UNIMEM-R1: TOWARD GENERALIZABLE LONG-TERM MEMORY MANAGEMENT VIA CMI-AUGMENTED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory managers determine what persistent agents memorize from the long and noisy context. Recent reinforcement learning (RL) approaches train such managers with downstream question answering (QA) accuracy as the reward signal, encouraging appropriate management of question-relevant memories. These methods show the QA only rewarding system can make model identify relevant memories, but their sole reliance on a finite set of sampled questions offers only a partial view of memory value. As information that is not useful for a particular QA pair can still support users' future needs, we introduce **UniMem-R1**[1], a hybrid RL framework that complements extrinsic QA supervision with an intrinsic Conditional Mutual Information (CMI) reward. From an information theory perspective, CMI measures the information gain of the memory originated from the recent dialogue after conditioning on the existing memory bank. As a result, UniMem-R1 can jointly assess the new memory's task relevance and redundancy regarding the current memory. Experiments on LoCoMo, LongMemEval, and MemoryAgentBench show that UniMem-R1 outperforms RL baselines on matched sized backbone from 8.5% to 57.0% on each dataset's overall metric.

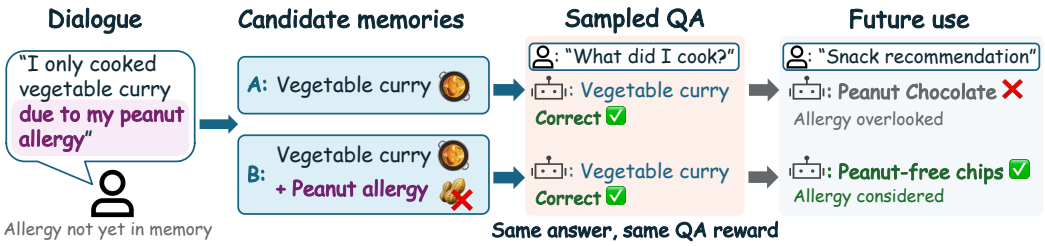


Figure 1: Illustration of hybrid memory supervision. QA assigns the same reward to Memories A and B because both answer the sampled question correctly. The memory intrinsic reward complements QA by rewarding memory B for retaining relevant allergy information.

1 INTRODUCTION

Large Language Models (LLMs) perform well on many real-world tasks Yang et al. (2025); DeepSeek-AI et al. (2025), yet they do not retain a persistent record of past interactions by default Wu et al. (2025). To address this, long-term memory systems were designed, to preserve user related knowledge across sessions to support chat continuity and personalization Chhikara et al. (2025); Li et al. (2025b). In this scenario, a noise in memory management can propagate and persist across future interactions. Robust memory management is essential for agentic systems, since omitting critical information inevitably leads to cascading errors and degraded task performance Yan et al. (2025); Yu et al. (2026); Shen et al. (2026); Ma et al. (2026); Kang et al. (2025).

Recent works leverage question answering to guide the manager in selectively recording task-relevant historical information Yan et al. (2025); Ma et al. (2026); Shen et al. (2026). If the memory

[1]Code is available at `https://anonymous.4open.science/r/CMIMem-FA30`.

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.

constructed supports correct answers, the manager's all actions receive the same positive reward. This signal teaches the manager to capture question relevant contents, anchors learning to end-task utility and achieve promising performance. However, these methods inherently neglect information irrelevant to observed questions, leaving critical context unrecorded for future novel queries. Increasing the number of synthetic questions can improve QA pattern coverage, but the reward still remains determined by what those questions happen to ask Casula et al. (2024); Geng et al. (2025). Since a memory item with no immediate value for a particular QA pair can still support an unseen question, a future preference, a summary, or an associative cue Xu et al. (2022); Zhang et al. (2026a); Marko et al. (2024). Solely rely on QA for memory manager training may fail to generalize to new tasks or domains.

We use an example in Figure 1 to illustrate this issue. While discussing a recipe, a user mentions a peanut allergy, and the sampled question only asks which dish the user cooked. Two memories can answer this question equally well, even if one of them totally ignores the allergy. In this scenario, solely relying on the QA outcome leaves the allergy unrewarded, makes the memory manager ignore the allergy. As a result, the memory manager may later recommend some recipes that contain peanuts, even though the user has clearly mentioned the peanut allergy.

Hence, a good rewarding system for memory manager training should consider hybrid supervision that combines a memorization evaluation with the downstream QA evaluation. On the one hand, QA teaches managers to retain evidence relevant to downstream tasks. On the other hand, a complementary memorization criterion should evaluate how well each memory action preserves the recent dialogue turns and contributes new content to the existing memory bank. A useful candidate should explain content in the dialogue while also contributing information that the bank does not already contain. Memory relevance alone would lead to redundancy, whereas memory novelty alone could reward unrelated noise. Such a criterion should assess both properties jointly, leaving QA to determine downstream task-level usefulness.

Motivated by this novel insight, we propose **UniMem-R1**, a reinforcement learning framework that augments QA-based extrinsic reward with intrinsic supervision grounded in **Conditional Mutual Information (CMI)** Wyner (1978); Li et al. (2025a). CMI originates from information theory, where it measures the dependence between two variables once a third variable is known Wyner (1978). Similar information-theoretic objectives have been adopted as learning signals for LLMs across a range of tasks. (e.g., fine-grained reward design for tool learning Qian et al. (2026), entropy-based rewards for unsupervised reasoning Zhang et al. (2026b), and conditional dependency modeling for multimodal integration Li et al. (2025a)). In our setting, CMI measures how much information a candidate memory provides about the current dialogue after conditioning on the existing memory bank. A high CMI value identifies content that is relevant to the dialogue and not already explained by stored memories. A low value indicates a redundant or irrelevant candidate. The resulting signal directly supervises extraction together with the sampled QA queries, combines QA correctness at the trajectory level with per-operation CMI feedback.

We summarize our contributions as follows:

- We identify the query-conditioned view of memory value as an underexplored limitation of learned memory managers.
- We introduce an action-conditioned CMI reward that jointly measures memories' relevance to the current dialogue and novelty with respect to the existing memory bank, providing dense supervision for individual memory operations.
- We unify CMI and QA rewards in an RL framework that improves memory manager's effectiveness as well as its cross-task generalization ability. According to our experiments, UniMem-R1 outperforms matched 4B RL baselines from 8.5% to 57.0% across three long-term memory benchmarks that contain diverse tasks.

2 Related Work

2.1 Training-free Memory Systems

The training-free Memory Systems organize memories without a trained manager. Among this category, MemOS Li et al. (2025b) treats memory as a schedulable system resource; AriGraph Anokhin

et al. (2024) and Zep Rasmussen et al. (2025) organize memories as knowledge graphs; and Mem0 Chhikara et al. (2025) combines LLM reasoning, prompts, and retrieval to extract, deduplicate, and update memories. These methods organize LLM memory through prompts, rules, or system design, and usually rely on powerful flagship LLMs (such as Qwen-3.7-Max Qwen (2026)) for effectiveness. Consequently, these methods can be expensive to deploy, and their handcrafted memory policies may not generalize to new tasks or domains.

2.2 Fine-tuning-based Memory Managers

Among this category, some methods choose to personalize LLM memories into model parameters. For example, PersonalLLM Zollo et al. (2024), PENSIEVE Jiang et al. (2025), and Yo'LLaVA Nguyen et al. (2024) adapt an answer model to user-specific data or subjects. Such parameterized memory can support personalization without an external store, but per-user fine-tuning is difficult to scale and can suffer from catastrophic forgetting Luo et al. (2025). Other methods train a standalone memory manager from downstream outcomes with RL techniques. Memory-R1 Yan et al. (2025) uses memory-related QA correctness scored by an LLM judge; AgeMem Yu et al. (2026) combines management rules with judge feedback; FineMem Ma et al. (2026) and MemBuilder Shen et al. (2026) construct denser rewards from session-specific QA pairs. However, current RL-based managers rely solely on QA correctness, although the QA correctness can ground learning in downstream tasks and teaches the manager to retain relevant evidence. The supervision remains tied to the sampled questions and downstream evaluator, so information not exercised by those questions may receive no positive signal.

2.3 Information-theoretic intrinsic rewards

Information-theoretic signals have recently been used as dense, intrinsic rewards that supervise the internal process of LLMs. Representative uses include turn-level information gain for multi-turn agents Wang et al. (2026a), episode-wise information gain for efficient reasoning Wang et al. (2026b), and reward design for tool use and unsupervised reasoning Qian et al. (2026); Zhang et al. (2026b). Unlike these scenarios, where information gain is measured toward a certain task answer, in memory management, the value of new memory content must be judged against an evolving memory bank independent of any sampled query, rewarding content that may be crucial for future usages. Analogously, human memory preferentially encodes novel or surprising information relative to prior knowledge Sobczak et al. (2025); Yin et al. (2025). Inspired by this principle, a useful LLM memory must also capture the current dialogue while excluding information already explained by the existing memory bank. Across the approaches above, this operation-level conditional information gain remains underexplored. In this paper, our UniMem-R1 introduces a hybrid rewarding system, which combines the information-theoretic intrinsic CMI reward with the downstream QA supervision. This rewarding system can provide dense supervision for individual memory operations, improving the trained memory manager's effectiveness and cross-task generalization ability.

3 Method

3.1 Conditional Mutual Information

We seek a criterion that identifies information relevant to the current dialogue but not already contained in memory. Conditional Mutual Information (CMI) provides this conditional view. For random variables X, Y, and Z with joint distribution p, CMI is defined as Wyner (1978)

$$I(X;Y \mid Z) = \mathbb{E}_{p(x,y,z)}\left[\log \frac{p(x,y \mid z)}{p(x \mid z)p(y \mid z)}\right] = H(X \mid Z) - H(X \mid Y,Z). \tag{1}$$

Here, lowercase x, y, and z denote realizations; $\mathbb{E}_{p(x,y,z)}$ is expectation under their joint distribution; and $H(\cdot \mid \cdot)$ denotes conditional entropy. The entropy form thus interprets CMI as the expected reduction in uncertainty about X obtained from Y after Z is known. A high value means that X and Y share substantial information not explained by Z, whereas $I(X;Y \mid Z) = 0$ indicates conditional independence. CMI and closely related information-gain objectives have been applied to conditional-dependence modeling, multimodal representation learning, exploration, and reward design Li et al. (2025a); Houthooft et al. (2016); Zhang et al. (2026b); Qian et al. (2026).

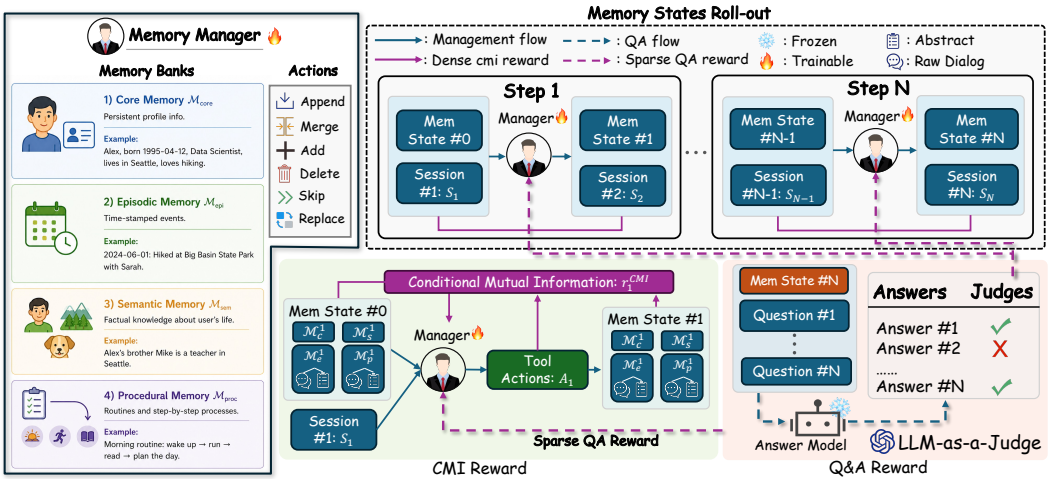


Figure 2: UniMem-R1 combines an extrinsic QA reward with an intrinsic per-operation CMI reward. QA provides end-task grounding, while CMI values information contributed to the evolving memory without conditioning on a sampled question.

By separating shared information from what the conditioning context already explains, CMI offers a basis for evaluating the informational contribution of a memory candidate beyond its usefulness for a sampled question.

3.2 CMI-Guided Memory Valuation

To directly supervise information extraction beyond the sampled questions, we score each candidate by the dialogue information it contributes relative to the current bank. At session t, the policy π_θ observes the dialogue C_t and existing memory M_{t-1}, then emits an operation a_t and a candidate fragment m_t^{new}. The formal memory state and action spaces are given in Appendix A.1.

In Eq. (1), we take the dialogue as X, the candidate as Y, and the existing memory as the conditioning context Z. This asks how much additional information the candidate provides about the dialogue once the stored information is known. As summarized in Figure 2, we estimate this contribution for each operation and combine it with QA supervision. The ideal intrinsic reward is

$$r_t^{\text{CMI}} = I\big(C_t; m_t^{\text{new}} \mid M_{t-1}^{\text{local}}\big), \tag{2}$$

where I denotes CMI, C_t is the dialogue at session t, m_t^{new} is the candidate fragment, and $M_{t-1}^{\text{local}} \subset M_{t-1}$ contains the most relevant entries from the existing memory bank M_{t-1}. Conditioning removes information already predictable from the bank, while the remaining dependence between C_t and m_t^{new} measures how well the candidate extracts previously uncovered dialogue content. Equation (2) therefore evaluates relevance and non-redundancy jointly rather than rewarding either property in isolation.

We describe a candidate as having high information gain relative to the bank when it captures dialogue information not already explained by that bank. In the example in Figure 1 in Section 1, recording the user's peanut allergy preserves dialogue information beyond the sampled question about the dish. On the one hand, if the allergy is not already stored, recording it adds evidence for future snack or restaurant recommendations. On the other hand, if the allergy is already stored, repeating it contributes little additional information. The same candidate can therefore have different informational value in different memory states.

Residual-Projection Estimator. For efficient training, we approximate Eq. (2) in a frozen embedding space. Under a scalar jointly Gaussian model, CMI is monotone in squared partial correlation:

$$I(C; m_{\text{new}} \mid M) = -\tfrac{1}{2}\log\big(1 - \rho^2_{C,m_{\text{new}}|M}\big), \tag{3}$$

where C, m_{new}, and M abbreviate the session context, candidate fragment, and local conditioning memory, respectively, and $\rho_{C,m_{\text{new}}|M}$ is their partial correlation Schober et al. (2018). This identity

motivates removing memory-predictable directions before scoring a candidate. Orthogonal residuals isolate the context information not covered by memory and the candidate content not predictable from memory. With $\mathbf{e}_C, \mathbf{e}_{\text{new}} \in \mathbb{R}^d$ denoting L2-normalized context and fragment embeddings, and $\mathbf{M} \in \mathbb{R}^{n_t^{\text{local}} \times d}$ stacking the n_t^{local} retrieved memory embeddings by row, we compute

$$\mathbf{r}_C = \mathbf{e}_C - \mathbf{M}^\top \left(\mathbf{M}\mathbf{M}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{M}\mathbf{e}_C, \tag{4}$$

$$\mathbf{r}_{\text{new}} = \mathbf{e}_{\text{new}} - \mathbf{M}^\top \left(\mathbf{M}\mathbf{M}^\top + \lambda \mathbf{I}\right)^{-1} \mathbf{M}\mathbf{e}_{\text{new}}, \tag{5}$$

where $\mathbf{r}_C$ and $\mathbf{r}_{\text{new}}$ are the context and candidate residuals, $\mathbf{I}$ is the $n_t^{\text{local}} \times n_t^{\text{local}}$ identity matrix, and the Tikhonov coefficient $\lambda > 0$ prevents ill-conditioning when local memories are correlated Calvetti et al. (2000). Our efficient estimator is the residual cosine

$$\widehat{r}_t^{\text{CMI}} = \frac{\mathbf{r}_C \cdot \mathbf{r}_{\text{new}}}{\|\mathbf{r}_C\| \, \|\mathbf{r}_{\text{new}}\|}. \tag{6}$$

Here, $\widehat{r}_t^{\text{CMI}}$ is the signed fragment score at session t, $\mathbf{r}_C$ and $\mathbf{r}_{\text{new}}$ are the context and candidate residuals, $\cdot$ denotes their inner product, and $\|\cdot\|$ is the Euclidean norm. For non-negative association, residual cosine follows the Gaussian-model ordering in squared partial correlation, although it is not calibrated in nats. Operation-specific scoring and reward shaping are applied below. The computation requires only embedding and projection. We also compare log-probability and direct-embedding variants in Table 3; their definitions are given in Appendix F.

Local Conditioning Set. The retrieval query concatenates the session context, candidate fragment, and any existing memory targeted by the operation. If the local conditioning set is empty, the raw operation score is $\max(0, \cos(\mathbf{e}_C, \mathbf{e}_{\text{new}}))$, the non-negative cosine similarity between the context embedding $\mathbf{e}_C$ and candidate embedding $\mathbf{e}_{\text{new}}$. Appendix G gives the exact construction.

Operation-Specific Scoring. For ADD and MERGE, the raw operation score is $\max(0, \widehat{r}_t^{\text{CMI}})$, where $\widehat{r}_t^{\text{CMI}}$ is the signed fragment score at session t. For REPLACE and UPDATE, it is the difference between the new and old fragment scores:

$$\Delta\widehat{r}_t^{\text{CMI}} = \widehat{r}_t^{\text{CMI}}(m_{\text{new}}) - \widehat{r}_t^{\text{CMI}}(m_{\text{old}}). \tag{7}$$

Here, $\Delta\widehat{r}_t^{\text{CMI}}$ is the raw replacement score at session t, m_{new} and m_{old} are the proposed and replaced fragments, and $\widehat{r}_t^{\text{CMI}}(\cdot)$ evaluates either fragment against the same session and conditioning memory. A positive difference indicates increased conditional association with the dialogue; a negative difference indicates a decrease. These raw scores provide operation-level information feedback beyond the sampled QA pairs and are regularized before entering the training reward.

3.3 HYBRID EXTRINSIC-INTRINSIC REWARD

Intrinsic information extraction alone cannot guarantee downstream utility, so we retain QA correctness as an extrinsic outcome anchor. Following Shen et al. (2026), session C_t has J pre-generated QA pairs $\mathcal{Q}_t = \{(q_{t,j}, y^\star_{t,j})\}_{j=1}^J$, where $q_{t,j}$ and $y^\star_{t,j}$ are the j-th question and gold answer. Given the rollout-updated state M_t, a fixed reader predicts $\hat{y}_{t,j} = \text{Answer}(q_{t,j}; M_t)$, and a judge evaluates correctness. Averaging over the J QA pairs for session t, the QA reward is

$$r_t^{\text{QA}} = \frac{1}{J} \sum_{j=1}^{J} \mathbb{1}\left[\text{Judge}(q_{t,j}, \hat{y}_{t,j}, y^\star_{t,j}) = \text{Correct}\right]. \tag{8}$$

Here, $q_{t,j}$, $y^\star_{t,j}$, and $\hat{y}_{t,j}$ are the j-th question, gold answer, and reader prediction, respectively; Judge returns a correctness label; Correct denotes a correct answer; and $\mathbb{1}[\cdot]$ is the indicator function. We map each non-zero raw operation score through a Gaussian target band:

$$\widetilde{r}_{t,k}^{\text{CMI}} = \exp\left[-\frac{(s_{t,k} - \mu)^2}{2\sigma^2}\right]. \tag{9}$$

Here, $s_{t,k}$ is the raw score for operation k in session t from the operation-specific scoring above, $\widetilde{r}_{t,k}^{\text{CMI}}$ is its shaped reward, μ is the target center, and $\sigma > 0$ controls the bandwidth. Zero raw scores retain zero reward. We use $\mu = 0.35$ and $\sigma = 0.15$, selected through reward-distribution monitoring

during training. This mapping favors scores near the target band, so a larger raw score does not always yield a larger training reward. Combining the mean $\bar{r}_t^{\text{CMI}}$ of shaped operation rewards $\widetilde{r}_{t,k}^{\text{CMI}}$ within session t with QA correctness r_t^{QA} gives

$$r_t = \alpha\, \bar{r}_t^{\text{CMI}} + (1-\alpha) r_t^{\text{QA}}, \qquad \alpha \in [0,1]. \tag{10}$$

Here, r_t is the composite reward at session t, and α controls the relative contribution of intrinsic CMI feedback and extrinsic QA correctness. The CMI component supplies per-operation credit in long-horizon rollouts Kazemnejad et al. (2025), whereas QA supplies trajectory-level task grounding. We additionally retain the format, length, duplication, and contribution-aware attribution regularizers detailed in Appendix D. This combination evaluates both the information retained by individual operations and the usefulness of the resulting bank for downstream answers.

3.4 Reinforcement Learning with GRPO

We aggregate per-operation CMI with session-level QA into a rollout reward and optimize π_θ using session-level GRPO DeepSeek-AI et al. (2025). Under this setting, the CMI reward can provide dense feedback to each memory action, while the QA reward can anchor learning to downstream task utility, achieving hybrid supervision that values information beyond the sampled questions. For session C_t, we sample N structured rollouts covering all memory types. For rollout $i \in \{1, \ldots, N\}$, r_i is its composite reward and $\mathbb{1}[\text{valid}_i]$ indicates whether its structured output is valid. The group-normalized advantage is $A_i = (r_i \mathbb{1}[\text{valid}_i] - \mu_g)/(\sigma_g + \epsilon)$, where μ_g and σ_g are the group mean and standard deviation and $\epsilon > 0$ ensures numerical stability. For the policy π_θ with trainable parameters θ and rollout advantage A_i, we minimize the clipped surrogate loss $\mathcal{L}(\theta)$ below, where $\mathbb{E}$ averages over sampled rollouts and β weights the KL divergence D_{KL} from the frozen reference policy π_{ref}:

$$\mathcal{L}(\theta) = -\mathbb{E}[\min(\rho_i A_i, \text{clip}(\rho_i, 1-\epsilon_l, 1+\epsilon_h) A_i)] + \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}), \tag{11}$$

where x_i and a_i are the prompt and sampled structured action for rollout i, $\rho_i = \pi_\theta(a_i \mid x_i)/\pi_{\text{old}}(a_i \mid x_i)$ is the likelihood ratio against the behavior policy π_{old}, and ϵ_l and ϵ_h are asymmetric clipping bounds. We follow the baseline GRPO procedure Shen et al. (2026); our contribution is the hybrid reward and its operation-level CMI component. In our training, we further use a stratified easy-to-hard curriculum during RL (Details are shown in Appendix C.1) to warm up RL training.

4 Experiments

4.1 Experimental Setup

Datasets. We carry out experiments on **LongMemEval**'s Wu et al. (2025) s_cleaned split, whose questions span recall, cross-session aggregation, and temporal reasoning; **LoCoMo** Maharana et al. (2024), which contains long human conversations; and **MemoryAgentBench** Hu et al. (2026), which tests long-range understanding and selective forgetting through summarization, recommendation, and open-ended questions (Appendix K). Following Shen et al. (2026), we train on the same LongMemEval RL split and report its held-out test split. LoCoMo is not used for training and serves as an out-of-distribution benchmark.

Table 1: Main results on LoCoMo, LongMemEval (LME-S), and MemoryAgentBench (MAB). Bold/underline denote best/second best; DSV4 denotes DeepSeek-V4-Flash-Preview.

Method	LoCoMo	LME-S	MAB
RAG Methods			
RAG (top-5)	50.5	50.0	24.5
RAG (top-10)	49.3	50.5	30.7
Prompting Methods			
Mem0	51.6	47.0	37.2
MemBuilder-P (DSV4)	71.0	82.0	44.1
MemBuilder-P (Qwen3.7-Max)	71.9	82.0	47.0
Ours-P (DSV4)	71.0	**83.0**	<u>51.3</u>
Ours-P (Qwen3.7-Max)	<u>72.4</u>	<u>82.5</u>	49.1
RL-based Methods			
Memory-R1-4B	62.7	60.8	42.3
MemBuilder-RL-4B	68.1	56.5	35.1
UniMem-R1-4B	**73.9**	68.8	**55.1**

Baselines. We compare three families. Retrieval-only **RAG** (top-5/10) feeds FAISS-retrieved dialogue turns directly to the answer model. Training-free baselines include **Mem0** Chhikara et al. (2025) [2] , **MemBuilder-P** Shen et al. (2026) reproduced with the

[2]Scores of Mem0 on LoCoMo and LongMemEval, and Memory-R1 on LoCoMo follow Shen et al. (2026) for a controlled comparison.

Table 2: Performance comparison of tested baselines on MemoryAgentBench. Category averages are computed from their subtask scores and rounded to one decimal; Overall is the macro-average of the four unrounded top-level category scores (AR, TTL, LRU, and SF). The metrics used are listed in Table 17. All systems use Qwen3.7-Max as both the answer and judge LLM.

			AR				TTL			LRU		SF	Overall
Agent Type	SH-QA	MH-QA	LME(S*)	EventQA	Avg.	MCC	Recom.	Avg.	Summ.	DetQA	Avg.	FC-SH	Scores
RAG Methods													
RAG (top-5)	38.0	19.0	58.0	50.0	41.3	12.8	0.8	6.8	13.5	56.3	34.9	15.0	24.5
RAG (top-10)	52.0	23.0	61.3	61.0	49.3	13.6	1.4	7.5	**16.1**	**71.8**	**44.0**	22.0	30.7
Prompting Methods													
Mem0	78.0	65.0	53.3	**81.6**	69.5	2.0	12.9	7.5	15.6	66.2	40.9	31.0	37.2
MemBuilder-P (Qwen3.7-Max)	88.0	72.0	70.0	78.6	77.2	43.6	17.2	30.4	14.6	57.7	36.2	44.2	47.0
MemBuilder-P (DSV4)	**89.0**	75.0	69.3	81.2	78.6	53.6	16.2	34.9	14.8	46.5	30.7	32.2	44.1
Ours-P (DSV4)	87.0	**76.0**	**74.0**	79.8	**79.2**	84.4	17.3	50.9	15.0	63.4	39.2	36.0	51.3
Ours-P (Qwen3.7-Max)	**89.0**	72.0	72.7	78.4	78.0	87.0	14.3	50.7	15.0	64.8	39.9	28.0	49.1
RL-based Methods													
Memory-R1-4B	66.1	53.9	60.6	76.9	64.4	63.2	11.2	37.2	13.9	65.5	39.7	28.0	42.3
MemBuilder-RL-4B	78.0	65.0	45.3	77.0	66.3	4.2	15.3	9.8	14.9	49.3	32.1	32.2	35.1
MemBuilder-RL-8B	55.0	72.0	67.0	74.0	67.0	68.2	16.1	42.2	15.1	56.3	35.7	62.0	51.7
UniMem-R1-4B	79.0	66.0	60.3	75.4	70.2	**88.0**	**17.8**	**52.9**	15.3	63.4	39.4	58.0	55.1
UniMem-R1-8B	85.0	67.0	61.0	72.4	71.4	85.8	16.1	51.0	15.1	62.0	38.6	**66.0**	**56.7**

prompting backbones listed in Tables 1 and 2, and **Ours-P**, a training-free prompting baseline that applies explicit memory-quality principles to our retrieval-indexed architecture without RL. Ours-P is a lightweight, training-free reference rather than an RL reward ablation. RL baselines include **Memory-R1**, the SFT-free **MemBuilder-RL** reimplementation retaining its ADRPO training method, and **UniMem-R1**. Matched RL methods use the same LongMemEval split and Qwen3-4B-Instruct-2507 backbone.

Memory Manager Configuration. Following prior structured-memory systems Sumers et al. (2023); Shen et al. (2026), we use a bank with Core, Episodic, Semantic, and Procedural memory and let four type-specific agents share a single policy. Retrieval-indexed memory (RM) additionally links fragments to their source dialogue turns for retrieval-time grounding. The 68.8 LongMemEval result in Table 1 uses CMI+QA without RM. Appendix A.1 and Algorithms 1–2 give the formal state transition, per-type action spaces, and update and retrieval procedures.

Implementation. The LLM backbone for UniMem-R1-4B, Memory-R1-4B, and MemBuilder-RL-4B are obtained by RL post-training Qwen3-4B-Instruct-2507 Yang et al. (2025), whereas UniMem-R1-8B and MemBuilder-RL-8B uses Qwen3-8B as backbone. Qwen3-Embedding-0.6B Zhang et al. (2025) is the shared encoder for retrieval and the CMI estimator. We use Qwen3.7-Max to generate the synthetic QA pairs, serve as the QA answerer and correctness judge for the extrinsic QA reward. We optimize UniMem-R1 with GRPO Shao et al. (2024) on one 8×H20 96GB node and use an empirical target-band regularizer (Ablations and settings are shown in Appendix D).

4.2 Results and Analysis

Table 1 summarizes results on LoCoMo, LongMemEval (s_cleaned), and MemoryAgentBench. For MemoryAgentBench, we replace substring exact match with LLM-as-Judge because substring matching can reward verbose outputs that merely contain the gold answer (Appendix K).

4.2.1 Memory Management vs RAG

RAG generally underperforms structured prompting and RL methods, although its broad passage coverage remains useful for summarization in the Long-Range Understanding (LRU) category, which requires integrating information across long histories Wang et al. (2026c). Ours-P matches or outperforms MemBuilder-P across the tested prompting backbones despite using the same four memory partitions. Our episodic and procedural entries additionally retrieve their source dialogue turns, enabling provenance-aware downstream QA. This advantage is largest on Multi-Class Classification (MCC), which requires predicting class labels from examples accumulated in the interaction history. Retrieving source turns preserves the example–label pairs that compressed memories may omit. Single-hop Fact Consolidation (FC-SH) tests whether answers reflect updated facts, including counterfactual corrections. Here, the shared reader may override stored facts with parametric knowledge, so the score reflects answer-model behavior as well as memory fidelity (Appendix K.1).

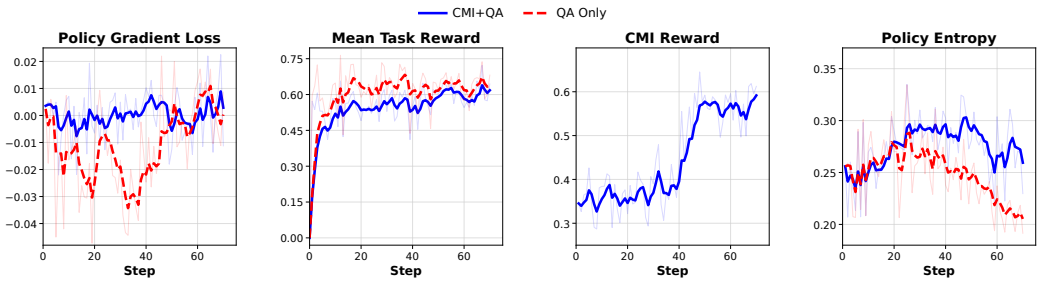


Figure 3: Training dynamics over 70 steps (left to right): policy gradient loss, mean task reward (QA accuracy), CMI reward, and policy entropy.

4.2.2 CMI & QA Reward Complementarity

QA evaluates memory through sampled questions, while CMI also values dialogue information beyond what those questions cover. After training on LongMemEval, UniMem-R1-4B surpasses the controlled MemBuilder-RL baseline by 20.0 points overall on MemoryAgentBench, with gains in test-time learning and long-range understanding. This result supports the cross-task generalization of the combined system, while Table 3 examines the reward components on LongMemEval. On LoCoMo and MemoryAgentBench, UniMem-R1-4B (only trained on LongMemEval) can even outperform the Qwen3.7-Max LLM with the same memory structure and prompt (Ours-P), with much fewer parameters. At 8B level, UniMem-R1 beats all other RL baselines overall (56.7 vs 51.7) and even the Qwen3.7-Max (49.1) with the same memory structure and prompt. These comparisons show how the trained manager performs relative to training-free systems with stronger proprietary LLMs. We further examine how the reward design changes constructed memories through five held-out LongMemEval cases in Table 4, with full traces in Appendix J.

4.2.3 Training Dynamics

Figure 3 compares CMI+QA with QA-only training. The CMI+QA policy-gradient curve remains centered near zero, whereas QA-only drifts negative. Task reward is non-monotonic during training as session difficulty varies under the curriculum described in Appendix C.1. CMI+QA responses average 39% of the QA-only token length, while the mean evaluation-time memory footprint decreases from 454.1K to 420.2K characters, driven largely by a smaller procedural store (138.2K to 84.4K). Together with length and duplication regularization, the combined objective pairs improved QA performance with lower generation and storage requirements. Appendix C.3 provides the response-length curve and type-wise memory statistics.

4.3 Ablation Experiments

On the same Qwen3-4B backbone and answering pipeline, we examine how QA and CMI supervision contribute individually and jointly to memory management. We report overall accuracy and performance on each LongMemEval task category in Table 3.

Retrieval-indexed memory aids factual recall. Prompting-only retrieval-indexed memory (+RM) raises overall accuracy from 53.3 to 55.0, with the largest gain on Knowledge Update. Retrieving source turns gives the reader access to factual details that compressed memory entries may omit, helping recover the evidence needed for downstream answers.

QA supervision under retrieval-indexed memory. The +RM+QA variant yields 53.3 overall, compared with 55.0 for RM without RL and 56.5 for QA without RM. This result shows that the benefit of QA supervision also depends on the memory configuration. Retrieval provides access to more source evidence, but QA-only training still evaluates memory through sampled questions, leaving the bank-relative information contributed by each fragment without direct supervision.

CMI alone lacks task grounding. CMI-only training reaches 45.0 overall accuracy, compared with 68.8 when combined with QA. CMI evaluates dialogue relevance and novelty, while QA assesses whether the resulting memory supports correct answers. This result indicates that memory intrinsic valuation also needs downstream task feedback to guide memory manager training.

Table 4: Representative case studies on LongMemEval (details in Appendix J). KU/MS/SS-P/TR denote Knowledge Update/Multi-session/Single-session Preference/Temporal Reasoning queries. Memory Size Diff. is query-time total entries difference between QA-only and CMI+QA.

Type	Question	Key Memory (QA → CMI+QA)	Memory Size Diff.	Answer (QA → CMI+QA)
KU	Stars for Starbucks Gold?	Correction to 120 missing → correction explicitly recorded	$721 \rightarrow 595$ (-126)	$125 \times \rightarrow$ **120** ✓
MS	Total money raised for charity?	Generic fundraising tips dominate → individual donation amounts clearly recorded	$665 \rightarrow 674$ $(+9)$	$1,750 × → **$3,750** ✓
MS	Days for laptop backpack to arrive?	Purchase date missing → purchase and arrival dates both recorded	$672 \rightarrow 673$ $(+1)$	Not answerable × → **5 days** ✓
SS-P	Tips for keeping kitchen clean?	Occupation-only core → cooking habits included in core	$728 \rightarrow 620$ (-108)	Generic tips × → **Personalized** ✓
TR	Most recent transport: bus or train?	Older bus trips dominate → recent Mar. 3 train trip clearly recorded	$601 \rightarrow 604$ $(+3)$	Bus × → **Train** ✓

The two rewards complement each other. QA-only reaches 56.5, whereas CMI+QA reaches 68.8 and improves every category, with the largest gain in Single-session Preference (59.3 to 81.5). These gains are consistent with CMI encouraging the retention of missing or non-redundant dialogue evidence while QA grounds memory decisions in downstream usefulness. Adding RM yields 68.5 accuracy.

Table 3: Ablations on LongMemEval. All metrics are percentages rounded to one decimal. The "w." rows use CMI+QA+RM and vary the CMI estimator.

Variant	Overall	KU	MS	SS-A	SS-P	SS-U	TR
Qwen3-4B	53.3	54.1	40.4	88.0	70.4	85.2	28.9
+RM	55.0	65.6	39.4	90.0	63.0	79.6	33.3
+QA	56.5	62.3	43.6	90.0	59.3	83.3	36.0
+RM+QA	53.3	60.7	40.4	86.0	40.7	81.5	35.1
+CMI	45.0	47.5	27.7	84.0	59.3	66.7	27.2
+CMI+QA	**68.8**	**75.4**	50.0	96.0	81.5	**94.4**	53.5
w. logprob	67.3	**75.4**	**54.3**	98.0	66.7	**94.4**	47.4
w. direct emb.	66.5	63.9	42.6	98.0	**88.9**	85.2	**59.6**
w. residual	68.5	67.2	51.1	**100.0**	85.2	90.7	55.3

Estimator and reward variants. With RM enabled, residual projection reaches 68.5 overall, versus 67.3 for the implemented logprob confidence estimator and 66.5 for direct embedding. By removing memory-predictable embedding directions, it evaluates relevance after accounting for stored information, using embedding and projection in place of two autoregressive generations. The logprob estimator generates a profile before and after the memory update and compares their length-normalized log-probabilities. Residual projection instead uses embedding and projection to evaluate relevance after accounting for stored information. Appendix E reports additional reward ablations with RM enabled. Removing target-band regularization yields 67.8 accuracy; increasing the CMI weight α from 0.3 to 0.5 and 0.7 lowers accuracy to 63.0 and 59.5, respectively. These results support retaining substantial QA supervision when optimizing the intrinsic memory score.

Qualitative consistency. Table 4 links five answer changes to concrete memory differences, with full traces in Appendix J. Across these cases, CMI+QA retains key evidence missed by QA-only with either fewer or nearly unchanged total entries, indicating better evidence selection rather than uniform compression.

Evaluation controls. The paired traces use policies trained on the same data, and all benchmark methods share the downstream answer model and judge. Because substring exact match can reward verbose answers containing both obsolete and corrected values, we use LLM judging for free-form tasks while retaining Recall@5 for recommendation and ROUGE-L for summarization (Appendix K.3). We exclude FC-MH because counterfactual biographies can make stronger answer models override stored evidence with parametric knowledge, confounding memory quality with world knowledge (Appendix K.2).

5 Conclusion

We introduce **UniMem-R1**, an RL framework that unifies extrinsic downstream QA grounding with intrinsic, per-operation CMI rewards. QA evaluates whether the resulting bank supports downstream tasks, while CMI scores the information contributed by each edit relative to the current memory state. Experiments show our UniMem-R1 outperforms the reported RL baselines on three tested long-term memory benchmarks, with both 4B and 8B backbones.

AI USE STATEMENT

We used generative AI tools in the experimental workflow and manuscript preparation. Qwen3.7-Max generated the synthetic QA pairs used for RL training and served as the fixed answer model and correctness judge in the reported experiments, as detailed in Section 4.1 and Appendix C. Qoder and Qwen3.7-Max assisted with code debugging, formatting adjustments, and English-language editing. The authors reviewed all AI-generated or AI-assisted artifacts, verified the reported results, and take full responsibility for the final content of this work.

REFERENCES

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. arXiv preprint arXiv:2407.04363, 2024.

Daniela Calvetti, Serena Morigi, Lothar Reichel, and Fiorella Sgallari. Tikhonov regularization and the l-curve for large discrete ill-posed problems. Journal of computational and applied mathematics, 123(1-2):423–446, 2000.

Camilla Casula, Sebastiano Vecellio Salto, Alan Ramponi, and Sara Tonelli. Delving into qualitative implications of synthetic data for hate speech detection. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1099. URL https://aclanthology.org/2024.emnlp-main.1099/.

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.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Xiang Geng, Zhejian Lai, Jiajun Chen, Hao Yang, and Shujian Huang. Alleviating distribution shift in synthetic data for machine translation quality estimation. In Proceedings of the 63rd Annual

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 7546–7560, 2025.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In Advances in Neural Information Processing Systems, volume 29, 2016.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. In International Conference on Learning Representations, volume 2026, pp. 156259–156291, 2026.

Hongda Jiang, Xinyuan Zhang, Siddhant Garg, Rishab Arora, Shiun-Zu Kuo, Jiayang Xu, Aaron Colak, and Xin Luna Dong. Memory-qa: Answering recall questions based on multimodal memories. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 24255–24277, 2025.

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.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in rl training of llms. In Forty-second International Conference on Machine Learning, 2025.

Chenxin Li, Yifan Liu, Panwang Pan, Hengyu Liu, Xinyu Liu, Wuyang Li, Cheng Wang, Weihao Yu, Yiyang Lin, and Yixuan Yuan. Infobridge: Balanced multimodal integration through conditional dependency modeling. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 393–404, 2025a.

Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, et al. Memos: A memory os for ai system. arXiv preprint arXiv:2507.03724, 2025b.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. IEEE Transactions on Audio, Speech and Language Processing, 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. arXiv preprint arXiv:2601.08435, 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.

Martin Marko, Drahomír Michalko, Adam Kubinec, and Igor Riečanskỳ. Measuring semantic memory using associative and dissociative retrieval tasks. Royal Society Open Science, 11(2), 2024.

Thao Nguyen, Haotian Liu, Yuheng Li, Mu Cai, Utkarsh Ojha, and Yong Jae Lee. Yo'llava: Your personalized language and vision assistant. Advances in Neural Information Processing Systems, 37:40913–40951, 2024.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. Advances in Neural Information Processing Systems, 38:105523–105553, 2026.

Qwen. Qwen3.7: The agent frontier, 2026. URL https://qwen.ai/blog?id=qwen3.7.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. Zep: a temporal knowledge graph architecture for agent memory. arXiv preprint arXiv:2501.13956, 2025.

Patrick Schober, Christa Boer, and Lothar A Schwarte. Correlation coefficients: appropriate use and interpretation. Anesthesia & analgesia, 126(5):1763–1768, 2018.

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

Zhiyu Shen, Ziming Wu, Fuming Lai, Shaobing Lian, and Yanghui Rao. Membuilder: Reinforcing llms for long-term memory construction via attributed dense rewards. arXiv preprint arXiv:2601.05488, 2026.

Alexandra Sobczak, Tineke Steiger, Marthe Mieling, and Nico Bunzeck. Curiosity and surprise differentially affect memory depending on age. Scientific Reports, 15(1):32423, 2025.

Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. Cognitive architectures for language agents. Transactions on Machine Learning Research, 2023.

Guoqing Wang, Sunhao Dai, Guangze Ye, Zeyu Gan, Wei Yao, Yong Deng, Xiaofeng Wu, et al. Information gain-based policy optimization: A simple and effective approach for multi-turn search agents. In International Conference on Learning Representations, volume 2026, pp. 104502–104529, 2026a.

Jingyao Wang, Wenwen Qiang, Zeen Song, Changwen Zheng, and Hui Xiong. Learning to think: Information-theoretic reinforcement fine-tuning for llms. Advances in Neural Information Processing Systems, 38:126208–126245, 2026b.

Yubo Wang, Haoyang Li, Fei Teng, and Lei Chen. Agrag: Advanced graph-based retrieval-augmented generation for llms. In 2026 IEEE 42nd International Conference on Data Engineering (ICDE). IEEE, 2026c.

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

Aaron D Wyner. A definition of conditional mutual information for arbitrary ensembles. Information and Control, 38(1):51–59, 1978.

Jing Xu, Arthur Szlam, and Jason Weston. Beyond goldfish memory: Long-term open-domain conversation. In Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers), pp. 5180–5197, 2022.

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. arXiv preprint arXiv:2508.19828, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Amy Jing-Wen Yin, Anise MS Wu, Yingxin Xiong, Liffy Ka Heng Leong, Caren Man Wai Lei, Jing Zhai, Davis Ka Chio Fong, Zhen Yuan, Ruey-Song Huang, and Robin Chark. The neural vulnerabilities in reward processing in gambling disorder. Journal of Behavioral Addictions, 14 (2):1010–1020, 2025.

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. arXiv preprint arXiv:2601.01885, 2026.

Kai Zhang, Xinyuan Zhang, Ejaz Ahmed, Hongda Jiang, Caleb Kumar, Kai Sun, Zhaojiang Lin, Sanat Sharma, Shereen Oraby, Aaron Colak, et al. Assomem: Scalable memory qa with multi-signal associative retrieval. In International Conference on Learning Representations, volume 2026, pp. 103385–103405, 2026a.

Qingyang Zhang, Haitao Wu, Changqing Zhang, Peilin Zhao, and Yatao Bian. Right question is already half the answer: Fully unsupervised llm reasoning incentivization. Advances in neural information processing systems, 38:67345–67372, 2026b.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. arXiv preprint arXiv:2506.05176, 2025.

Thomas P Zollo, Andrew Wei Tung Siah, Naimeng Ye, Ang Li, and Hongseok Namkoong. Personalllm: Tailoring llms to individual preferences. arXiv preprint arXiv:2409.20296, 2024.

APPENDIX

A NOTATION

Table 5 summarizes the mathematical notation used throughout this paper.

A.1 STRUCTURED MEMORY STATE AND ACTION SPACES

A conversation is represented as a sequence of dialogue sessions $\mathcal{D} = (C_1, C_2, \ldots, C_T)$. Following the structured memory configuration described in Section 4.1, the memory state is

$$M_t = \left(M_t^{\text{core}}, M_t^{\text{epi}}, M_t^{\text{sem}}, M_t^{\text{pro}}\right), \tag{12}$$

where the four components store core, episodic, semantic, and procedural memory, respectively. Let $\mathcal{A}^\tau$ denote the operations available to memory type τ. The complete action space is

$$\mathcal{A} = \bigcup_{\tau \in \mathcal{T}} \mathcal{A}^\tau, \qquad \mathcal{T} = \{\text{core}, \text{epi}, \text{sem}, \text{pro}\}. \tag{13}$$

Here, $\mathcal{T}$ is the set of memory types and $\mathcal{A}$ is the union of their type-specific action spaces. At session t, let $\mathcal{O}_t^\tau$ denote the parsed output for memory type τ. Core contains at most one action–fragment pair, whereas the other outputs contain zero or more pairs. Applying all valid operations gives

$$M_t = \text{Apply}(M_{t-1}, \{\mathcal{O}_t^\tau\}_{\tau \in \mathcal{T}}). \tag{14}$$

The operator Apply executes the valid operations in Core, Episodic, Semantic, and Procedural order to produce M_t. Empty operation lists and explicit SKIP actions leave the corresponding partition unchanged. The complete operation sets are listed in Table 11.

B ALGORITHM DETAILS

This appendix provides pseudocode for the two core procedures of UniMem-R1. Algorithm 1 gives the structured multi-dimensional memory update used at training time, and Algorithm 2 gives the retrieval-augmented QA evaluation used at inference time. The LLM and JUDGELLM steps in Algorithm 2 use the answer-generation and binary-judging prompts listed in Appendix I.4.

Table 5: Mathematical notation. All symbols are indexed by time t unless otherwise specified.

Notation	Meaning
$\mathcal{D}$	Dialogue sequence
C_t	Session at time t
T	Number of sessions
M_t	Memory state at t
M_t^{core}	Core memory
M_t^{epi}	Episodic memory
M_t^{sem}	Semantic memory
M_t^{pro}	Procedural memory
M_{t-1}	Previous memory state
M_{t-1}^{local}	Local retrieved subset
n_t^{local}	Number of entries in the local subset
π_θ	Policy network
π^{core}, π^{epi}, π^{sem}, π^{pro}	Type-specific agents
$\mathcal{A}$	Action space
$\mathcal{A}^{\text{core}}$	Core actions
$\mathcal{A}^{\text{epi}}$	Episodic actions
$\mathcal{A}^{\text{sem}}$	Semantic actions
$\mathcal{A}^{\text{pro}}$	Procedural actions
a_t	Action at t
m_t^{new}	Candidate memory fragment
$I(X;Y \mid Z)$	Conditional mutual information
$p_{\text{ref}}(\cdot)$	Reference LM
r_t^{CMI}	CMI reward
$\widehat{r}_t^{\text{CMI}}$	Estimated CMI reward
$\widetilde{r}_t^{\text{CMI}}$	Regularized CMI reward
r_t^{QA}	QA reward
r_t	Composite reward
α	Reward mixing weight
$\mathbf{e}_C, \mathbf{e}_{\text{new}}$	Session and candidate embeddings
$\mathbf{M}$	Local-memory embedding matrix
$\mathbf{r}_C, \mathbf{r}_{\text{new}}$	Residual embeddings
N	Rollouts per prompt
β	KL penalty coefficient
μ, σ	Gaussian center and width

Algorithm 1 Structured Multi-dimensional Memory Update at turn t.

Require: Session C_t, memory state M_{t-1}, timestamp τ_t, policy π_θ
Ensure: Updated memory M_t, per-turn reward r_t

$s_t \leftarrow \textsc{FormatSession}(C_t, \tau_t)$
for $\tau \in \mathcal{T} = \{\text{core}, \text{epi}, \text{sem}, \text{pro}\}$ **do**
  $R_t^\tau \leftarrow \textsc{TopK}(M_{t-1}^\tau, s_t)$
end for
$\{\mathcal{O}_t^\tau\}_{\tau \in \mathcal{T}} \leftarrow \textsc{ParallelDecodeAndParse}(\{R_t^\tau\}_{\tau \in \mathcal{T}}, s_t, \pi_\theta)$
$M_t \leftarrow M_{t-1}$
for $\tau \in (\text{core}, \text{epi}, \text{sem}, \text{pro})$ **do**
  for $o \in \mathcal{O}_t^\tau$ **do**
    $M_t^\tau \leftarrow \textsc{Apply}(o, M_t^\tau)$
  end for
end for
$r_t \leftarrow \textsc{Reward}(M_{t-1}, M_t, C_t)$ ▷ action CMI plus session QA; Eq. 10
return $M_t,\ r_t$

Algorithm 2 Retrieval-Augmented QA Evaluation.

Require: Question Q, memory state M, user id u, top-k
Ensure: Answer A, correctness score s

$e_Q \leftarrow \text{EMBED}(Q)$
$R \leftarrow \text{FAISSSEARCH}(e_Q, M, k)$ **filter by** u
$c \leftarrow M$.core
$\mathcal{X} \leftarrow \emptyset$
for $m_i \in R$ **do**
    $\mathcal{X} \leftarrow \mathcal{X} \cup \text{LOOKUPCHUNK}(m_i.\text{session_idx})$
end for
$ctx \leftarrow [\, c;\ R;\ \mathcal{X};\ \text{TIMEINFO}(R)\,]$
$A \leftarrow \text{LLM}(ctx, Q)$
$s \leftarrow \text{JUDGELLM}(Q, A, A^\star)$
return $A,\ s$

C EXPERIMENTAL CONFIGURATIONS

This appendix lists the full experimental configuration referenced in Section 4.1, including dataset partitions and RAG segmentation (Table 6), memory and retrieval settings (Table 7), and the GRPO and CMI-reward hyperparameters used during policy optimization (Table 8). All values reported here are the defaults used to produce the results in Section 4; ablation studies vary only the parameters explicitly named in their respective tables.

Table 6: Dataset partitions and RAG-baseline segmentation. We follow the official LongMemEval split released by Wu et al. (2025); the SFT partition is left untouched as our framework dispenses with supervised warm-start. LoCoMo serves as a strictly out-of-distribution benchmark and is never exposed during training.

Item	Value
LongMemEval (longmemeval_s)	
RL split (training) – dialogues	50
RL split (training) – sessions	$\sim$2,400
RL split – synthetic QA pairs	12,000
QA pairs per session	5
QA generation model	Qwen3.7-Max
SFT split	untouched
Test split – questions	400
Question categories	6
LoCoMo (OOD evaluation)	
Conversations	full released set
Max sessions per conversation	35
RAG baseline segmentation	
Chunk granularity	utterance-level
Chunk size (`RAG_CHUNK_TURN_SIZE`)	1 turn
Index backend	FAISS (cosine)
Retrieved chunks per query (K)	$\{5, 10\}$

Table 7: Memory and retrieval configuration for the matched Qwen3-4B RL methods (MemBuilder-RL and UniMem-R1).

Component	**Setting**
Policy backbone	Qwen3-4B-Instruct-2507
Sentence encoder	Qwen3-Embedding-0.6B
Encoder serving	self-hosted vLLM
Answer model	Qwen3.7-Max
LLM judge	Qwen3.7-Max
Top-k (build)	20
Top-k (QA)	10
Core cap L_{core}	5,000
Compression trigger	$>$90% usage
Memory types	core/epi/sem/pro

Table 8: GRPO hyperparameters shared by **UniMem-R1** and **MemBuilder-RL**, together with the CMI-reward settings used only by **UniMem-R1**. Auxiliary penalties follow Shen et al. (2026) defaults except where noted.

Hyperparameter	**Value**
GRPO Shao et al. (2024) optimization	
Hardware	8$\times$ H20-96GB
Epochs	5
Learning rate	1×10^{-6}
Train batch size	32
Rollouts/session (N)	8
KL coefficient (β)	0.02
PPO clip (ϵ_l, ϵ_h)	0.2, 0.2
Rollout temperature	0.8
Rollout top-p	0.9
Max prompt len.	20,000 tok
Max response len.	6,000 tok
Action-conditioned CMI reward	
Tikhonov reg. (λ)	1×10^{-4}
Top-similar in cond. set	8
Recent episodic additions	3
Query components	session/new/target
Multi-item aggregation	mean
Reward regularization and composition	
Target-band map	Gaussian
Target center (μ)	0.35
Bandwidth (σ)	0.15
Peak	1.0
CMI/QA mix (α)	0.3
Empty-action penalty	-0.5
Auxiliary action penalties	
Intra-sess. dup-add	0.3
Cross-sess. dup-add	0.5
Core-specificity	0.3
Sem-SKIP (just.)	$+0.1$
Sem-SKIP (unjust.)	-0.1

C.1 CURRICULUM LEARNING

To expose the information-extraction policy to increasingly demanding contexts without a task-specific SFT warm start, we use a stratified curriculum. We initialize from an instruction-tuned

model and define the difficulty $d(C_t)$ of session C_t as

$$d(C_t) = w_s t + w_c|\mathcal{C}_t| + w_q|\mathcal{Q}_t|, \tag{15}$$

where the session index t captures accumulated history depth, $\mathcal{C}_t$ and $\mathcal{Q}_t$ are the candidate-fragment and QA-pair sets, $|\cdot|$ denotes set size, and w_s, w_c, and w_q are the respective non-negative weights for depth, fragment count, and question count. Sessions are divided into three percentile-based tiers and sorted by difficulty. Batches mix all tiers to avoid abrupt transitions, while samples progress from easy to hard within each tier. Within each difficulty tier, this schedule trains the policy on increasingly demanding memory decisions while retaining both dialogue-information supervision and downstream QA grounding.

C.2 Training Metrics

Figures 4 and 5 complement the main training curves with efficiency and partition-level diagnostics. In Figure 4, CMI+QA maintains a near-perfect format pass rate while its length penalty declines and its response length stays compact. QA-only generations, by contrast, grow sharply in the latter half of training. Thus, the shorter CMI+QA outputs do not arise from malformed responses.

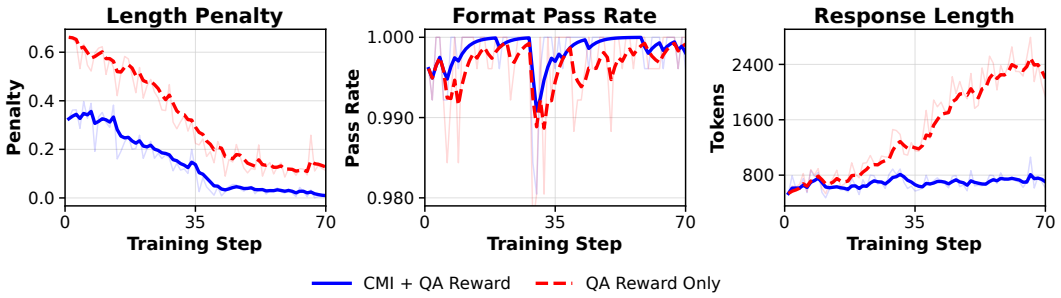


Figure 4: Training efficiency and auxiliary metrics over 70 steps (left to right): length penalty, format pass rate, and average response length. CMI+QA maintains valid outputs while learning shorter memory operations; the CMI-reward and policy-entropy curves are included in Figure 3.

Figure 5 decomposes rewards across the core, episodic, semantic, and procedural agents. All four trajectories remain bounded and generally rise, with no partition collapsing or dominating. Table 9 in Appendix C.3 reports the corresponding evaluation-time footprint.

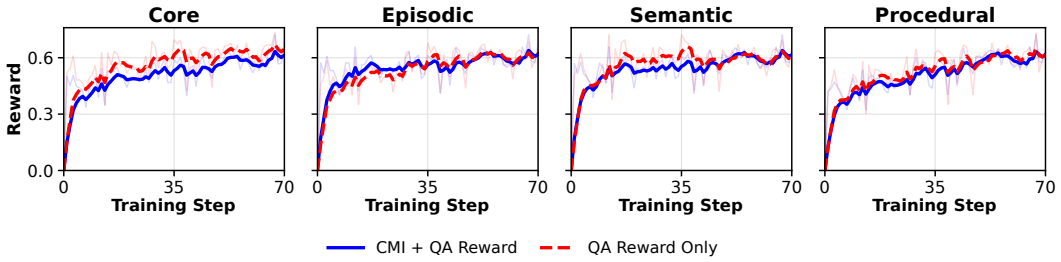


Figure 5: Per-agent reward progression over training (left to right): core, episodic, semantic, and procedural memory. All four trajectories remain bounded and generally increase.

C.3 Training Efficiency Details

Compared with QA-only, the full CMI+QA configuration reduces total entries by 5.2% (635.6 vs. 670.5) and characters by 7.5% (420.2K vs. 454.1K). The largest reduction is procedural (82.6 vs. 139.9 entries; 84.4K vs. 138.2K characters), while episodic memory grows moderately, indicating redistribution rather than uniform shrinkage. The reported configuration retains length and duplication regularizers alongside the CMI+QA reward. Downstream accuracy is reported in Section 4.2.

Table 9: Mean memory-bank size per evaluated conversation. Count is the number of entries; Chars(K) is the number of characters in thousands.

	QA Only		CMI+QA	
Type	**Count**	**Chars(K)**	**Count**	**Chars(K)**
Core	3.9	**3.4**	**2.7**	3.5
Episodic	**284.1**	**207.2**	318.8	227.9
Semantic	242.6	105.3	**231.5**	**104.4**
Procedural	139.9	138.2	**82.6**	**84.4**
Total	670.5	454.1	635.6	420.2

D Reward Regularization Terms

Beyond the core CMI and QA reward (Eq. 10), we apply several implementation-level regularization terms.

Empirical Target-Band Regularizer. The Gaussian target-band mapping and all of its notation are defined with the hybrid reward in Eq. (9). The reported configuration uses target center μ=0.35 and bandwidth σ=0.15, selected through reward-distribution monitoring during training.

Format Validity Gate. At the session level, if any of the four type-specific agents produces a malformed output (failing structured format validation), the entire session reward is masked to zero. Additionally, at each turn, a format penalty $\rho_f = -0.5$ is added to the per-turn CMI reward when the output is malformed, providing an immediate corrective signal before the session-level gate takes effect.

Length Penalty. To discourage excessively verbose memory entries, a multiplicative length penalty is applied to the rollout reward:

$$r_{\text{penalized}} = r \cdot (1 - \lambda_\ell \cdot \ell(M)), \tag{16}$$

where r and $r_{\text{penalized}}$ are the rollout rewards before and after length regularization, $\lambda_\ell \in [0, 1]$ is the penalty weight, and $\ell(M)$ is the normalized length cost of generated memory M relative to the configured thresholds.

Duplication Penalties. Two deduplication mechanisms discourage the policy from exploiting repetitive memory patterns: *(i)* an intra-session penalty that reduces reward proportionally when the same content is added multiple times within a single session (weight $w_{\text{intra}} = 0.3$); *(ii)* a cross-session penalty that detects and penalizes verbatim or near-verbatim memories appearing across different sessions (weight $w_{\text{cross}} = 0.5$).

Core Specificity Penalty. A pattern-based filter detects generic, template-like core memories (e.g., "lifelong learner") and applies a multiplicative penalty (weight 0.3). The penalty favors specific core identity fragments.

Attribution Weighting. To concentrate gradient signal on the memory types most relevant to downstream QA, we track which memory slot is most frequently retrieved during answer generation across the batch. The dominant agent type $\tau^* = \arg\max_\tau \text{retrieve_count}(\tau)$ receives an amplified reward weight α_{attr} (set to 4.0 in our experiments), while non-dominant types retain unit weight. Formally, the attribution-adjusted reward for agent τ is:

$$r_\tau^{\text{attr}} = \begin{cases} \alpha_{\text{attr}} \cdot r_\tau & \text{if } \tau = \tau^* \\ r_\tau & \text{otherwise} \end{cases} \tag{17}$$

Here, r_τ is the unweighted reward for memory type τ, and r_τ^{attr} is its attribution-adjusted value. This mechanism encourages the policy to prioritize memory types that empirically contribute most to QA accuracy, without requiring per-type reward engineering.

E Additional Reward Ablations

Table 10 complements the estimator comparison in Table 3 with target-band-regularization and reward-weight sensitivity results.

Table 10: Reward-regularization and weight ablations on LongMemEval with RM enabled. All metrics are percentages rounded to one decimal. UniMem-R1+RM uses residual projection, Gaussian target-band regularization, and α=0.3.

Variant	Overall	KU	MS	SS-A	SS-P	SS-U	TR
w/o target-band reg.	67.8	**72.1**	39.4	96.0	**88.9**	**90.7**	**60.5**
α=0.5	63.0	68.9	50.0	98.0	**88.9**	87.0	37.7
α=0.7	59.5	63.9	43.6	96.0	74.1	83.3	39.5
UniMem-R1+RM	**68.5**	67.2	**51.1**	**100.0**	85.2	**90.7**	55.3

F Alternative Information-Gain Estimators

In addition to the residual-projection estimator used in the main method, we compare the following variants in Table 3.

Log-Probability Reference Estimator. A distribution-level reference uses a frozen LM q and a generated user-profile probe Y. Let $q_t(Y) = q(Y \mid M_t, C_t)$ and $q_{t-1}(Y) = q(Y \mid M_{t-1}, C_t)$. The ideal per-token information gain is

$$\mathcal{I}_q(a_t) = \frac{1}{L} D_{\mathrm{KL}}(q_t \| q_{t-1}) = \frac{1}{L} \mathbb{E}_{Y \sim q_t} \left[\log \frac{q_t(Y)}{q_{t-1}(Y)} \right]. \tag{18}$$

Here, a_t is the memory action that transforms M_{t-1} into M_t, L is the number of tokens in probe Y, D_{KL} is KL divergence, and the expectation is taken over $Y \sim q_t$. A one-sample estimator generates $Y \sim q_t$ and scores the same sequence under both states. It requires no QA labels or judge, but costs one generation and two scoring passes per action. Our ablation instead compares the length-normalized log-probabilities of profiles independently generated under the two states; Section 4.3 distinguishes this confidence estimator from Eq. (18).

Direct-Embedding Estimator. We also test a direct estimator that explicitly multiplies session relevance by non-redundancy:

$$\widehat{r}^{\mathrm{CMI}}_{t,\mathrm{emb}} = \cos_+(\mathbf{e}_C, \mathbf{e}_{\mathrm{new}}) \left[1 - \max_{m \in M^{\mathrm{local}}_{t-1}} \cos_+(\mathbf{e}_{\mathrm{new}}, \mathbf{e}_m) \right], \tag{19}$$

where $\mathbf{e}_C$ and $\mathbf{e}_{\mathrm{new}}$ embed the session and candidate, $\mathbf{e}_m$ embeds local-memory entry m, and $\cos_+(x, y) = \max(0, \cos(x, y))$ is nonnegative cosine similarity. The first factor measures session relevance and the bracketed factor measures non-redundancy with respect to M^{local}_{t-1}. Table 3 compares all three estimators.

G Local CMI Conditioning Details

For each candidate, M^{local}_{t-1} contains the eight most similar entries from the same memory type. The retrieval query concatenates the session, new fragment, and any target memory. For episodic actions, the three most recent entries are added to preserve temporal context. Duplicate indices are removed. If the bank is empty, residual projection is skipped and the estimator uses non-negative session–candidate cosine similarity. Multiple candidate fragments are scored separately and averaged.

H Memory Action Space

In the batch mode used for the reported experiments, the shared policy is decoded once per memory type at every session. Core returns a single-operation object, while the other memory types return

Table 11: Batch-mode action schema used in the reported experiments. Core emits at most one operation; the other memory types emit operation lists and may therefore apply multiple edits per session.

Memory Type	Actions	Cardinality	State-transition semantics
Core	APPEND, REPLACE, REWRITE	0–1	Append a fragment, replace an exact span, or rewrite the complete bounded profile.
Episodic	ADD, UPDATE, MERGE	0–many	Add a time-stamped event, append a follow-up linked to an earlier event, or append a timeline summary while retaining source events.
Semantic	ADD, UPDATE, SKIP	0–many	Add a topic-specific fact, emit a revised entry for an existing topic, or explicitly skip redundant/common knowledge.
Procedural	ADD, UPDATE	0–many	Add or revise a procedure; an empty list represents no procedural edit.

operation lists. Table 11 summarizes these action spaces, and the following sections define each operation.

The optional agentic implementation uses one compact DSL response rather than four independently decoded JSON outputs. It requires exactly one Core line and permits multiple lines for the other memory types; SKIP maps to an empty operation list. Its released prompt exposes SKIP/APPEND/REPLACE for Core, SKIP/ADD/UPDATE for Episodic and Semantic memory, and SKIP/ADD for Procedural memory. The shared parser and executor additionally recognize Core REWRITE, Episodic MERGE, and Procedural UPDATE for compatibility, but these additional actions are not requested by the compact prompt. All results in the main paper use the batch schema in Table 11.

H.1 CORE MEMORY ACTIONS

Core Memory maintains a persistent, bounded-length user profile that is always included in the prompt context (not stored in the vector database). It captures stable identity attributes such as name, occupation, personality traits, preferences, key relationships, and long-term goals. The character limit is set to $L_{\text{core}} = 5000$ characters; when usage exceeds 90%, the prompt recommends REWRITE. A separate safety pass compresses the block only if its post-operation length exceeds the hard character limit.

APPEND. Appends new factual content to the end of the existing core memory block. This is the default operation when the block has not yet reached capacity ($< 90\%$ full). The model outputs a content string that is concatenated with a newline separator. *Example scenario:* A user mentions their occupation for the first time. The model appends "Works as a software engineer at Google, specializing in machine learning" to the profile.

REPLACE. Performs a targeted substitution of a specific text span within the core memory. The model outputs an $(\text{old_text}, \text{new_text})$ pair; the first exact occurrence of old_text is replaced by new_text. This enables fine-grained factual corrections without rewriting the entire block. *Example scenario:* The user mentions acquiring additional items; the model replaces "owns 17 postcards" with "owns 25 postcards."

REWRITE. Replaces the complete Core block with a consolidated profile. The batch prompt recommends this operation when usage exceeds 90% or a major reorganization is needed. Independently, the implementation invokes a safety compression pass if the post-operation block exceeds the hard $L_{\text{core}} = 5000$-character limit.

H.2 EPISODIC MEMORY ACTIONS

Episodic Memory stores time-ordered event records in the vector database, each containing an absolute timestamp, a concise summary, and detailed descriptive content. This dimension serves as the user's "diary," capturing who, what, when, where, and why for each interaction event.

ADD. Inserts a completely new event record into the episodic store. Each entry follows the format: `YYYY-MM-DD: summary | Details: ...` and is indexed in the vector database for similarity-based retrieval. The model should create one ADD operation per distinct event. *Example scenario:* The user mentions attending a cooking class on March 15th. The model creates a new episodic entry with the event date, a summary of the class, and details including the instructor, dishes learned, and classmates met.

UPDATE. Creates a new event entry that explicitly references and extends a previous episodic record. The original record is preserved (append-only semantics); the new entry carries its own timestamp and adds follow-up information. This maintains a linked narrative across sessions. *Example scenario:* The user previously mentioned starting a gym routine. In a later session, they report progress. The model creates a new entry referencing the original, noting updated achievements.

MERGE. Consolidates multiple related episodic records into a single timeline summary spanning a date range. The merged entry synthesizes common patterns and draws well-supported conclusions across events, while the original entries remain in the vector store for granular retrieval. *Example scenario:* Three consecutive weekly cooking class entries are merged into a timeline summary ("2024-03-15 to 2024-03-29: Alex's Italian cooking journey") capturing progression, recurring participants, and evidence-based conclusions about learning patterns.

H.3 SEMANTIC MEMORY ACTIONS

Semantic Memory stores general knowledge, concepts, entity descriptions, and factual information in the vector database. Entries follow a fine-grained, topic-specific structure: `Name - Aspect: summary | Details: ...`. This dimension captures what the system knows about people, objects, places, and concepts in the user's world.

ADD. Inserts a new knowledge entry for a previously unknown concept, person, object, or place. Each entry is scoped to a single topic or aspect (e.g., "Sarah – Hobbies" vs. "Sarah – Career") to maintain granularity and retrieval precision. *Example scenario:* The user mentions their friend Sarah for the first time, describing her profession. The model creates "Sarah – Career: Senior backend engineer at fintech startup | Details: ...".

UPDATE. Augments an existing semantic entry with newly revealed information about the same topic. The model outputs the original entry and its expanded version, enabling the vector store to index the richer representation. *Example scenario:* In a subsequent session, the user mentions Sarah recently got promoted. The model updates the career entry to include the promotion details.

SKIP. Explicitly declines to perform any operation. This action is selected when the session content contains only common knowledge (e.g., well-known software, famous landmarks) or information already fully captured in existing memory entries. *Example scenario:* The user asks about Python programming. Since Python is common knowledge and not user-specific, the model outputs SKIP with the reason "common knowledge."

H.4 PROCEDURAL MEMORY ACTIONS

Procedural Memory captures step-by-step processes, workflows, routines, and how-to instructions mentioned by the user. Entries follow the format: `Description | Steps: 1.... 2.... | Context: ....` This dimension is the least frequently activated, as most conversations do not contain explicit procedural content.

ADD. Records a new procedure or workflow extracted from the session. Each entry includes numbered steps with specific details (times, quantities, tools) and optional context about when/where the procedure applies. *Example scenario:* The user describes their morning routine for managing a health condition. The model extracts and stores the numbered steps with specific medications, timings, and dosages.

UPDATE. Revises an existing procedural entry with corrections, additional steps, or refined details. The model outputs both the original and updated versions, preserving the change history. *Example scenario:* The user mentions they now add an extra step to their sourdough bread recipe (cold-proofing overnight). The model updates the existing recipe procedure to include this new step.

H.5 ACTION SPACE DESIGN RATIONALE

The heterogeneous action spaces across memory types reflect their distinct storage semantics and update patterns:

- **Core Memory** operates on a single bounded-length block (not vectorized), hence requiring text-editing primitives (APPEND, REPLACE, and REWRITE), rather than collection-level operations.
- **Episodic Memory** prioritizes temporal completeness and narrative continuity, supporting MERGE to consolidate recurring patterns while preserving individual event granularity.
- **Semantic Memory** includes an explicit SKIP action because not all sessions contain novel user-specific knowledge; this prevents pollution of the knowledge base with common facts.
- **Procedural Memory** uses ADD and UPDATE for procedures; an empty operation list leaves this partition unchanged.

During RL training, the CMI module scores each emitted edit before the scores are averaged for the rollout. This gives the policy action-sensitive feedback without a separate hand-written reward for every operation.

I PROMPT TEMPLATES

This section documents the verbatim prompt templates used in UniMem-R1, organized by their role in the system: (i) memory management agents that update each of the four memory slots (Sec. I.1); (ii) the agentic RL rollout prompts that drive policy training (Sec. I.2); (iii) auxiliary prompts used by the CMI reward estimator (Sec. I.3); and (iv) evaluation prompts for answer generation and LLM-based judging (Sec. I.4). Placeholders enclosed in double braces `{{...}}` (Jinja2-style) or single braces `{...}` (Python f-string style) are filled at runtime.

I.1 MEMORY MANAGEMENT PROMPTS

These prompts are issued by the four type-specific memory agents that share policy π_θ and operate on their respective memory dimensions. Each prompt enforces a strict JSON output schema and includes detailed in-context examples to anchor action selection.

Core Memory Agent. The Core agent maintains the persistent user profile within a fixed character budget $L_{\text{core}} = 5000$. The four-agent data-construction prompt uses APPEND, REPLACE, and REWRITE. The compact RL prompt below uses SKIP, APPEND, and REPLACE.

Listing 1: Core Memory Agent prompt

```
You are the Core Memory Manager. Your role is to analyze user messages and
extract fundamental information about the user that will be beneficial in
future conversations.

Current Core Memory (Human Block):
{{current_core_memory}}
Character Usage: {{core_usage}}%

New Messages:
{{messages}}
```

```
What to Extract and Save:
- User's name, identity, role, occupation, location
- Personality traits and characteristics
- Preferences and values (what they like/dislike, care about)
- Personal profile facts and background
- Key relationships (family, close friends, colleagues)
- Long-term projects, goals, and aspirations
- User behaviors and habits
- Critical life events and milestones

Instructions:
1. Examine all messages thoroughly to extract EVERY detail about the
   user's preferences, personal information, and vital facts.
2. Decide on ONE operation:
   - APPEND : Add new information to existing block (if <90% full)
   - REPLACE: Update specific outdated or incorrect information
   - REWRITE: Reorganize and consolidate the entire block
              (if >90% full or major updates needed)

Return JSON with ONE of these operations:
  {"operation": "APPEND",  "content": "..."}
  {"operation": "REPLACE", "old_text": "...", "new_text": "..."}
  {"operation": "REWRITE", "content": "... (under 5000 chars)"}

Return ONLY the JSON object. No explanations or extra text.
```

Episodic Memory Agent. The Episodic agent extracts time-stamped events from each session and emits one or more ADD/UPDATE/MERGE operations (Appendix H.2). Timestamps are normalized to absolute formats by combining the conversation timestamp with relative expressions found in the dialogue.

Listing 2: Episodic Memory Agent prompt (abridged)

```
You are the Episodic Memory Manager. Manage time-ordered event memories.

Each episodic memory MUST include:
  (a) summary    : Short textual summary of the event
  (b) timestamp  : YYYY-MM-DD[ HH:MM] (absolute time only)
  (c) details    : Detailed description (who, what, when, where, why)
  (d) event_type : conversation | activity | observation | plan

IMPORTANT TIMESTAMP RULES:
Conversation Timestamp: {{conversation_timestamp}}
  - Use ONLY absolute dates (no "yesterday", "last week" in timestamps).
  - Resolve relative expressions using the conversation timestamp.
  - Preserve the original time expression in the Details field.

Existing Recent Episodic Memories:
{{existing_episodic}}

New Messages:
{{messages}}

For each new event, choose one action:
  - ADD    : Completely new event
  - UPDATE : New related event referencing a prior record
             (old version preserved in history)
  - MERGE  : Combine related events into a timeline with date range,
             drawing well-supported conclusions

Return JSON:
{"operations": [
  {"action": "ADD",    "memory": "YYYY-MM-DD: summary | Details: ..."},
  {"action": "UPDATE", "old_memory": "...", "new_memory": "..."},
  {"action": "MERGE",  "old_memories": [...], "new_memory": "..."}
]}

Output at most 50 operations per response. Return ONLY the JSON object.
```

Semantic Memory Agent. The Semantic agent stores fine-grained, topic-specific knowledge about entities (people, places, objects, concepts). It explicitly emits SKIP for common knowledge, preventing pollution of the user-specific knowledge base.

Listing 3: Semantic Memory Agent prompt (abridged)

```
You are the Semantic Memory Manager. Manage conceptual knowledge about
people, places, objects, and concepts.

ONLY save NEW concepts that are NEW to you. DO NOT save common knowledge
such as well-known software, famous people, or common places.

GRANULARITY PRINCIPLE: Store information in FINE-GRAINED, TOPIC-SPECIFIC
entries. For people, split by aspect when appropriate:
  {Name} - Career/Work | Hobbies/Interests | Family | Pets |
  Personality/Values | Possessions

Each semantic memory entry MUST include:
  (a) name     : The concept/person/object name
  (b) summary  : A concise explanation
  (c) details  : Extended description (physical attributes, relationships,
                 background, distinguishing features)
  (d) category : person | object | place | concept | relationship

Existing Semantic Memories (sample):
{{existing_semantic}}

New Messages:
{{messages}}

For each concept, decide on operation:
  - ADD    : Completely new concept/person/object
  - UPDATE : Add new information to an existing concept
  - SKIP   : Common knowledge or already fully captured

Return JSON:
{"operations": [
  {"action": "ADD",    "memory": "Name - Aspect: summary | Details: ..."},
  {"action": "UPDATE", "old_memory": "...", "new_memory": "..."},
  {"action": "SKIP",   "reason": "..."}
]}

Return ONLY the JSON object.
```

Procedural Memory Agent. The Procedural agent extracts step-by-step processes, workflows, and routines. It returns an empty operation list when no procedure is present; the compact RL prompt represents the same choice with SKIP.

Listing 4: Procedural Memory Agent prompt (abridged)

```
You are the Procedural Memory Manager. Manage step-by-step processes,
workflows, and instructions.

Each procedural memory entry MUST include:
  (a) entry_type  : workflow | guide | recipe | troubleshooting | routine
  (b) description : Short descriptive text
  (c) steps       : Numbered steps with specific details
                    (times, temperatures, quantities, tools)
  (d) context     : When/where/why this procedure is used

Existing Procedural Memories:
{{existing_procedural}}

New Messages:
{{messages}}

For each procedure, choose:
  - ADD    : New procedure or workflow
  - UPDATE : Revise an existing procedure with corrections/additions

Most conversations contain no procedural content -- in that case return an
empty operations array.

Return JSON:
{"operations": [
  {"action": "ADD",
   "memory": "Description | Steps: 1. ... 2. ... | Context: ..."},
  {"action": "UPDATE", "old_memory": "...", "new_memory": "..."}
]}
```

Core Memory Compression. The Core prompt recommends REWRITE once usage exceeds 90%. If the resulting block nevertheless exceeds the hard L_{core} limit after APPEND, REPLACE, or REWRITE, this auxiliary prompt compresses it to under 3,000 characters while preserving identity-defining facts.

Listing 5: Core Memory compression prompt

```
The Core Memory is too long ({length} chars, limit: {limit}).

Compress it to under 3000 characters, keeping only core identity and
critical facts:
  - User's name, role, occupation, key relationships
  - Personality traits and important preferences
  - Long-term goals and critical life events
  - Unique characteristics that define the user

Remove or compress:
  - Redundant descriptions and verbose explanations
  - Minor details and conversational context
  - Detailed examples (keep only key takeaways)

Current content:
{content}

Output format: {"content": "compressed version under 3000 chars"}
Respond with ONLY the JSON object.
```

I.2 AGENTIC RL ROLLOUT PROMPTS

In compact rollout mode, the policy emits all four memory operations within a single response. This removes four separate model calls per session. The system prompt fixes the output grammar; the user prompt contains retrieved memory and the current session.

System Prompt. Defines the DSL grammar and per-type action constraints for one rollout step. Each line follows `TYPE:ACTION|field1|field2`.

Listing 6: Agentic RL system prompt

```
You are a Memory Manager. After each conversation session, output memory
operations in compact DSL format (one operation per line).

Memory types: CORE, EPISODIC, SEMANTIC, PROCEDURAL.

Format -- each line is TYPE:ACTION|field1|field2:
  CORE:SKIP
  CORE:APPEND|new info to add
  CORE:REPLACE|old text|new text
  EPISODIC:SKIP
  EPISODIC:ADD|YYYY-MM-DD: event summary
  EPISODIC:UPDATE|old memory text|new memory text
  SEMANTIC:SKIP
  SEMANTIC:ADD|Topic - concise fact
  SEMANTIC:UPDATE|old memory text|new memory text
  PROCEDURAL:SKIP
  PROCEDURAL:ADD|How to X: 1. step 2. step

Rules:
- Output ONLY DSL lines. No explanations, no markdown, no JSON.
- Every response MUST include exactly one line for CORE
  (SKIP/APPEND/REPLACE).
- For EPISODIC, SEMANTIC, PROCEDURAL: output one or more ADD/UPDATE
  lines, or a single SKIP line if nothing relevant.
- Be concise: 1 sentence per memory entry. No redundancy.
- Use | as field separator. Do not use | inside field values.
```

Per-turn User Prompt. At each rollout step, the retrieved top-k memories from each slot are injected together with the current session text and timestamp.

Listing 7: Agentic per-turn user prompt template

```
[Memory] Core: {core_memory}
[Memory] Episodic: {episodic_memories}
[Memory] Semantic: {semantic_memories}
```

```
[Memory] Procedural: {procedural_memories}
[Session #{session_index}, {session_date}]
{session_text}
Output memory operations:
```

I.3 CMI Reward Estimation Prompts

The CMI reward module includes two auxiliary prompts: a user-profiling prompt for the high-cost logprob reference estimator, and a merge-summary prompt invoked when the policy emits a MERGE action with multiple source memories. The main residual-projection estimator uses neither prompt. These prompts do not form a memory-quality gate or an additional RL reward.

User Profiling Prompt (Logprob CMI). Used by the `LogprobCMIEstimator` reference. The model is asked to generate a user profile conditioned on either M_{t-1} or M_t; the average token log-probability of the generated profile under each condition is differenced to obtain the CMI estimate.

Listing 8: User profiling prompt for logprob-based CMI

```
You are a user profiling assistant. Based on the memory state and
conversation below, write a concise but comprehensive profile of this
user. Include their preferences, personality traits, key facts, and any
important context about their life.

Memory state:
{memory_text}

Recent conversation:
{context}

[user turn]
Write a brief user profile based on the above information.
```

Memory Merge Summary Prompt. When the policy emits a MERGE action with multiple source fragments, this prompt invokes an external summarizer to produce a single merged memory text that preserves all key facts and timestamps without introducing new information.

Listing 9: Memory merge summary prompt

```
You are a memory management assistant. Merge the following {agent_type}
memories into a single, concise summary that preserves all key facts,
timestamps, and details. Do NOT add any information not present in the
original memories. Output only the merged memory text, nothing else.

Memories to merge:
{memories_text}
```

I.4 Evaluation Prompts

For offline evaluation on LongMemEval, LOCOMO, and MemoryAgentBench, we use a retrieval-augmented answer-generation prompt followed by an LLM judge that scores correctness against the gold answer.

Answer Generation Prompt. The answer model is given the user's Core Memory, top-k retrieved memories, optional supporting conversation chunks, and a current-time context for resolving relative time expressions.

Listing 10: Retrieval-augmented answer generation prompt (abridged)

```
{additional_context}

Core Memory (User Profile):
{core_memory}

Retrieved Memories:
1. {memory_1}
2. {memory_2}
...
```

```
[Optional] Supporting Original Conversations:
{chunk_context}

The current date/time is {current_time}. Use this as the reference point
when answering questions about relative time.

Question: {question}

Instructions:
1. Carefully analyze the retrieved memories to find relevant information.
2. Consider synonyms and related concepts.
3. If memories mention specific dates/times, use those for time questions.
4. If memories contain contradictory information, prefer the most recent.
5. Focus on memory content, not exact word matches.

For factual questions (What/When/Where/Who):
- Answer based on direct information in the memories.
- If the specific fact is not mentioned, respond: "Not answerable".

For inference/reasoning questions (Would/Could/Likely):
- Make reasonable inferences based on related information.

When to say "Not answerable":
- The question asks about a DIFFERENT person than the memories describe.
- The event/action is NOT mentioned in ANY memory.
- The retrieved information is about a similar but DIFFERENT event.

Provide a concise, direct answer or state "Not answerable".
```

LLM Judge Prompt. Following the LongMemEval Wu et al. (2025) and LOCOMO Maharana et al. (2024) evaluation protocols, we use an LLM judge to score binary correctness. The judge handles temporal equivalence, "Not answerable" edge cases, and entity-mismatch errors explicitly.

Listing 11: LLM judge prompt for binary correctness scoring

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

The gold answer will usually be a concise short answer that includes the
referenced topic. The generated answer might be much longer; be generous
-- as long as it touches on the same topic as the gold answer, count it
as CORRECT.

For time-related questions: as long as the generated answer refers to
the same date/period as the gold answer (even if format differs, e.g.,
"May 7th" vs "7 May"), count it as CORRECT.

Handling "Not answerable" cases:
  1. GOLD = "Not answerable":
     The generated answer is CORRECT if it indicates unavailability via
     any equivalent phrasing ("no information", "cannot be determined").
  2. GOLD is a SPECIFIC answer:
     A generated "Not answerable" is WRONG.
     If the answer attributes the correct fact to the WRONG person/entity,
     it is also WRONG.
  3. CRITICAL: "Not answerable" can ONLY be CORRECT when GOLD is also
     "Not answerable". Do NOT be misled by reasoning -- focus on whether
     the answer actually provides the requested information.

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

First provide a one-sentence explanation, then return the label as JSON:
  {"label": "CORRECT" | "WRONG"}
```

J Case Studies: CMI Reward Impact on Memory Construction

To qualitatively contrast the memory-construction behaviors induced by different reward designs, we examine paired evaluation traces from two checkpoints trained on identical data: a QA-only

policy (`eval_qa_rag_50_rp`, `global_step_50`) and a CMI+QA policy (`eval_cmi_qa_rag_20`, `global_step_20`). For each LongMemEval question we reconstruct the per-user memory bank produced by both policies and trace the specific entries that govern the final answer. The five representative cases below cover question types on which the two policies produce different answers.

Case 1: Knowledge Update Tracking. The user originally believed Starbucks Gold required 125 stars, then corrected this to 120 stars in a later session. *Question:* "How many stars do I need to reach the gold level on my Starbucks Rewards app?" *Expected:* 120; *QA-Only:* 125; *CMI+QA:* 120.

Table 12: Memory bank sizes for Case 1 (Knowledge Update).

Policy	Core	Epi.	Sem.	Proc.
QA-Only	7	359	216	139
CMI+QA	3	304	214	74

Key memories.

- `[QA-Only/SEMANTIC]` Gold level requirement: **125 stars**; `[EPISODIC]` corrected to **125 stars** (the wrong value persisted across both stores).
- `[CMI+QA/SEMANTIC]` Gold level requirement: **120 stars**; `[EPISODIC]` confirms the correct requirement is **120 stars**.

Analysis. This pair is consistent with the combined reward preserving knowledge-updating information. The QA-only policy preserved an outdated semantic memory ("**125 stars**") and even propagated the same stale value into episodic memory. In contrast, CMI+QA produced both an updated semantic entry and an episodic entry that explicitly confirms the correction. The action-level CMI signal favors relevant content not covered by the bank, which provides a plausible operation-level explanation for overwriting rather than duplicating outdated facts.

Case 2: Cross-session Information Aggregation. The user discusses several fundraising events across sessions; the correct total is the sum across events. *Question:* "How much money did I raise for charity in total?" *Expected:* $3,750; *QA-Only:* $1,750; *CMI+QA:* $3,750.

Table 13: Memory bank sizes for Case 2 (Cross-session Aggregation).

Policy	Core	Epi.	Sem.	Proc.
QA-Only	1	301	222	141
CMI+QA	4	307	294	69

Key memories.

- `[QA-Only/PROCEDURAL]` "Fundraising strategy for charity cycling event" (generic how-to steps) $\times 3$ near-duplicates; `[SEMANTIC]` previous bake sale success: $1,000.
- `[CMI+QA/PROCEDURAL]` "Raising money for food bank" with specific goal $300; `[EPISODIC]` user raised **$1,000** for children's hospital through bake sale; additional entries record the cycling event and other fundraising activities.

Analysis. The QA-only policy produced 141 procedural entries dominated by generic "how to fundraise" templates, but only 222 semantic entries, burying the individual dollar amounts in procedural noise. CMI+QA reduced procedural entries to 69 (-51%) while increasing semantic entries to 294, preserving each factual amount as a distinct, retrievable entry. This redistribution is consistent with CMI assigning less value to generic procedural steps that repeat across sessions and more value to uncovered session facts.

Case 3: Fact Linkage in Semantic Memory. Answering requires linking a purchase date and an arrival date that are mentioned in different sessions. *Question:* "How many days did it take for my laptop backpack to arrive after I bought it?" *Expected:* 5 days (Jan 15 to Jan 20); *QA-Only:* not answerable; *CMI+QA:* 5 days.

Key memories.

Table 14: Memory bank sizes for Case 3 (Fact Linkage).

Policy	Core	Epi.	Sem.	Proc.
QA-Only	2	298	245	127
CMI+QA	4	349	240	80

- `[QA-Only/EPISODIC]` "Received new laptop backpack on **January 20th**" (no purchase date recorded anywhere in the bank).
- `[CMI+QA/SEMANTIC]` "Purchased from Amazon on **January 15th**"; `[EPISODIC]` backpack arrived on **January 20th**.

Analysis. Answering requires linking the purchase date (Jan 15) and the arrival date (Jan 20). The QA-only policy stored only the arrival event in episodic memory and failed to preserve the purchase date, making the question unanswerable. CMI+QA created a semantic entry that consolidates both facts in co-located form. The CMI+QA policy preserves both dates in retrievable memory. This is consistent with operation-level scoring favoring session information that is not already covered by the bank.

Case 4: Core Memory Enrichment for User Preferences. Preference questions are best answered when core memory captures stable user attributes. *Question:* "My kitchen's becoming a bit of a mess again. Any tips for keeping it clean?" *Expected:* personalized tips that build on existing kitchen knowledge; *QA-Only:* generic tips; *CMI+QA:* personalized advice that references utensil organization and cooking habits.

Table 15: Memory bank sizes for Case 4 (Preference Personalization). Core counts list both number of entries and total character budget.

Policy	Core	Epi.	Sem.	Proc.
QA-Only	1 (2,059 chars)	274	298	155
CMI+QA	4 (3,933 chars)	263	249	104

Key memories.

- `[QA-Only/CORE]` a single line focused on the user's educator identity, **entirely missing** kitchen and food interests.
- `[CMI+QA/CORE]` four lines including food preferences and cooking interests (hot chicken, vegan cooking, mushroom Bolognese), enabling personalized kitchen advice.

Analysis. The QA-only policy built a minimal one-line core memory consisting only of the user's profession. CMI+QA built a richer four-line core profile spanning diverse interests including food and cooking, while simultaneously reducing non-core noise (procedural $155 \rightarrow 104$, semantic $298 \rightarrow 249$). The richer profile is consistent with assigning value to stable user attributes that are not already covered by the existing core memory.

Case 5: Temporal Information Preservation. The user took multiple bus trips in January and a notable train trip in March; the question targets the most recent mode. *Question:* "Which mode of transport did I use most recently, a bus or a train?" *Expected:* train; *QA-Only:* bus; *CMI+QA:* train.

Table 16: Memory bank sizes for Case 5 (Temporal Reasoning).

Policy	Core	Epi.	Sem.	Proc.
QA-Only	5	262	195	139
CMI+QA	2	284	229	89

Key memories.

- `[QA-Only/EPISODIC]` five bus commute entries dominate (Jan 31 commute), with **no temporal summary**; `[PROCEDURAL]` generic commute optimization steps (139 entries total).

- [CMI+QA/SEMANTIC] "Recently increased train usage … 2-hour train ride to visit family on **March 3rd**"; [EPISODIC] train ride March 3rd ($\times 2$); procedural reduced to 89 entries.

Analysis. The QA-only policy stored five episodic bus entries but never produced a temporal summary, so retrieval over-weighted older bus events when answering "most recently". CMI+QA created a semantic entry that consolidates "recently increased train usage … March 3rd" and reduced procedural noise (89 vs. 139 entries). The result is consistent with bank-relative scoring favoring a non-redundant temporal summary. CMI itself does not condition on a future temporal query.

Summary of Findings. Across the five cases we observe five consistent patterns:

- CMI+QA consistently reduces procedural memory entries (avg. -41%), eliminating generic how-to steps that add retrieval noise without contributing factual content.
- CMI+QA better preserves knowledge updates in semantic memory, tracking corrections rather than accumulating contradictory versions of the same fact.
- CMI+QA consolidates temporally-relevant information into semantic summaries that support temporal reasoning at retrieval time.
- CMI+QA builds richer core memory profiles for preference personalization, populating core with diverse stable attributes.
- The action-level CMI signal $I(C_t; m_{\text{new}} \mid M_{t-1})$ scores information contributed relative to the current bank. Together with the auxiliary length and duplication terms, it provides a plausible explanation for the four behavioral patterns above.

K MemoryAgentBench Benchmark Details

MemoryAgentBench Hu et al. (2026) stress-tests memory-augmented agents along axes that LongMemEval and LoCoMo do not isolate, in particular long-range understanding, time-aware reasoning, and the ability to discard outdated facts. Because the benchmark inherits scoring rules from heterogeneous source datasets, the choice of evaluation metric has a sizable effect on relative rankings. This section documents the four query categories and motivates the LLM-as-Judge protocol used in Table 2 of the main paper.

K.1 Query Categories

MemoryAgentBench partitions its evaluation into four capability categories, each backed by a distinct group of source datasets.

Accurate Retrieval (AR). This dimension probes whether a memory system can surface previously stored facts when queried. It aggregates Ruler QA in single-hop (SH-QA) and multi-hop (MH-QA) variants, three LongMemEval (S*) subsets that focus on cross-session recall, and EventQA. AR primarily tests retrieval fidelity because a correct answer requires the relevant fact to survive the writing pipeline and remain addressable at query time.

Test-Time Learning (TTL). TTL evaluates whether an agent can pick up new patterns from the conversation history rather than relying on parametric priors. It contains Multi-Class Classification (MCC), which tests label prediction from in-context examples across five classification datasets, and a recommendation task built on Recsys-Redial (Recom.). Strong AR performance does not guarantee TTL accuracy, since the agent must convert dialogue traces into reusable behavioral templates.

Long-Range Understanding (LRU). LRU measures whether memory representations preserve enough structure to answer questions that require holistic reasoning over the entire history. The included tasks are InfBench summarization (Summ.) and DetectiveQA (DetQA). This dimension penalizes systems that store fragmented snippets without consolidation, since neither summarization nor detective reasoning can be answered from any single retrieved snippet.

Selective Forgetting (SF). SF probes the inverse of retrieval: whether a system correctly overwrites or suppresses information that the user has invalidated. It uses FactConsolidation in single-hop (FC-SH) and multi-hop (FC-MH) settings. Each conversation introduces a fact and a later

correction, so a correct answer must reflect the corrected version and must not be misled by the original fact still appearing verbatim in the dialogue.

Interpreting FC-SH scores. The FC-SH subset evaluates whether an agent answers according to the fact that has been planted in memory, even when that fact conflicts with real-world knowledge. Because the expected answers are intentionally fabricated, the task measures memory fidelity rather than factual reasoning. Under the MemBuilder-P prompt and memory architecture, the Qwen3.7-Max backbone scores 44.2 on this subset. Because all methods use the same Qwen3.7-Max answer model, this aggregate score does not isolate memory fidelity from answer-model behavior: the shared reader may detect a conflict and override retrieved memory with parametric knowledge. For example, when asked "What language does Theodor Herzl speak?" with the planted answer "French", the shared Qwen3.7-Max answer model can identify the adversarial construction and answer "German" from parametric knowledge despite the retrieved fact. We report this score for completeness but caution that it reflects an interaction with the benchmark's adversarial design rather than memory-management capability alone.

K.2 Excluded Subset

The FactConsolidation Multi-Hop (FC-MH) subset is excluded from our MemoryAgentBench evaluation due to detectable counterfactual constructions that render the evaluation insensitive to memory system quality.

FC-MH constructs counterfactual biographical facts (e.g., attributing "association football" to a baseball player) and requires multi-hop reasoning over these synthetic assertions. When the answer model's pretraining corpus contains the true biographical data, it can detect the implausibility of the stored facts and default to parametric knowledge. Models with stronger world knowledge are penalized more heavily under this setup, creating a systematic confound between answer-model capability and memory faithfulness. Since our evaluation framework uses a shared external answer model across all methods, retaining this subset would introduce a confound that is orthogonal to memory management quality.

K.3 Evaluation Metrics and Rationale

For most subtasks, the official MemoryAgentBench scoring uses substring exact match (SEM), defined as

$$\mathrm{SEM}(\hat{a}, a^\star) = \mathbf{1}\big[\,\mathrm{norm}(a^\star) \subseteq \mathrm{norm}(\hat{a})\,\big],$$

where $\hat{a}$ is the predicted answer, $a^\star$ is the gold answer, $\mathbf{1}[\cdot]$ is the indicator function, and $\mathrm{norm}(\cdot)$ lowercases the text, strips punctuation and articles, and collapses whitespace. SEM is convenient and reproducible, yet it interacts poorly with verbose policies in two ways. First, long predictions raise the baseline probability that the normalized gold string appears somewhere in the output by chance, which rewards length over correctness. Second, on Selective Forgetting tasks the dialogue itself contains the outdated fact: a verbose answer that quotes both the old and the corrected value still receives credit under SEM, masking the very behavior the task is designed to test.

To mitigate these issues, we assemble a per-task metric set that is tighter where SEM is unreliable and falls back to task-native metrics where LLM judging is not applicable. The full mapping is listed in Table 17. For the recommendation task we keep the official Recall@5 because the gold targets are movie identifiers rather than free-form text, and for InfBench summarization we use ROUGE-L F1 because LLM judges return near-zero scores for some valid summaries. The remaining tasks are scored by an LLM judge that compares each prediction against the gold answer using the prompt in Appendix I.4.

Table 17: Metrics used in the LLM-as-Judge evaluation (Table 2), selected according to each task's output format and scoring behavior.

Column	Task(s)	Metric	Rationale
SH-QA MH-QA	Ruler QA1 Ruler QA2	LLM-as-Judge LLM-as-Judge	Tolerates paraphrased correct answers that substring match may miss.
LME(S*) EventQA	LongMemEval EventQA	LLM-as-Judge LLM-as-Judge	Open-ended; no single gold string. Open-ended; requires semantic judgment.
MCC	5 ICL tasks	LLM-as-Judge	Strict exact match yields 0 for all models; substring match is too lenient.
Recom.	Recsys Redial	Recall@5 (official)	LLM judge cannot compare movie IDs; edit-distance matching is required.
Summ.	InfBench Sum	ROUGE-L F1	LLM judge returns 0 for SFT models despite valid summaries being generated.
DetQA	DetectiveQA	LLM-as-Judge	Tolerates correct answers with different surface forms.
FC-SH FC-MH	FactConsol. SH FactConsol. MH	LLM-as-Judge LLM-as-Judge	Substring match inflates scores when outdated facts accidentally match.