^a, q) \;\geq 0. \tag{8}$$

Since $s(M_t, q)$ is the same for all candidates in a group, crediting $s(M_t^a, q)$ yields the same advantages (Equation 3) as crediting its improvement $s(M_t^a, q) - s(M_t, q)$, so R_{future} rewards edits that improve support for future queries without additional judge calls on M_t. The past term instead penalizes deterioration relative to the pre-edit memory:

$$R_{\text{past}}(a) = \frac{1}{|\mathcal{Q}_{\text{past}}|} \sum_{q \in \mathcal{Q}_{\text{past}}} \min\big(0,\; s(M_t^a, q) - s(M_t, q)\big) \;\leq 0. \tag{9}$$

A past query contributes 0 if M_t^a answers it at least as well as M_t, and a penalty only if M_t^a answers it worse. Because each past query is measured in its original temporal context, R_{past} does not require the Memory Manager to keep outdated facts as current knowledge.

Why both sides are needed. The two terms play complementary roles. R_{future} is purely a credit that gives the Memory Manager a reason to record information beyond what q_t needs; without R_{past}, however, edits could overwrite earlier knowledge at no cost. R_{past} is purely a penalty that constrains how memory is written; without R_{future}, the window never rewards an edit, so leaving the memory unchanged is never worse on it and the manager has no incentive to record more than q_t requires. Together, they favor edits that are complete for later queries and precise enough to preserve earlier ones.

Cost. Each candidate requires $2K$ judge calls on M_t^a, while $s(M_t, q)$ for $q \in \mathcal{Q}_{\text{past}}$ is computed once and shared by the whole group. The cost therefore grows linearly in K, which favors a compact window.

3.4 Fine-Grained Utility Measurement

Why lexical metrics are not enough. Every reward term above is computed from $s(M, q)$ (Equation 2), so the reward is only as informative as this measurement. Token-level F1 and exact match compare surface forms rather than meaning: F1 stays high when an answer copies most of the reference but changes the fact in question, such as reporting March 19 instead of March 12, while exact match gives no credit to equivalent answers such as "March 12" and "On March 12". Such errors hurt utility changes more than post-edit utility, because a change such as $\Delta_t(a)$ is a difference of two scores and inherits the errors of both. A metric that misses a semantic change reports $\Delta_t(a) = 0$ for an edit that in fact helped or hurt, and a binary metric restricts $\Delta_t(a)$ to $\{-1, 0, +1\}$, leaving most edits in the uninformative zero case.

Judge design. We therefore obtain $s(M, q)$ from a fixed LLM judge J that compares $\mathcal{A}(q, M)$ with y by meaning and accepts equivalent wording. J uses a different base model from the Memory Manager and $\mathcal{A}$: a judge sharing the base model of $\mathcal{A}$ would tend to prefer answers in its own style, rewarding an edit for matching that style rather than for improving the memory. J scores in two stages (Table 2): it first places the answer in one of five equal-width bands of $[0, 1]$, and then reports a score to two decimal places within that band. The coarse stage is easier to apply consistently than a direct continuous score, and the fine stage gives $\Delta_t(a)$ the resolution it needs, so that partial improvements within a band are not lost to rounding.

Table 2: Quality bands of the LLM judge J, which selects a band and then reports a two-decimal score within it.

Band	Criterion
0.0–0.2	Incorrect or unrelated
0.2–0.4	Mostly incorrect, some overlap
0.4–0.6	Partially correct, missing details
0.6–0.8	Largely correct, minor omissions
0.8–1.0	Correct and complete

Final reward. The three components measure complementary aspects of an edit: its post-edit utility and utility change on q_t, and the utility of the edited memory on earlier and later queries. Their weighted sum gives the reward used in Equation 3:

$$R(a) = w_{\text{cur}} R_{\text{cur}}(a) + w_{\text{past}} R_{\text{past}}(a) + w_{\text{future}} R_{\text{future}}(a), \tag{10}$$

with nonnegative weights that sum to one (values in Appendix C). A component is dropped when its query set is empty.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on three long-term memory benchmarks. LoCoMo (Maharana et al., 2024) tests conversational memory with single-hop, multi-hop, open-domain, and temporal questions. LongMemEval (Wu et al., 2024) evaluates memory over extended interactions, and we use its oracle setting. MemoryAgentBench (Cherif, 2026) covers four capabilities: accurate retrieval, conflict resolution, long-range understanding, and test-time learning; we report its overall score weighted by

Table 3: **LoCoMo test results.** LLM-Judge scores (%) on the 1,307 QA pairs of the 8 held-out conversations. JanusMem is trained on 152 QA pairs from a single conversation.

Method	Single Hop	Multi Hop	Open Domain	Temporal	Overall
A-Mem	32.7	17.4	12.3	10.9	24.3
Mem0	27.4	25.3	13.9	20.1	24.7
MemoryOS	31.9	23.2	19.9	19.6	27.1
TreeMem	42.7	31.5	16.7	14.6	33.4
Memory-R1	40.4	28.0	19.3	18.6	32.5
JanusMem (Ours)	**54.2**	**36.4**	**26.5**	**28.7**	**44.1**

Table 4: **Zero-shot results on MemoryAgentBench and LongMemEval.** LLM-Judge scores (%). JanusMem uses the same checkpoint as in Table 3, without training on either benchmark.

Method	MemoryAgentBench					LongMemEval
	Accurate Retrieval	Conflict Resolution	Long-Range Understanding	Test-Time Learning	Overall	Overall
A-Mem	20.5	9.4	21.1	0.5	14.3	13.5
Mem0	17.8	10.2	19.9	0.7	12.9	17.7
MemoryOS	42.6	17.6	30.0	1.0	20.8	19.1
TreeMem	41.3	17.8	30.8	1.5	20.4	25.7
Memory-R1	**47.6**	**20.6**	33.8	1.7	25.9	28.8
JanusMem (Ours)	44.1	20.1	**52.1**	**5.2**	**28.8**	**36.4**

task size. We split LoCoMo by conversation: we train on 152 QA pairs from one conversation, validate on 81 QA pairs from another, and test on 1,307 QA pairs from the remaining 8 conversations. LongMemEval and MemoryAgentBench are never used for training.

Baselines. We compare with memory systems whose editing decisions are specified by prompts or rules, A-Mem (Xu et al., 2026), Mem0 (Chhikara et al., 2025), and MemoryOS (Kang et al., 2025), and with learning-based memory managers, TreeMem (Mao et al., 2026) and Memory-R1 (Yan et al., 2026b). Memory-R1 is our closest baseline: it uses the same Memory Manager and Answer Agent framework but trains the Memory Manager with an exact-match outcome reward.

Implementation. Both the Memory Manager and the Answer Agent are initialized from Qwen2.5-7B-Instruct (Qwen Team, 2024), and Qwen2.5-14B-Instruct serves as the judge J. Following Memory-R1, the Answer Agent is first trained with its own reward and then kept fixed while the Memory Manager is trained. Unless stated otherwise, JanusMem uses a bidirectional window with $K = 5$, group size $G = 128$, and reward weights $w_{\text{cur}} = 0.8$ and $w_{\text{past}} = w_{\text{future}} = 0.1$. All results for JanusMem across benchmarks come from a single checkpoint. Hyperparameters and training details are given in Appendix B.

Evaluation. We report LLM-Judge accuracy as the primary metric. During training, J provides continuous scores in $[0, 1]$ (Section 3.4); at test time, it assigns binary labels to fully correct answers. Exact-match (EM) and token-level F1 results are reported in Appendix D.

4.2 Main Results

LoCoMo. Trained on only 152 QA pairs, JanusMem achieves an overall LLM-Judge score of 44.1% on the 8 held-out conversations, outperforming the strongest baseline, TreeMem, by 10.7% and Memory-R1 by 11.6% (Table 3). It also leads in all four question categories. The gains are largest on single-hop and multi-hop questions, where JanusMem improves over Memory-R1 from 40.4% to 54.2% and from 28.0% to 36.4%, respectively. Answering these questions requires retrieving specific facts, or combining several of them, from conversations that span many sessions, so an answer fails whenever a needed fact was recorded incompletely or later overwritten. The bidirectional window targets exactly these failures: future queries reward recording all relevant details of a turn, and past queries penalize edits that erase facts that remain needed.

Zero-shot transfer. Without training on either benchmark, the same checkpoint achieves overall LLM-Judge scores of 28.8% on MemoryAgentBench and 36.4% on LongMemEval, outperforming

Table 5: **Backbone transfer.** LLM-Judge scores (%) with different backbones for both agents.

Backbone	Method	LoCoMo	LongMemEval	MemoryAgentBench
Llama3.2-3B-Instruct	Memory-R1	10.8	18.2	10.1
Llama3.2-3B-Instruct	JanusMem	36.7	37.8	15.5
Qwen2.5-7B-Instruct	JanusMem	44.1	36.4	28.8

Table 6: **Effect of the utility change.** LLM-Judge scores (%) without the bidirectional window. The last row reports absolute gains over post-edit utility alone.

Memory Manager reward	LoCoMo	LongMemEval	MemoryAgentBench
EM	32.5%	28.9%	22.5%
Post-edit utility	35.6%	31.9%	23.8%
Post-edit utility + utility change	**40.6%**	**37.4%**	**27.9%**
Gain from utility change	*+5.0*	*+5.5*	*+4.1*

all baselines (Table 4). The largest gain is on long-range understanding, where JanusMem improves over Memory-R1 from 33.8% to 52.1%. This task requires integrating information spread across a long input, which depends on the memory retaining a complete and consistent record rather than only the facts needed for the most recent query; this is the behavior that the bidirectional window rewards during training. JanusMem also exceeds Memory-R1 by 7.6% on LongMemEval, while Memory-R1 remains stronger on accurate retrieval and conflict resolution. These results indicate that an editing policy learned from a small LoCoMo training set transfers to distinct long-term memory benchmarks.

Other backbones. Table 5 repeats the comparison with Llama-3.2-3B-Instruct (Grattafiori et al., 2024) as the backbone for both agents. JanusMem improves over Memory-R1 by 25.9%, 19.6%, and 5.4% on LoCoMo, LongMemEval, and MemoryAgentBench, respectively. The gap is especially large on LoCoMo, where Memory-R1 reaches only 10.8%, indicating that its training barely improves the 3B model. This is expected from its exact-match reward: a small backbone rarely produces answers that exactly match the reference, so almost all candidate edits in a group receive the same zero reward, the group-normalized advantages vanish, and the policy receives little learning signal. The continuous scores of J instead credit partially correct answers and the utility change credits partial improvements, so candidate edits remain distinguishable even when no answer is fully correct. JanusMem thereby makes RL training effective for small backbones, reaching 36.7% on LoCoMo and outperforming the larger Qwen2.5-7B backbone on LongMemEval.

4.3 Ablation Studies

We ablate the three components of JanusMem in the order of Section 3.2–3.4: the utility change, the bidirectional window, and the judge granularity. We then study the GRPO group size.

Utility change. Table 6 compares Memory Manager rewards without the bidirectional window. Replacing exact match with the post-edit utility measured by J improves all three benchmarks, and adding the utility change further improves LLM-Judge scores by 5.0%, 5.5%, and 4.1% on LoCoMo, LongMemEval, and MemoryAgentBench, respectively. This comparison measures the combined effect of the utility change and its bounded transformation and does not isolate them.

Bidirectional window. We vary the window size K of the combined past and future window and average the LLM-Judge scores over the three benchmarks (Table 7). Base uses only R_{cur}, without a window. Relative to Base, $K = 1$ is slightly worse, whereas $K = 3$, $K = 5$, and $K = 8$ improve the mean score. Performance rises from $K = 1$ to a peak at $K = 5$ and then falls at $K = 8$, which reflects a bias–variance trade-off in estimating an edit's effect across time. A small window gives a high-variance estimate: with $K = 1$, the window term depends on a

Table 7: Mean LLM-Judge score over the three benchmarks for different window sizes K, and its difference from Base.

Window	Mean LLM-J	Δ vs. Base
Base	35.6%	–
$K = 1$	35.5%	−0.1
$K = 3$	36.5%	+0.9
$K = 5$	**36.8%**	**+1.2**
$K = 8$	35.7%	+0.1

single query per side, and whether that query happens to be affected by the edit is largely incidental. Accordingly, $K = 1$ improves LoCoMo and LongMemEval but degrades MemoryAgentBench, leaving the mean slightly below Base. A large window instead biases the estimate toward queries that are unrelated to the edit: queries whose evidence lies far from the current turn are rarely affected by it, so their scores barely differ across candidate edits and cancel under group normalization, while averaging over them dilutes the contribution of the few queries the edit actually affects. A moderate window balances variance and relevance, and since the cost also grows linearly in K, we use $K = 5$ by default.

Judge granularity. We compare three reward granularities: binary exact match (EM), LLM3, which restricts J to scores in $\{0, 0.5, 1\}$, and LLM5, which uses the five bands of Table 2 with two-decimal scores. We vary the granularity separately for training the Memory Manager and the Answer Agent, and use the post-edit utility alone, without the utility change or the window (Table 8). Finer rewards consistently improve performance: for every Answer Agent reward, a Memory Manager trained with LLM5 outperforms one trained with EM, and with LLM5 for the Answer Agent, the score rises from 35.3% with EM to 38.9% with LLM3 and 40.2% with LLM5. Finer rewards also make candidate edits distinguishable: in the corresponding Memory Manager training runs, the fraction of candidate groups receiving identical rewards falls from 85.0% with EM to 43.8% with LLM3 and 21.6% with LLM5. This resolution matters even more for JanusMem than for these post-edit-only rewards, because the utility change is a difference of two scores and vanishes under coarse measurement whenever both scores fall into the same level.

Table 8: LoCoMo LLM-Judge scores for different reward granularities of the Memory Manager (rows) and the Answer Agent (columns).

	Answer Agent reward		
Manager reward	**EM**	**LLM3**	**LLM5**
EM	32.5%	34.1%	35.3%
LLM3	34.2%	36.2%	38.9%
LLM5	34.8%	35.6%	**40.2%**

GRPO group size. We vary the number of candidate edits G sampled for each training context. As shown in Figure 3, the LoCoMo LLM-Judge score increases overall from 37.4% at $G = 8$ to 42.8% at $G = 128$, although intermediate group sizes do not follow a strictly monotonic trend. Larger groups provide more candidates for within-group comparison, which may help GRPO distinguish useful edits from ineffective ones; since the number of training steps also grows with G (Appendix E), this trend should be read together with the training schedule.

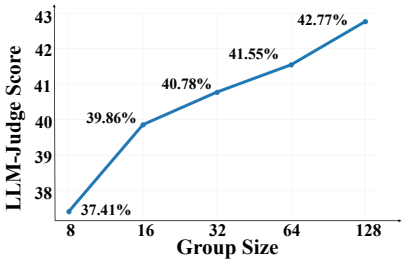

Figure 3: LoCoMo LLM-Judge scores under different GRPO group sizes.

5 Conclusion

We presented JanusMem, a reinforcement learning framework that learns long-term memory edits by measuring their actual effect across time. JanusMem scores each edit on the current query by both its post-edit utility and its utility change from the pre-edit memory, and a bidirectional window further credits improvement on nearby future queries and penalizes deterioration on nearby past queries, with all utilities measured by a fine-grained LLM judge. Trained on only 152 QA pairs from a single LoCoMo conversation, JanusMem achieves state-of-the-art performance on the held-out conversations and transfers zero-shot to LongMemEval and MemoryAgentBench, and its gains extend to a smaller backbone. These results show that measuring what an edit changes, rather than only the outcome it leads to, is an effective and data-efficient signal for learning to maintain long-term memory.

Discussion. Our results suggest that what the learning signal measures is one of the bottlenecks for learning memory editing: shifting from the utility of the edited memory to the utility change an edit causes lets JanusMem learn from only 152 QA pairs and train a 3B backbone that exact-match rewards barely improve. JanusMem also has limitations: it requires reference answers and queries annotated with evidence positions, inherits the biases and cost of the LLM judge ($2K+1$ judge calls per candidate edit), and, following Memory-R1, is trained on prescribed rather than self-generated memory trajectories. Promising directions include reference-free utility estimates, training on self-generated trajectories, relevance-based windows, and other forms of persistent agent state.

AI USE STATEMENT

In this work, we used generative AI tools for drafting and editing manuscript text. We have not used generative AI tools to generate experimental results, figures, or bibliographic entries. We have reviewed all AI-assisted writing. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve new data collection from human participants. Our experiments use existing research benchmarks:LoCoMo, LongMemEval, and MemoryAgentBench, and do not deploy the proposed system on live user conversations. Nevertheless, long-term memory systems can retain sensitive information, and erroneous or malicious ADD, UPDATE, or DELETE operations could introduce misleading content or remove information needed for reliable responses. Our benchmark results do not establish the safety of deploying the method in personal-assistant settings; such deployment would require appropriate privacy protections, access controls, and safeguards for memory modification. We use a fixed Qwen2.5-14B-Instruct judge for evaluation across methods, but its assessments may still reflect model-specific biases or errors and should not be treated as a substitute for independent validation.

REPRODUCIBILITY STATEMENT

We provide the algorithm and implementation details in Sections 4.1, including the memory-update procedure and reward computation. Appendix B.3 documents the datasets, preprocessing steps, evaluation protocol, baselines, hyperparameters, and compute setup. Additional ablations and analyses are reported in Appendix E.1.

REFERENCES

Ahmed Cherif. Agentmembench: A systematic benchmark for evaluating long-term memory management strategies in conversational ai agents. *arXiv preprint arXiv:2608.00009*, 2026.

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.

Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. In *International Conference on Learning Representations*, volume 2026, pp. 98706–98729, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25972–25981, 2025.

Weitao Ma, Xiaocheng Feng, Lei Huang, Xiachong Feng, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Bing Qin. Fine-mem: Fine-grained feedback alignment for long-horizon memory management. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19666–19684, 2026.

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

Marina Mao, Alexandr Liu, Pengbo Li, Siheng Li, Bo Zhou, and Xiang Wang. Tree-based credit assignment for multi-agent memory system. *arXiv preprint arXiv:2605.04811*, 2026.

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

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

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

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.

Sikuan Yan, Ahmed Bahloul, Ercong Nie, Susanna Schwarzmann, Riccardo Trivisonno, Volker Tresp, and Yunpu Ma. Memory-r2: Fair credit assignment for long-horizon memory-augmented llm agents. *arXiv preprint arXiv:2605.21768*, 2026a.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, 2026b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 21457–21483, 2026.

Yanwei Yue, Boci Peng, Xuanbo Fan, Jiaxin Guo, Qiankun Li, and Yan Zhang. Mem-t: Densifying rewards for long-horizon memory agents. *arXiv preprint arXiv:2601.23014*, 2026.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Bryan Kian Hsiang Low, and Paul Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pp. 58413–58438, 2026.

Appendix

Table of Contents

A Notation **13**

B Experimental Setup **13**

B.1 Models and System Components 13

B.2 Separate Training of the Two Agents 13

B.3 Data and Evaluation Protocols 14

B.4 Optimization and Environment 14

C Reward Implementation **14**

C.1 Scoring a Candidate Edit 14

C.2 Temporal Evaluation Scope 15

D Full Benchmark Results **15**

D.1 Overall Metrics 15

D.2 LoCoMo Question Categories 16

D.3 MemoryAgentBench Tasks 16

E Additional Analyses **16**

E.1 Temporal-Window Performance 16

E.2 Memory-Bank Size 16

E.3 Memory Operations on LoCoMo 17

E.4 GRPO Group Size 17

F Qualitative Analysis **17**

F.1 Distinguishing Evidence from Memory State 17

F.2 Illustrating a Small Positive Change 18

F.3 Failure Modes 18

G Prompt Templates **18**

G.1 Memory Manager 19

G.2 Answer Agent 21

G.3 Training and Test Judges 21

H Limitations **22**

Table 9: **Notation.** Symbols used in the method and appendix.

Symbol	Meaning
Memory and answering	
M_t, M_t^a	Pre-edit bank and its state after candidate edit a.
x_t, C_t	Current input and retrieved entries, $C_t \subseteq M_t$.
$a, \mathcal{T}$	Memory operation and state transition.
$\pi_\theta, \mathcal{A}$	Memory Manager policy and fixed Answer Agent.
q, q_t, y	Probe query, current query, and reference answer.
$J, s(M, q)$	Fixed LLM judge and answer-quality utility.
Reward and training	
$\Delta_t(a)$	Current-query utility change.
α, τ	Current-reward mixing weight and delta sensitivity.
K	Maximum number of nearby queries on each side.
$\mathcal{Q}_{\text{past}}$	Nearby past queries.
$\mathcal{Q}_{\text{future}}$	Nearby future queries.
R_{cur}	Current-query reward.
R_{past}	Past-query degradation penalty.
R_{future}	Future-query credit.
$w_{\text{cur}}, w_{\text{past}}, w_{\text{future}}$	Weights in the final reward.
$G, \hat{A}_i$	GRPO group size and normalized advantage.
Reference-replayed training state	
M_t^-, M_t^+	Pre-edit bank and bank after the candidate edit.
a_i^{ref}	Teacher-generated operation at turn i.
$s_t^-(z), s_t^+(z)$	Scores on query z before/after the edit.

A Notation

Table 9 summarizes the symbols in the method and appendix. During training, the reference-replayed pre-edit state M_t^- is the M_t used to compare candidate edits in Section 3.1; M_t^+ is the result of one candidate, denoted M_t^a in the method. At test time, the Memory Manager advances the bank with its own operations.

B Experimental Setup

B.1 Models and System Components

JanusMem comprises a Memory Manager (MM), which edits a persistent memory bank, and an Answer Agent (AA), which retrieves memories and answers questions. The primary policies are initialized from Qwen2.5-7B-Instruct. Additional experiments use Llama-3.2-3B-Instruct as the policy backbone. Memory retrieval uses `all-MiniLM-L6-v2` embeddings.

We use Qwen2.5-14B-Instruct as the LLM judge. During training, it supplies continuous semantic scores in $[0, 1]$ to two decimal places. During test evaluation, it assigns binary labels indicating whether an answer is fully correct. Thus, the training-time utility score and test-time LLM-Judge accuracy are distinct quantities.

B.2 Separate Training of the Two Agents

The MM and AA are optimized separately rather than trained end to end on a memory bank continually generated by the current MM policy. We first use GPT-4o-mini to generate reference memory operations for the training dialogue. Replaying these operations in temporal order constructs the reference memory bank used to prepare training contexts. The teacher-generated operations define the training environment; they should not be confused with human-annotated gold memory edits.

Answer Agent training. For each training QA pair, we retrieve relevant entries from the constructed reference memory bank and provide them, together with the question, to AA. AA learns

Table 10: **Data splits.** Conversation-level LoCoMo split.

Split	Conversations	QA pairs	Turns
Train	1	152	419
Validation	1	81	369
Test	8	1,307	5,094

to answer using this fixed memory input. MM does not participate in AA training, and AA is not trained on memory banks generated by the concurrently changing MM policy.

Memory Manager training. For a training turn t, we first replay the GPT-4o-mini reference operations from turns $1, \ldots, t-1$ to construct the memory state immediately before the current turn. The trained MM does not generate that pre-edit state. MM then observes the current turn and proposes a memory operation, which is applied to the prepared bank. A frozen AA answers the associated probe questions using the resulting bank, and the answer quality provides the MM reward. For the smooth-delta reward, the same frozen AA also answers those questions using the pre-edit bank.

Consequently, each candidate operation is evaluated from a fixed, teacher-constructed starting state:

$$M_t^- = \mathrm{Replay}\big(M_0, a_1^{\mathrm{ref}}, \ldots, a_{t-1}^{\mathrm{ref}}\big), \qquad M_t^+ = \mathcal{T}(M_t^-, a_t), \tag{11}$$

where a_i^{ref} is a teacher-generated reference operation and a_t is the candidate operation sampled from MM. At test time, in contrast, MM builds memory sequentially from its own generated operations. We keep these training and test procedures separate throughout the paper.

B.3 Data and Evaluation Protocols

LoCoMo. We split LoCoMo by conversation. One training conversation contains 152 evaluated QA pairs, one validation conversation contains 81, and eight held-out test conversations contain 1,307. Questions without standard reference answers are excluded from these counts.

The LoCoMo test split contains 727 single-hop, 239 multi-hop, 258 temporal, and 83 open-domain questions. MM considers up to 10 related existing memories when editing, and AA retrieves up to 60 memories when answering. The LoCoMo "Overall" result weights individual questions equally; a four-category macro average instead weights the four categories equally. We label these two aggregates separately.

Transfer benchmarks. We evaluate on LongMemEval and MemoryAgentBench without training on either benchmark. LongMemEval results use the 500-question oracle setting. MemoryAgentBench comprises Accurate Retrieval, Conflict Resolution, Long-Range Understanding, and Test-Time Learning. Its reported four-task macro average weights these tasks equally.

B.4 Optimization and Environment

The recorded primary runs use full-parameter group-relative policy optimization. Two NVIDIA H200 SXM GPUs host the policy-training ranks and a third hosts frozen AA and judge inference. The training environment uses Python 3.11, PyTorch 2.10.0+cu128, vLLM 0.19.1, TRL 1.9.1, Transformers 4.57.6, and Accelerate 1.14.0.

The recorded primary runs do not activate an SFT cold start; the MM SFT epoch setting is zero. Group size and the number of training steps vary in the corresponding analyses.

C Reward Implementation

C.1 Scoring a Candidate Edit

Let M_t^- be the pre-edit memory bank and M_t^+ the bank after a candidate action. For the same probe question z, the frozen AA produces an answer from each bank. The continuous judge scores these

Table 11: **Optimization settings.** Primary optimization settings.

Setting	Value
Learning rate	10^{-6}
Warmup	0 steps
Optimizer	`adamw_bnb_8bit`
Precision	`bf16`
Per-device batch size	2
Policy-training ranks	2
Gradient accumulation	32
Effective batch	128 completions per step
KL coefficient	0.01
Reward normalization	Group-wise
Policy sampling temperature	1.0
Maximum completion length	2,048 tokens
Maximum prompt length	4,096 tokens
Default random seed	42
Validation interval	10 steps

answers as $s_t^-(z)$ and $s_t^+(z)$. Comparing scores on the same question avoids conflating an edit's effect with differences in question difficulty.

The smooth utility-delta reward is

$$r_t(z) = \alpha s_t^+(z) + (1-\alpha)\tanh\left(\frac{s_t^+(z) - s_t^-(z)}{\tau}\right), \qquad \alpha = 0.7, \quad \tau = 0.25. \tag{12}$$

When $z = q_t$ and $M_t^- = M_t$, this per-query expression equals $R_{\text{cur}}(a_t)$ in Eq. 6. The first term values the resulting memory state; the second credits the change introduced by the candidate action. The hyperbolic tangent gives a smooth, bounded response to utility differences.

C.2 Temporal Evaluation Scope

The current component evaluates questions associated with the present turn. A past component checks whether an edit reduces performance on questions supported by earlier memory. A future component can measure its effect on questions associated with subsequent turns during training.

In the past-preservation configuration, the current component and past component have weights 0.8 and 0.2, respectively. The latter examines the most recent K eligible historical questions and penalizes degradation. In the combined past-and-future configuration, current, past, and future components have weights 0.8, 0.1, and 0.1. The future component provides training feedback; future dialogue content is not made available to MM as an observation at test time.

The window analysis considers $K \in \{1, 3, 5, 8\}$. We distinguish past preservation alone from combined past-and-future evaluation even when both use the same value of K.

D Full Benchmark Results

This section reports EM, token F1, and LLM-Judge accuracy for the principal comparisons. All table values are percentages unless noted otherwise. Consistent with the main paper, all headline JanusMem results across LoCoMo, LongMemEval, and MemoryAgentBench are evaluated from a single unified MM checkpoint (the past-and-future configuration with $K = 5$, trained solely on LoCoMo).

D.1 Overall Metrics

The comparison illustrates that gains in LLM-Judge accuracy need not coincide with gains in EM. For example, Memory-R1 has higher LoCoMo EM, whereas JanusMem has higher LoCoMo LLM-Judge accuracy.

Table 12: **Overall benchmark metrics.** Complete metrics for the reported main comparisons evaluated with a single unified JanusMem checkpoint across all benchmarks. LongMemEval uses the oracle setting.

Benchmark	Method	EM	F1	LLM-J
LoCoMo	Memory-R1 (EM–EM)	14.0	34.6	32.5
	JanusMem	6.0	28.5	44.1
LongMemEval	Memory-R1 (EM–EM)	16.4	26.6	28.8
	JanusMem	14.0	30.7	36.4
MemoryAgentBench	Memory-R1 (EM–EM)	16.9	24.0	25.9
	JanusMem	15.8	24.0	28.8

Table 13: **LoCoMo question categories.** LoCoMo LLM-Judge accuracy by question category for the primary JanusMem checkpoint.

Category	n	Memory-R1	JanusMem
Single hop	727	40.4	54.2
Multi hop	239	28.0	36.4
Open domain	83	19.3	26.5
Temporal	258	18.6	28.7
Instance-weighted overall	1,307	32.5	44.1

D.2 LoCoMo Question Categories

Single-hop questions account for 727 of the 1,307 test questions. We therefore show both category-level results and the equally weighted macro average alongside the instance-weighted overall score.

D.3 MemoryAgentBench Tasks

JanusMem's largest task-level advantage occurs on Long-Range Understanding. Memory-R1 remains stronger on Accurate Retrieval. Both systems have low absolute accuracy on Test-Time Learning.

E Additional Analyses

E.1 Temporal-Window Performance

Table 14 reports benchmark-level LLM-Judge accuracy for the past-preservation and combined past-and-future configurations across ablation runs.

The category-balanced cross-benchmark score is defined by

$$S_{\text{macro}} = \frac{1}{3}\sum_{b=1}^{3}\left[\frac{1}{|\mathcal{C}_b|}\sum_{c\in\mathcal{C}_b} S_{b,c}\right], \tag{13}$$

where $\mathcal{C}_b$ is the set of evaluated categories or tasks for benchmark b. This gives each category equal weight within its benchmark and gives each benchmark equal weight in the final aggregate.

Past preservation with $K = 3$ yields the highest reported category-balanced mean, 34.4%. Adding the future component changes performance differently across settings: it improves some individual categories but does not improve the $K = 3$ overall score on any of the three benchmarks shown here. We therefore do not interpret these results as evidence that the future component universally increases accuracy.

E.2 Memory-Bank Size

The values in Table 15 are sums of final memory-bank entries over all conversations or items in each benchmark setting. They are not memory sizes for one conversation. Because the benchmarks con-

Table 14: **Temporal-window performance.** LLM-Judge accuracy under temporal evaluation windows across ablation runs. LongMemEval uses the oracle setting. The final column first averages categories within each benchmark and then averages the three benchmarks equally.

Scope	K	LoCoMo	LongMemEval	MemoryAgentBench	Category-balanced mean
Past	1	42.1	33.0	27.7	31.7
Past + future	1	43.2	37.0	26.3	32.7
Past	3	45.1	38.0	28.8	**34.4**
Past + future	3	44.1	37.4	27.9	34.1
Past	5	41.6	36.0	28.8	33.0
Past + future	5	44.1	36.4	29.8	33.7
Past	8	40.9	38.0	27.4	33.0
Past + future	8	41.2	37.4	28.4	32.4

Table 15: **Memory-bank size.** Aggregate final memory-bank entries. LME denotes LongMemEval oracle; AMB abbreviations denote the four MemoryAgentBench tasks.

Scope	K	LoCoMo	LME	AMB-AR	AMB-CR	AMB-LRU	AMB-TTL
Past	1	4,581	10,278	10,751	26,279	45,441	35,388
Past + future	1	4,671	11,055	9,788	26,909	44,263	34,416
Past	3	4,768	11,140	10,436	27,848	45,141	38,424
Past + future	3	4,832	11,504	10,073	27,697	45,823	37,408
Past	5	4,614	10,791	10,835	26,873	45,415	35,938
Past + future	5	4,815	11,152	9,715	28,048	43,877	39,764
Past	8	4,651	11,212	10,303	27,311	44,348	37,550
Past + future	8	4,846	11,469	10,297	27,983	45,453	37,296

tain different numbers and types of items, raw totals should be compared only across configurations within the same benchmark column.

On LoCoMo, the eight test conversations have between 4,581 and 4,846 final memory entries in total across these settings, corresponding to roughly 573–606 entries per conversation. The changes are modest relative to the total bank size. Thus, a higher QA score should not automatically be attributed to storing more entries.

E.3 Memory Operations on LoCoMo

MM may emit multiple operations at one turn. We therefore report operation counts and the number of editing operations per evaluated turn, rather than treating one operation as equivalent to one dialogue turn. The NOOP percentage is computed over all output operations, including NOOP.

For example, the past-preservation configuration changes from 205 DELETE operations at $K = 1$ to 128 at $K = 3$, while final bank size increases from 4,581 to 4,768 entries. This pattern shows that fewer deletions do not require the manager to stop adding memories. Operation counts alone, however, cannot determine whether the avoided deletions would have been harmful; answering on historical questions is needed for that conclusion.

E.4 GRPO Group Size

The group-size results in Table 17 are not monotonic. Because the training schedule also changes with group size, they document the behavior of these complete configurations rather than the isolated causal effect of G.

F Qualitative Analysis

F.1 Distinguishing Evidence from Memory State

A qualitative edit trace must distinguish the source conversation from the memory bank actually used by the system. Ideally, each trace displays the source turn, the reference or inferred pre-edit bank, the candidate MM operation, the post-edit bank, the probe question, the two AA answers, and their scores. A source utterance alone is not a complete memory-bank snapshot.

Table 16: **Memory operations.** LoCoMo MM operation counts and editing operations per turn. Counts aggregate all eight test conversations.

Scope	K	ADD	UPDATE	DELETE	NOOP	NOOP (%)	Edits/turn
Past	1	4,786	1,663	205	47,872	87.8	1.31
Past + future	1	4,937	1,880	266	47,700	87.1	1.39
Past	3	4,896	1,396	128	48,372	88.3	1.26
Past + future	3	5,020	1,369	189	48,320	88.0	1.29
Past	5	4,831	1,317	218	48,322	88.4	1.25
Past + future	5	4,994	1,320	179	48,392	88.2	1.28
Past	8	4,782	1,111	131	48,650	89.0	1.18
Past + future	8	4,994	1,479	148	48,271	87.9	1.30

Table 17: **GRPO group size.** Recorded LoCoMo group-size runs under a schedule with varying training steps.

Group size	Training steps	LoCoMo LLM-J
8	50	37.4
16	100	39.9
32	200	43.2
64	400	41.5
128	800	42.8

F.2 Illustrating a Small Positive Change

Consider a LoCoMo question asking which games Jolene and her partner played together. Its reference answer contains several titles, including *It Takes Two*. An earlier source utterance describes their shared gaming experience without naming a game; a later utterance explicitly mentions playing *It Takes Two*.

If applying a candidate edit raises the score on that question from 0.000 to 0.140, its answer can still fail exact match because the full list is incomplete. Nevertheless, Eq. 12 assigns the corresponding per-question reward

$$0.7(0.140) + 0.3\tanh(0.140/0.25) \approx 0.250. \tag{14}$$

This is an illustration of how the reward responds to a small positive change. Unless the associated pre-edit and post-edit bank snapshots and generated AA answers have been independently verified, the two source utterances must not be presented as a fully logged empirical edit transition.

F.3 Failure Modes

An edit may store a correct fact that is not later retrieved, overwrite an older fact needed for a past question, or yield an answer with some correct details but missing required components. These failures occur at different stages: memory editing, retrieval, or answering. We do not attribute every incorrect downstream answer to MM alone.

G Prompt Templates

The following templates specify the primary MM and AA inputs and the judge instructions. Braced strings are runtime placeholders. Templates merely defined in the repository but unused in the reported experiments are omitted.

G.1 MEMORY MANAGER

Memory Manager prompt

Full-bank editing prompt.

```
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"}
        ]
        }

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"}
        ]
        }

## Old Memory
{related_memories}

## Retrieved Facts
{new_facts}

Respond with a JSON object containing the "memory" array.
```

The {related_memories} field contains related existing entries; {new_facts} contains incoming facts marked by speaker and timestamp.

G.2 Answer Agent

Answer Agent prompt

Answering with 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}

Question: {question}
```

Memories are grouped by speaker and retain timestamps. The final answer is extracted from the text following the `**Answer:**` marker.

G.3 Training and Test Judges

Continuous training judge

Semantic score used for smooth-delta training.

```
You are an impartial judge. Given a question, a gold (reference)
↪  answer, and a predicted answer, rate how correct the predicted
↪  answer is with a CONTINUOUS score between 0.00 and 1.00 (two
↪  decimal places).
```

```
Scoring rubric (pick the band, then place your exact score inside
↪  it):
- 0.00-0.19: wrong  contradicts the gold answer, misses the key
↪  information entirely, is unrelated, or is empty
- 0.20-0.39: mostly wrong  only incidental overlap with the gold
↪  answer
- 0.40-0.59: partially correct  some of the key information matches,
↪  but part is missing, imprecise, or slightly wrong
- 0.60-0.79: mostly correct  most of the key information matches;
↪  minor gaps or imprecision
- 0.80-1.00: fully correct  conveys the same key information as the
↪  gold answer (paraphrases, or different formats of the same
↪  date/number, still count as fully correct)

Question: {question}
Gold answer: {gold_answer}
Predicted answer: {predicted_answer}

Respond with exactly one number with two decimal places (e.g. 0.73):
```

Binary answer judge

Strict fully-correct evaluation.

```
You are an impartial binary judge. Given a question, a gold
↪  (reference) answer, and a predicted answer, decide whether the
↪  predicted answer is fully correct.

Scoring rubric (respond with exactly one value):
- 1: the predicted answer conveys the same key information as the
↪  gold answer; paraphrases and equivalent date/number formats count
↪  as correct
- 0: the answer is partially correct, incomplete, imprecise, wrong,
↪  contradictory, unrelated, or empty

Question: {question}
Gold answer: {gold_answer}
Predicted answer: {predicted_answer}

Score:
```

The continuous training score and binary test label have different interpretations and are not directly interchangeable.

H LIMITATIONS

Single-run evidence. The recorded experiments use one default random seed and do not include variance estimates from independent repeated training runs.

Edit-level evidence. Existing training artifacts contain aggregate reward statistics but do not consistently retain full before-and-after answers and judge scores for each candidate edit. Illustrative source utterances must not be presented as verified memory-state transitions without checking the actual bank snapshots and generated answers.

Task and metric variation. EM, F1, and LLM-Judge measure different answer properties and need not improve together. JanusMem also does not lead on every MemoryAgentBench task; absolute accuracy on Test-Time Learning remains low.

Training-state mismatch. MM is trained from reference memory states prepared by replaying teacher-generated operations, whereas at test time it acts on banks built from its own earlier de-

cisions. This difference can compound over long sequences and is a remaining limitation of the training protocol.