BEYOND OVERWRITING: SELECTIVE MEMORY INTEGRATION FOR LONG-CONTEXT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Efficiently processing long contexts remains a key challenge for large language models (LLMs). The memorize-while-reading paradigm addresses this challenge by processing documents chunk by chunk and dynamically updating a textual memory, with subsequent work introducing retrieval to recover lost evidence and gating mechanisms for controlling when to update memory and stop reading. Despite these advances, two challenges remain: (i) repeated overwrite-based updates lack explicit protection for previously stored information, potentially causing information loss that is difficult to recover through retrieval; and (ii) optimization based on rule-based rewards provides only indirect guidance for learning when to update memory and stop reading, which may lead to unstable optimization. To address these challenges, we propose GraftMem, which separates evidence acquisition from incremental memory integration through acquisition–integration (AcIn) updates. Combined with selective memory updating, this design effectively mitigates information loss from repeated overwriting. GraftMem also incorporates an early-exit mechanism to improve efficiency. To strengthen training, we combine multi-level reward shaping with decision supervision: alongside trajectory-level and state-level rewards, we directly supervise memory-update and early-exit decisions. Together, these signals alleviate supervision sparsity and support more stable policy optimization. Extensive experiments demonstrate that GraftMem achieves competitive accuracy on long-context reasoning tasks while substantially accelerating inference, achieving a favorable accuracy–efficiency trade-off.

1 INTRODUCTION

Efficient reasoning over extremely long contexts is an important capability for large language models (LLMs) (Xu et al., 2026; Wan et al., 2025; Comanici et al., 2025). For example, reviewing an entire book may require connecting critical evidence scattered across millions of tokens (Li et al., 2025; Wu et al., 2025). However, long-context reasoning remains challenging. The quadratic complexity of attention causes computational costs to grow rapidly with context length (Katharopoulos et al., 2020; Xiao et al., 2024; Bai et al., 2024). Moreover, as the context expands, LLMs often struggle to capture long-range dependencies, resulting in substantial performance degradation. They also cannot directly process inputs that exceed their maximum context window.

To address these limitations, MemAgent (Yu et al., 2026a) introduced a memory-augmented LLM agent based on the memorize-while-reading paradigm (Wang et al., 2025; Shi et al., 2026b). It partitions a long document into fixed-size chunks, processes them sequentially, and repeatedly overwrites a memory from which the final answer is generated. This design reduces the complexity of long-context reasoning to linear time. However, each memory update is generated from the question, the current chunk, and the memory state from the preceding step, while training relies solely on the final-answer reward. Consequently, this paradigm suffers from progressive information loss caused by repeated memory overwriting, as well as sparse and delayed supervision.

Recent studies have improved this paradigm in several ways. ReMemR1 (Shi et al., 2026a) uses callback queries for Top-1 retrieval of historical memory states and introduces state-level rewards. However, retrieval-based recall can be limited by evidence lost during memory formation. GRU-Mem (Sheng et al., 2026) introduces update and exit gates to control rapid memory growth and reduce computational overhead. However, it still relies on overwrite-based memory updates, which

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

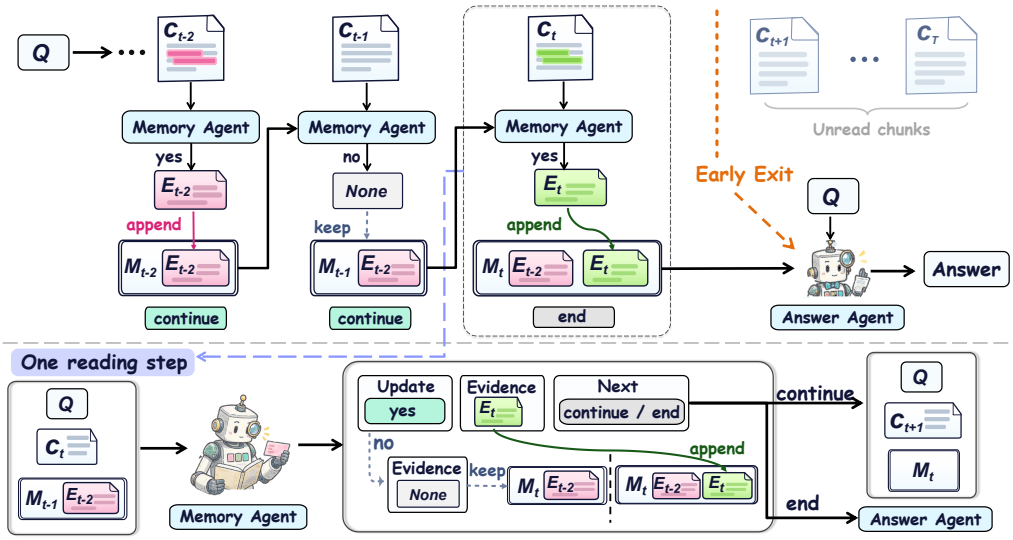


Figure 1: **The memory updating process of GraftMem.** At each step t, the memory agent generates an *Update–Evidence–Next* action (U_t, E_t, V_t), incrementally appends E_t ($U_t = \texttt{True}$) or retains the previous memory ($U_t = \texttt{False}$), and terminates reading when $V_t = \texttt{True}$. The answer agent then generates the answer from the question and terminal memory.

may lose previously stored evidence. MemReread (Ji et al., 2026) decomposes the question and performs additional streaming passes to recover previously omitted evidence. These extra passes can incur substantial inference overhead.

Overall, we identify two remaining challenges in the design and training of existing memory-augmented agents:

- **Evidence Loss from Overwrite-Based Updates.** Existing methods commonly overwrite memory during updates without explicitly isolating critical evidence or protecting previously stored information. Repeated overwriting can therefore omit supporting evidence and reduce performance. Although GRU-Mem uses an update gate to avoid unnecessary updates, it still overwrites memory when an update is made, leaving the risk of information loss unresolved.
- **Indirect Supervision of Control Decisions.** Existing methods use structured decisions to control memory updates and early termination, but do not provide separate supervision targets for these decisions. They therefore use reward-based policy optimization to learn these decisions from relatively sparse feedback, without an objective that directly supervises the correct actions. This may make optimization unstable.

To address these challenges, we propose **GraftMem**, a memory-augmented LLM agent with selective *acquisition–integration* (AcIn) updates that separate explicit evidence acquisition from incremental memory integration. At each reading step t, the agent generates a structured *Update–Evidence–Next* action (U_t, E_t, V_t): *Update* U_t determines whether to update memory, *Evidence* E_t contains newly acquired information, and *Next* V_t determines whether to terminate reading. When an update is needed, the agent explicitly acquires new evidence E_t and integrates it incrementally with M_{t-1} to obtain M_t; otherwise, it retains M_{t-1}. These selective AcIn updates make new evidence explicit and help preserve previously stored evidence without repeatedly overwriting the memory. After the memory transition, V_t determines whether to stop reading early.

To address the indirect supervision of control decisions, we train GraftMem with multi-level rewards and auxiliary decision supervision. Trajectory-level rewards evaluate the overall reasoning process, while state-level rewards provide fine-grained feedback on intermediate decisions and memory updates. Throughout reinforcement learning, we apply auxiliary token-level cross-entropy supervision to the *Update* and *Next* decisions. This direct supervision complements the multi-level rewards, helping the model learn more reliably when to update memory and when to stop reading.

Extensive experiments on long-context reasoning tasks show that GraftMem maintains competitive accuracy while substantially accelerating inference, achieving a favorable accuracy–efficiency trade-off. Further analyses examine memory growth and early-exit behavior, while ablation studies support the respective contributions of AcIn updates and the proposed training framework.

2 METHOD

As illustrated in Figure 1, GraftMem uses structured *Update–Evidence–Next* actions to selectively update memory through AcIn and decide when to stop reading.

2.1 PRELIMINARIES

Task formulation. We consider long-context question answering, where each instance consists of a question Q, a long context C, and a ground-truth answer y. The context length is typically large enough to exceed the maximum input window of the model.

Memory-augmented agents for long-context reasoning. Under the memorize-while-reading paradigm, C is divided into chunks $(c_1, \ldots, c_T)$. A memory agent ϕ_θ processes these chunks sequentially, maintaining a textual memory M_t that summarizes the information deemed important up to step t. The memory is initialized as $M_0 = \varnothing$. At each step, the agent takes the question Q, the current chunk c_t, and the previous memory M_{t-1} as input and generates an updated memory M_t:

$$M_t = \phi_\theta(Q, c_t, M_{t-1}), \qquad t = 1, \ldots, T. \tag{1}$$

After reading all the T chunks, an answer agent ψ_θ generates the answer using Q and M_T:

$$\hat{y} = \psi_\theta(Q, M_T). \tag{2}$$

Here, the two agents share the same parameters θ but use different prompts.

End-to-end RL for memory agents. Building on the multi-conversation training framework of MemAgent, memory agents can be optimized using multi-turn GRPO (Shao et al., 2024). For each training instance, the rollout policy samples a group of G trajectories. Let $o_t^{(g)}$ denote the token sequence generated in conversation t of trajectory g, and let T_g denote the number of conversations, including the final answer turn. The complete objective can be formulated as follows:

$$\mathcal{J}_{\mathrm{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{\sum_{g=1}^{G}\sum_{t=1}^{T_g}|o_t^{(g)}|}\sum_{g=1}^{G}\sum_{t=1}^{T_g}\sum_{i=1}^{|o_t^{(g)}|}\left(\ell_{g,t,i}^{\mathrm{clip}}(\theta) - \beta D_{\mathrm{KL}}[\pi_\theta \| \pi_{\mathrm{ref}}]\right)\right]. \tag{3}$$

Here, $\ell_{g,t,i}^{\mathrm{clip}}$ is the clipped policy surrogate, and β controls KL regularization against the reference policy π_{ref}. The full formulation and notation are provided in Appendix B.2.

2.2 SELECTIVE ACQUISITION–INTEGRATION UPDATES AND EARLY EXIT

GraftMem combines selective memory updating, AcIn updates, and early exit through a structured *Update–Evidence–Next* action:

$$U_t, E_t, V_t = \phi_\theta(Q, c_t, M_{t-1}). \tag{4}$$

Here, Update U_t determines whether to update memory, Evidence E_t contains newly acquired information, and Next V_t determines whether to stop reading (see Appendix B.1 for complete prompts).

Selective acquisition–integration updates. To mitigate information loss caused by repeated memory overwriting, GraftMem separates evidence acquisition from memory integration and performs updates selectively. Specifically, the memory agent first outputs U_t within `<update>` tags. If it judges that c_t contains useful new evidence, it outputs `yes` (U_t = `True`) and writes E_t within `<evidence>` tags. Otherwise, it outputs `no` (U_t = `False`) and writes `None.` in the evidence field. The memory is then updated incrementally when needed:

$$M_t = \begin{cases} M_{t-1}, & U_t = \texttt{False}, \\ \Pi_L(Q, M_{t-1} \oplus E_t), & U_t = \texttt{True}. \end{cases} \tag{5}$$

Here, $\oplus$ denotes concatenation and L is the memory budget. Π_L leaves the concatenated memory unchanged if it fits within L; otherwise, we perform an additional step to consolidate overlapping or redundant evidence within the budget. The consolidation step serves only as an overflow safeguard and was rarely triggered in our experiments, consistent with the analysis in Section 3.3.2.

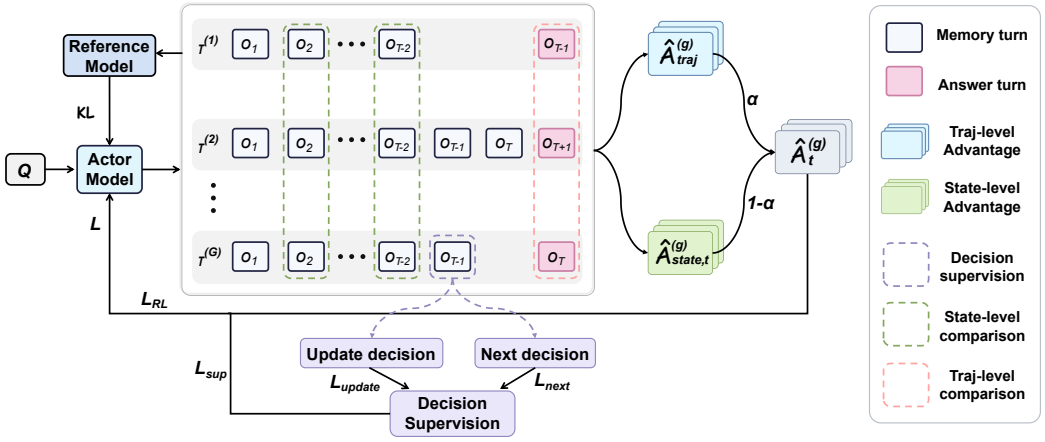


Figure 2: **Overview of GraftMem training.** Trajectory-level and state-level advantages are computed separately and combined for policy optimization, with auxiliary decision supervision.

Early exit. After the evidence field, the agent emits V_t within `<next>` tags to indicate whether the updated memory is sufficient to answer Q. The value `continue` ($V_t =$ `False`) advances to the next chunk, whereas `end` ($V_t =$ `True`) terminates reading after the current memory transition. The answer agent then generates the final answer from Q and the terminal memory.

2.3 Multi-Level Reward Shaping and Decision Supervision

Outcome rewards alone provide limited guidance for intermediate memory updates and control decisions. We therefore train GraftMem with multi-level reward shaping and auxiliary decision supervision, as illustrated in Figure 2.

2.3.1 Multi-Level Reward Design

Outcome reward. For each trajectory g, the outcome reward evaluates its terminal answer $\hat{y}^{(g)}$:

$$R_{\text{out}}^{(g)} = \mathbb{I}\Big(\text{is_equiv}\Big(y, \hat{y}^{(g)}\Big)\Big)\,. \tag{6}$$

Here, y is the ground-truth answer. The reward is 1 if the prediction matches y, and 0 otherwise.

Format reward. GraftMem requires structured outputs from both the memory and answer agents. The `<update>` field must contain either `yes` or `no`; when it contains `no`, the evidence field must contain `None`. The `<next>` field must contain either `continue` or `end`, and the final answer must be enclosed in a `\boxed{}` expression. The format reward is defined as:

$$R_{\text{format}}^{(g)} = \begin{cases} 1, & \text{The format of all turns is correct,} \\ 0, & \text{Otherwise.} \end{cases} \tag{7}$$

Exit reward. We assign each trajectory an exit reward based on its stopping position. Let t^* denote the index of the last chunk containing supporting evidence, and let $\hat{t}^{(g)}$ denote the stopping step of trajectory g, or T if no `end` is produced. This reward is formulated as follows:

$$R_{\text{exit}}^{(g)} = \begin{cases} -0.8, & \hat{t}^{(g)} < t^*, \\ 0, & \hat{t}^{(g)} = t^*, \\ -0.4, & \hat{t}^{(g)} > t^*. \end{cases} \tag{8}$$

Premature exit incurs a larger penalty than late exit because it may stop reading before sufficient evidence has been acquired.

Gate reward. To jointly train when to update memory and when to stop reading, we introduce a gate reward that is positive only when both Update and Next are correct. Specifically, the content of

the <update> tag should be yes if the current chunk contains supporting evidence and no otherwise. The content of the <next> tag should be continue before the last supporting-evidence step t^* and end at or after t^*. The reward is:

$$r_{\text{gate},t}^{(g)} = \begin{cases} 1, & \text{The gate decision is correct,} \\ -1, & \text{Otherwise.} \end{cases} \tag{9}$$

Memory reward. The gate reward guides when to update memory but not whether the memory captures supporting evidence. We therefore introduce a metric-based reward to measure the evidence gained from the current chunk. Let $\mathcal{F}_t$ denote the set of ground-truth supporting facts in chunk c_t. Their coverage in memory M is measured by average recall:

$$\text{Rec}(M, \mathcal{F}_t) = \frac{1}{|\mathcal{F}_t|} \sum_{f \in \mathcal{F}_t} \text{Recall}(f, M). \tag{10}$$

Here, $\text{Recall}(f, M)$ is the proportion of words in f that appear in M; we set $\text{Rec}(M, \varnothing) = 0$. For each memory transition, we define the memory reward as the change in recall:

$$r_{\text{mem},t}^{(g)} = \text{Rec}\Big(M_t^{(g)}, \mathcal{F}_t\Big) - \text{Rec}\Big(M_{t-1}^{(g)}, \mathcal{F}_t\Big). \tag{11}$$

When $U_t =$ False, the memory remains unchanged, so $r_{\text{mem},t}^{(g)} = 0$. This step-level reward encourages updates that increase coverage of supporting facts not yet reflected in memory.

The overall reward. We aggregate the rewards at the trajectory and state levels:

$$R_{\text{traj}}^{(g)} = R_{\text{out}}^{(g)} + R_{\text{format}}^{(g)} + R_{\text{exit}}^{(g)}, \qquad R_{\text{state},t}^{(g)} = r_{\text{gate},t}^{(g)} + r_{\text{mem},t}^{(g)}. \tag{12}$$

2.3.2 Advantage Calculation

Following Shi et al. (2026a), we compute trajectory-level and state-level advantages separately. Let $\mathcal{G}_t$ denote the set of trajectories that reach memory-update step t, and let $G_t = |\mathcal{G}_t|$. We define:

$$\widehat{A}_{\text{traj}}^{(g)} = R_{\text{traj}}^{(g)} - \frac{1}{G} \sum_{k=1}^{G} R_{\text{traj}}^{(k)}, \qquad \widehat{A}_{\text{state},t}^{(g)} = R_{\text{state},t}^{(g)} - \frac{1}{G_t} \sum_{k \in \mathcal{G}_t} R_{\text{state},t}^{(k)}. \tag{13}$$

The trajectory-level advantage measures the relative quality of each complete trajectory against the mean reward over G trajectories. The state-level advantage compares each transition against the mean reward at the same step t across $\mathcal{G}_t$ trajectories. Early termination can make $G_t < G$.

Finally, we combine the two terms to obtain the overall advantage $\widehat{A}_t^{(g)}$:

$$\widehat{A}_t^{(g)} = \alpha \widehat{A}_{\text{traj}}^{(g)} + (1 - \alpha) \widehat{A}_{\text{state},t}^{(g)}, \tag{14}$$

where α is a hyperparameter that balances the two terms.

2.3.3 Decision Supervision and Joint Optimization

Multi-level rewards provide feedback on complete trajectories and intermediate memory states. We additionally supervise the Update and Next decisions at the token level.

Decision supervision. Following the correctness criteria defined for the gate reward, we supervise Update at every processed step and Next only for $t \leq t^*$. We mask direct Next supervision for $t > t^*$ to avoid repeated end targets on delayed trajectories, leaving these decisions to reward-based optimization. The auxiliary loss applies token-level cross-entropy only to the target values within the <update> and <next> tags:

$$\mathcal{L}_{\text{sup}} = \mathcal{L}_{\text{update}} + \mathcal{L}_{\text{next}}. \tag{15}$$

Joint training objective. Inspired by prior work on supervision-guided reinforcement learning (Zeng et al., 2025; Zhang et al., 2026), we retain this auxiliary objective throughout policy optimization. The overall training objective is:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{RL}}(\theta) + \lambda_{\text{sup}} \mathcal{L}_{\text{sup}}(\theta), \qquad \mathcal{L}_{\text{RL}}(\theta) = -\mathcal{J}_{\text{GRPO}}(\theta), \tag{16}$$

where $\mathcal{J}_{\text{GRPO}}$ is the multi-turn GRPO objective in Equation 3, and λ_{sup} controls the contribution of decision supervision.

Table 1: Results on HotpotQA, 2WikiMultiHopQA, and SQuAD. Values are normalized sub-EM accuracy (%). **Bold** and underlined values denote the best and second-best results, respectively.

(a) HotpotQA (In-Distribution)

Method	7K	14K	28K	56K	112K	224K	448K	896K
Qwen2.5-Instruct-7B	60.16	59.38	60.94	42.97	26.56	22.66	14.06	7.81
MemAgent-7B	67.97	68.75	62.50	60.16	63.28	59.38	62.50	60.94
ReMemR1-7B	67.97	67.19	64.84	61.72	64.84	63.28	64.06	63.28
GraftMem-7B (w/o Exit)	68.75	69.53	65.63	**67.97**	**68.75**	**67.19**	65.63	**66.41**
GraftMem-7B (w Exit)	**71.88**	**72.66**	**66.41**	65.63	67.19	66.41	**67.97**	64.84

(b) 2WikiMultiHopQA (Out-of-Distribution)

Method	7K	14K	28K	56K	112K	224K	448K	896K
Qwen2.5-Instruct-7B	59.38	50.00	45.31	25.78	19.53	9.38	7.81	6.25
MemAgent-7B	63.28	58.59	57.03	58.59	53.13	53.91	51.56	49.22
ReMemR1-7B	61.72	60.16	64.06	57.03	54.69	51.56	51.56	50.00
GraftMem-7B (w/o Exit)	63.28	60.94	**65.63**	**63.28**	**58.59**	**57.81**	**54.69**	**53.13**
GraftMem-7B (w Exit)	**64.84**	**61.72**	62.50	59.38	56.25	54.69	53.13	51.56

(c) SQuAD (Out-of-Distribution)

Method	7K	14K	28K	56K	112K	224K
Qwen2.5-Instruct-7B	81.25	78.13	64.06	43.75	24.22	14.84
MemAgent-7B	83.59	82.03	78.13	76.56	74.22	72.66
ReMemR1-7B	**85.94**	**82.81**	81.25	78.91	77.34	75.78
GraftMem-7B (w/o Exit)	83.59	79.69	**82.81**	**82.81**	**79.69**	**78.13**
GraftMem-7B (w Exit)	84.38	80.47	81.25	80.47	78.91	77.34

3 EXPERIMENTS

In this paper, we explore the effectiveness of GraftMem by answering the following questions:

- **RQ1:** Does GraftMem improve accuracy and efficiency over existing methods?
- **RQ2:** How do selective AcIn updates and early exit shape agent behavior?
- **RQ3:** Do our training framework and AcIn updates improve performance?

3.1 EXPERIMENTAL SETUP

Models. We conduct experiments using two model sizes of the Qwen2.5 backbone: Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct (Yang et al., 2024). Full results at the 3B scale are provided in Appendix C.1.

Datasets. Following MemAgent, we construct long-context training data from HotpotQA (Yang et al., 2018). We evaluate on in-distribution (ID) HotpotQA and out-of-distribution (OOD) 2WikiMultiHopQA (Ho et al., 2020) and SQuAD (Rajpurkar et al., 2016). HotpotQA and 2WikiMultiHopQA assess multi-hop reasoning, whereas SQuAD assesses single-hop question answering. Following the benchmark construction of MemAgent, we evaluate context lengths from 7K to 896K on HotpotQA and 2WikiMultiHopQA. For SQuAD, the range ends at 224K because fewer distinct passages are available. Further dataset and implementation details are provided in Appendix B.

Baselines. We compare GraftMem with three categories of baselines: (1) general LLMs, including Qwen2.5-Instruct and DeepSeek-R1-Distill-Qwen (Guo et al., 2025); (2) long-context LLMs, including Qwen2.5-Instruct-1M (Yang et al., 2025); and (3) memory-augmented LLM agents, including MemAgent (Yu et al., 2026a) and ReMemR1 (Shi et al., 2026a). Due to space limitations, results for some of these baselines and additional comparisons are provided in Appendix C.1.

3.2 PERFORMANCE AND EFFICIENCY (RQ1)

Better performance. Table 1 compares GraftMem with other methods at the 7B scale. Overall, GraftMem achieves the highest accuracy on most of the tasks. On HotpotQA, GraftMem (w/o Exit) achieves up to 7.81% and 6.25% higher accuracy than MemAgent and ReMemR1, respectively. For

Table 2: Comparison of training efficiency across different methods.

Method	Avg. Time / Step (s) ↓	Peak Memory Usage (GiB) ↓
MemAgent	1508.96	**61.86**
ReMemR1	1586.64	65.69
GraftMem	**1333.19**	64.48

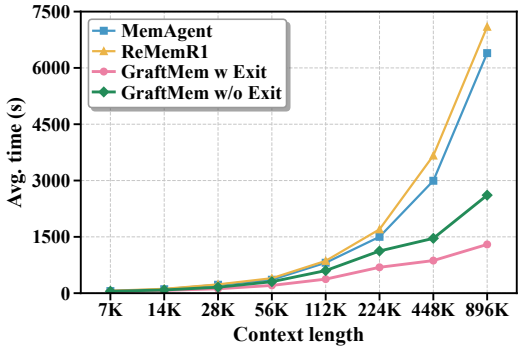


Figure 3: Inference time across context lengths.

GraftMem (w Exit), the corresponding gains are 7.03% and 5.47%. The gains extend to the out-of-distribution tasks. On 2WikiMultiHopQA, GraftMem improves over MemAgent and ReMemR1 by up to 8.60% and 6.25%, respectively. On SQuAD, the corresponding maximum improvements are 6.25% and 3.90%. These results show that GraftMem maintains strong performance as the context length increases and performs well on out-of-distribution data.

Training and inference efficiency. GraftMem achieves a favorable trade-off between accuracy and efficiency. During training, it reduces the average time per step by 11.6% and 16.0% compared with MemAgent and ReMemR1, respectively. Its peak GPU memory usage lies between the two baselines, indicating an acceptable memory footprint despite the additional decision supervision. During inference, GraftMem (w/o Exit) generally achieves higher accuracy than GraftMem (w Exit) at longer context lengths. This gap may reflect premature exits in the w Exit variant, as analyzed in Section 3.3.2. Nevertheless, Figure 3 shows that GraftMem (w Exit) substantially reduces inference time. At 896K, it achieves $2.0\times$, $4.9\times$, and $5.5\times$ speedups over GraftMem (w/o Exit), MemAgent, and ReMemR1, respectively. We therefore use GraftMem (w Exit) as the default configuration in subsequent experiments.

3.3 Analysis of Selective AcIn Updates and Early Exit (RQ2)

3.3.1 Distant Evidence Challenge

To evaluate evidence retention and integration over long reading spans, we construct a distant-evidence setting on HotpotQA. For each question, we insert distractor documents between the two supporting documents so that the gap between them exceeds half of the total context length.

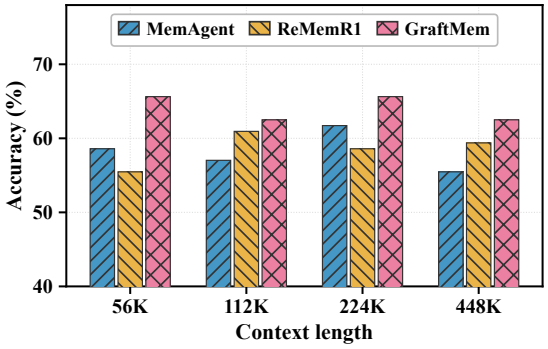


Figure 4: Accuracy under the distant evidence setting.

As shown in Figure 4, GraftMem achieves the best performance in this setting, consistent with its selective AcIn update mechanism. Selective updating avoids unnecessary memory updates, whereas AcIn separates evidence acquisition from incremental memory integration. This design can preserve supporting evidence acquired earlier while incorporating new evidence. In contrast, sequential memory overwriting in MemAgent may discard previously stored information. In ReMemR1, a larger number of intervening memory states may make relevant historical evidence harder to retrieve. These results suggest that selective AcIn updates are effective when supporting evidence is widely separated.

3.3.2 Memory and Reading Dynamics

Selective AcIn updates. We study how selective AcIn updates affect memory growth during long-context inference by measuring memory length after each processed chunk. For ReMemR1, this measure includes both the current memory and recalled memory.

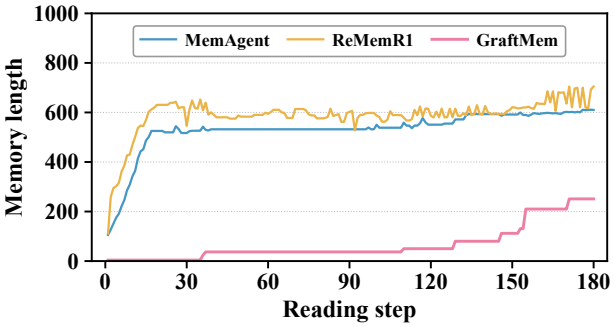
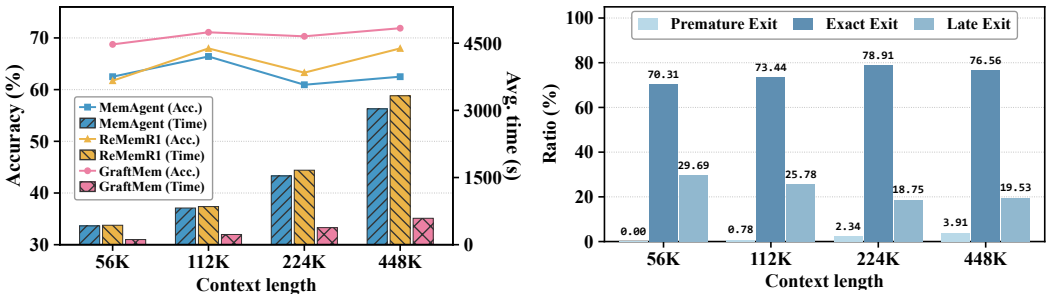


Figure 5: Memory size dynamics on HotpotQA (896K).

As shown in Figure 5, the memory length for MemAgent and ReMemR1 rises rapidly during the early reading steps, whereas GraftMem maintains a smaller memory with sparse, stepwise growth. This pattern is consistent with selective AcIn updates: GraftMem incrementally updates the memory only on a few critical chunks. By contrast, MemAgent updates its memory at each reading step, while ReMemR1 further introduces retrieval over historical memory. These observations indicate that GraftMem enables incremental memory updates during long-context reasoning while avoiding excessive memory growth. We further analyze GraftMem's performance when memory-budget consolidation is triggered (Appendix C.3).

Early-exit behavior. To investigate the behavior of the early-exit mechanism, we construct a Top20 setting. Specifically, we constrain the last supporting document to appear within the first 20% of the overall context span. We apply this setting to HotpotQA. As shown in Figure 6(a), GraftMem consistently achieves the highest accuracy and, at 448K, achieves inference speedups of $5.1\times$ and $5.6\times$ over MemAgent and ReMemR1, respectively. We further calculate the ratios of premature, exact, and late exits under the Top20 setting. As shown in Figure 6(b), in most cases, GraftMem is able to identify the position of the last supporting evidence and exit exactly at that point. This indicates that the early-exit mechanism can operate flexibly under such imbalanced-evidence scenarios.



(a) Performance under the Top20 setting.

(b) The ratios of premature, exact, and late exit.

Figure 6: Early-exit behavior on Top20 tasks.

3.4 Ablation Study (RQ3)

In this section, we conduct a series of ablation studies using Qwen2.5-3B-Instruct.

3.4.1 Effect of Multi-Level Rewards and Decision Supervision

Impact of α. We evaluate $\alpha \in \{1.0, 0.8, 0.5\}$ to examine the balance between trajectory-level and state-level advantages. As shown in Table 3, we find that $\alpha = 1.0$ (i.e., using only trajectory-level advantages) leads to less effective optimization. Assigning equal weight to both levels ($\alpha = 0.5$) also hinders the optimization of final correctness. In contrast, a mild value of 0.8 (i.e., balancing the two levels) achieves the best overall performance across context lengths. Based on these results, we adopt $\alpha = 0.8$ by default. Further analysis of training dynamics appears in Appendix C.6.1.

Effect of decision supervision. We remove auxiliary decision supervision by setting $\lambda_{\text{sup}} = 0$ to examine its contribution. As shown in Table 4, removing decision supervision generally reduces performance, particularly at longer context lengths. This suggests that direct supervision complements reward-based policy optimization in learning memory-update and termination decisions. Its benefit is especially evident at longer context lengths. We further analyze training dynamics in Appendix C.6.2.

Table 3: Accuracy on HotpotQA with different α values.

Method	α	7K	14K	28K	56K	112K	224K	448K	896K
GraftMem	1.0	62.50	**65.63**	58.59	52.34	54.69	55.47	51.56	50.00
	0.8	**65.63**	64.84	**60.94**	**57.81**	**58.59**	**58.59**	**54.69**	**52.34**
	0.5	63.28	62.50	56.25	55.47	57.03	54.69	52.34	48.44

Table 4: Ablation of decision supervision and AcIn updates on HotpotQA.

Method	7K	14K	28K	56K	112K	224K	448K	896K
w/o decision supervision	64.84	**66.41**	55.47	53.13	53.13	50.78	46.88	48.44
w/o AcIn updates	64.06	65.63	**61.72**	55.47	56.25	52.34	51.56	49.22
GraftMem (Full)	**65.63**	64.84	60.94	**57.81**	**58.59**	**58.59**	**54.69**	**52.34**

3.4.2 Effect of AcIn Updates

To analyze the contribution of the AcIn update mechanism, we replace AcIn updates with overwrite-based updates. Specifically, when $U_t = \texttt{True}$, the agent generates a new memory M_t instead of appending E_t to the previous memory.

As shown in Table 4, the overwrite-based variant generally performs worse than the default GraftMem configuration. At shorter context lengths, the two achieve similar accuracy, and the overwrite-based variant occasionally performs slightly better. This small gap is consistent with supporting evidence spanning fewer chunks. At longer context lengths, however, the overwrite-based variant consistently achieves lower accuracy. Repeated overwriting may create more opportunities to omit previously acquired evidence as context length increases. The growing performance gap is consistent with this risk and supports the effectiveness of AcIn update mechanism. Further analysis of the training dynamics is provided in Appendix C.6.3.

4 Related Work

Long-context Reasoning. Existing approaches improve long-context processing through architectural changes and context-window extension. Sparse attention reduces computation by restricting attention connections (Beltagy et al., 2020). Linear attention and state-space models offer alternative ways to process long sequences efficiently (Katharopoulos et al., 2020; Gu & Dao, 2023). Position Interpolation and YaRN extend pretrained context windows through positional adaptations (Chen et al., 2023; Peng et al., 2024). While these methods have shown effectiveness in handling long-context reasoning, they still suffer from performance degradation in overly long contexts.

Memory LLM Agents. Memory mechanisms allow agents to retain information beyond their immediate input windows (Li et al., 2025). Under the memorize-while-reading paradigm, agents process documents chunk by chunk, update a textual memory, and answer from the final memory (Yu et al., 2026a). Reinforcement learning has improved this paradigm on long-context reasoning, but information loss from repeated memory overwriting and sparse supervision of intermediate decisions remain challenges. Appendix A provides a more detailed discussion.

5 Conclusion

In this paper, we proposed GraftMem to address two challenges in memory-augmented LLM agents: information loss from repeated memory overwriting and indirect supervision of memory-update and stopping decisions. Selective AcIn updates separate evidence acquisition from incremental memory integration, helping preserve previously stored evidence while limiting memory growth. Early exit reduces unnecessary reading and improves inference efficiency. We further developed a training framework that combines multi-level reward shaping with auxiliary decision supervision to support more stable policy optimization. Experiments across diverse long-context reasoning tasks show that GraftMem achieves competitive accuracy while substantially accelerating inference, yielding a favorable accuracy–efficiency trade-off. We further analyze how selective AcIn updates and early exit shape the agent's behavior. Ablation studies further confirm the effectiveness of AcIn updates and the proposed training framework.

AI USE STATEMENT

During the preparation of this manuscript, we used LLMs as writing assistants. Some sentences were drafted or refined by LLMs, but all text was reviewed and finalized by the human authors. We also used LLMs to assist with literature searches and the identification of related work. LLMs were not used for any core intellectual contributions, including research ideation or methodology formulation. We take full responsibility for the final content of this work, including all scientific claims, analyses, and conclusions.

ETHICS STATEMENT

Our research is limited to computational experiments on publicly available benchmarks consisting of publicly sourced text and does not involve human subjects. This work focuses on foundational research in long-context reasoning. We acknowledge that LLM-based systems may exhibit biases and factual inaccuracies, and that hallucinations during long-context processing may mislead users and adversely affect their decisions. When applied to private data, our method's memory mechanism may also retain sensitive information. Downstream applications should therefore consider data privacy, access controls, and the reliability of generated answers, and should be evaluated for fairness, transparency, and potential discriminatory impacts.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide a source code package in the supplementary materials. The package includes:

- Configuration files and instructions for setting up the experimental environment.
- Complete code for generating our evaluation datasets from publicly available benchmarks (HotpotQA, 2WikiMultiHopQA, and SQuAD).
- Code and scripts for training GraftMem.
- Evaluation scripts for the main experiments.

In addition, Appendix B provides dataset and implementation details, including experimental configurations. We hope these materials enable researchers to reproduce and extend our work.

REFERENCES

Chenxin An, Fei Huang, Jun Zhang, Shansan Gong, Xipeng Qiu, Chang Zhou, and Lingpeng Kong. Training-free long-context scaling of large language models. *arXiv preprint arXiv:2402.17463*, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

Zhuoen Chen, Dongfang Li, Meishan Zhang, Baotian Hu, and Min Zhang. Dynamic long context reasoning over compressed memory via end-to-end reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8064–8083, 2026.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Baibei Ji, Xiaoyang Weng, Juntao Li, Zecheng Tang, Yihang Lou, and Min Zhang. Memreread: Enhancing agentic long-context reasoning via memory-guided rereading. *arXiv preprint arXiv:2605.10268*, 2026.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 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.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Xinze Li, Yushi Bai, Bowen Jin, Fengbin Zhu, Liangming Pan, and Yixin Cao. Long context vs. rag: Strategies for processing long documents in llms. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4110–4113, 2025.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

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.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

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

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *International Conference on Learning Representations*, volume 2024, pp. 31932–31951, 2024.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 2383–2392, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Leheng Sheng, Yongtao Zhang, Wenchang Ma, Yaorui Shi, Ting Huang, Xiang Wang, An Zhang, Ke Shen, and Tat-Seng Chua. When to memorize and when to stop: Gated recurrent memory for long-context reasoning. *arXiv preprint arXiv:2602.10560*, 2026.

Yaorui Shi, Yuxin Chen, Siyuan Wang, Sihang Li, Hengxing Cai, Qi Gu, Xiang Wang, and An Zhang. Look back to reason forward: Revisitable memory for long-context llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 78055–78082, 2026a.

Yaorui Shi, Shugui Liu, Yu Yang, Wenyu Mao, Yuxin Chen, Qi Gu, Hui Su, Xunliang Cai, Xiang Wang, and An Zhang. Memocr: Layout-aware visual memory for efficient long-horizon reasoning. *arXiv preprint arXiv:2601.21468*, 2026b.

Fanqi Wan, Weizhou Shen, Shengyi Liao, Yingcheng Shi, Chenliang Li, Ziyi Yang, Ji Zhang, Fei Huang, Jingren Zhou, and Ming Yan. Qwenlong-l1: Towards long-context large reasoning models with reinforcement learning. *arXiv preprint arXiv:2505.17667*, 2025.

Xinyu Wang, Mingze Li, Peng Lu, Xiao-Wen Chang, Lifeng Shang, Jinping Li, Fei Mi, Prasanna Parthasarathi, and Yufei Cui. Infmem: Learning system-2 memory control for long-context agent. *arXiv preprint arXiv:2602.02704*, 2026.

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

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, et al. Resum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, volume 2024, pp. 21875–21895, 2024.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

An Yang, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoyan Huang, Jiandong Jiang, Jianhong Tu, Jianwei Zhang, Jingren Zhou, Junyang Lin, Kai Dang, Kexin Yang, Le Yu, Mei Li, Minmin Sun, Qin Zhu, Rui Men, Tao He, Weijia Xu, Wenbiao Yin, Wenyuan Yu, Xiafei Qiu, Xingzhang Ren, Xinlong Yang, Yong Li, Zhiying Xu, and Zipeng Zhang. Qwen2.5-1m technical report, 2025. URL https://arxiv.org/abs/2501.15383.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. In *International Conference on Learning Representations*, volume 2026, pp. 39458–39486, 2026a.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026b.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Min Zeng, Jingfei Sun, Xueyou Luo, Shiqi Zhang, Li Xie, Caiquan Liu, and Xiaoxin Chen. GTA: Supervised-guided reinforcement learning for text classification with large language models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 1050–1060, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.56. URL https://aclanthology.org/2025.findings-emnlp.56/.

Wenhao Zhang, Yuexiang Xie, Yuchang Sun, Yanxi Chen, Guoyin Wang, Yaliang Li, Bolin Ding, and Jingren Zhou. On-policy rl meets off-policy experts: Harmonizing supervised fine-tuning and reinforcement learning via dynamic weighting. In *International Conference on Learning Representations*, volume 2026, pp. 120693–120726, 2026.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization, 2025. URL https://arxiv.org/abs/2507.18071.

A Related Work

Long-context Reasoning. Long-context reasoning requires integrating evidence across lengthy inputs, yet models can struggle when relevant information appears in the middle of the context (Liu et al., 2024). Existing approaches broadly address computational efficiency and context-window extension. Sparse attention reduces computation by restricting connections through local windows, global tokens, or other sparse patterns (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). Linear attention reformulates attention computation to achieve linear complexity (Katharopoulos et al., 2020), while state-space models efficiently summarize sequence history through recurrent states (Gu et al., 2021; Gu & Dao, 2023). For context extension, Position Interpolation and YaRN adapt RoPE-based positional representations to accommodate longer inputs (Chen et al., 2023; Peng et al., 2024). Dual Chunk Attention further supports length extension through chunk-based attention and adjusted relative positions (An et al., 2024). While these methods have shown effectiveness in handling long-context reasoning, they still suffer from performance degradation in overly long contexts

Memory LLM Agents. Memory mechanisms enable LLM agents to retain and access information beyond their immediate context windows Hsieh et al. (2024); Maharana et al. (2024); Bai et al. (2024). MemGPT and MemoryOS organize memory hierarchically to support information management across interactions (Packer et al., 2023; Kang et al., 2025). For long-context reasoning, MemAgent (Yu et al., 2026a) introduces a memorize-while-reading paradigm combined with reinforcement learning: agents read long documents chunk by chunk, update a memory state during reading, and answer questions using the final memory. ReMemR1 (Shi et al., 2026a) generates callback queries to retrieve relevant historical memory states and introduces state-level rewards, allowing previous memories to be revisited during subsequent updates. GRU-Mem (Sheng et al., 2026) introduces update and exit gates to address the risk of memory explosion and the lack of an exit mechanism. InfMem (Wang et al., 2026) learns selective memory compression and allows retrieval of relevant content from the source document to refine memory, while MemReread (Ji et al., 2026) decomposes the question and performs additional streaming passes to recover previously omitted evidence. LycheeMemory (Chen et al., 2026) combines compressed KV representations with selective memory recall to reduce inference overhead. Combined with reinforcement learning, these methods have substantially improved LLM performance, but challenges remain in evidence acquisition and memory loss.

Reinforcement Learning in LLMs. Reinforcement learning has become an important approach to LLM post-training. Early work uses human preference feedback with Proximal Policy Optimization (PPO) to align model outputs with user intentions (Ouyang et al., 2022; Schulman et al., 2017). More recent work uses verifiable outcome rewards to improve reasoning (Guo et al., 2025). Group Relative Policy Optimization (GRPO) estimates advantages from groups of sampled responses without a separate value model (Shao et al., 2024). Subsequent methods improve optimization efficiency and stability: DAPO introduces decoupled clipping and dynamic sampling (Yu et al., 2026b), while GSPO uses sequence-level importance ratios and clipping (Zheng et al., 2025). Beyond single-response reasoning, Search-R1 trains agents to interleave reasoning with search (Jin et al., 2025). Other works combine supervised objectives with policy optimization. GTA (Zeng et al., 2025) applies cross-entropy supervision to an initial guess and reinforcement learning to the final answer and output format. CHORD (Zhang et al., 2026) incorporates supervised fine-tuning as an auxiliary objective during on-policy RL, using global and token-level weighting to balance expert supervision with policy exploration.

B Implementation Details

B.1 Prompt Templates

Figure 7 presents the prompt templates used by memory agent ϕ_θ and answer agent ψ_θ in GraftMem. The placeholders {`question`}, {`memory`}, and {`chunk`} are replaced with the question, current memory, and current document chunk, respectively.

Prompt:

```
You are presented with a problem, one section of a long context that may contain
the answer to the problem, and a previous memory. Please inspect the current
section carefully, and produce one incremental memory write.
First decide update: whether the current section contains new information that
may help answer the problem. If update=no, write None. as the evidence and keep
the previous memory unchanged. If update=yes, write only the new evidence from
the current section; the controller will append it to the previous memory.
If the section contains no useful new evidence, output:
<update>no</update>
<evidence>None.</evidence>
If the section contains useful new evidence, output:
<update>yes</update>
<evidence>...</evidence>
Only after completing the evidence write, decide whether the memory after this
update is sufficient to answer the problem. Output:
<next>continue</next> or <next>end</next>
Return end when the memory is sufficient to answer the problem. If more evidence
is still needed, return continue.

<problem> {question} </problem>
<memory> {memory} </memory>
<section> {chunk} </section>
```

(a) Prompt for the memory agent ϕ_θ in GraftMem.

Prompt:

```
You are presented with a problem and a previous memory. Answer the problem based
on the memory and put the answer in \boxed{}.

<problem> {question} </problem>
<memory> {memory} </memory>

Your answer:
```

(b) Prompt for the answer agent ψ_θ in GraftMem.

Prompt:

```
You are presented with a problem and an accumulated memory that exceeds the
memory budget. Consolidate the evidence in the memory into a shorter version
that fits within memory_budget tokens.
Please merge duplicate or semantically overlapping evidence and remove redundant
wording.

Return only the consolidated memory.
<problem> {question} </problem>
<memory> {memory} </memory>

Consolidated memory:
```

(c) Prompt for memory consolidation when the memory budget is exceeded.

Figure 7: Prompt templates for memory and answer agents.

B.2 Full Expression of the Training Objective

For each training instance $(Q, C, y) \sim \mathcal{D}$, the rollout policy $\pi_{\theta_{\text{old}}}$ samples a group of G trajectories. We write the g-th trajectory as $\tau^{(g)} = (o_1^{(g)}, \ldots, o_{T_g}^{(g)})$, where T_g denotes the total number of generated conversations in trajectory g. The final conversation $o_{T_g}^{(g)}$ is generated by the answer agent, while $o_1^{(g)}, \ldots, o_{T_g-1}^{(g)}$ are generated by the memory agent. Due to early termination, T_g may vary across trajectories.

The full multi-turn GRPO objective and its clipped policy surrogate are defined as:

$$\begin{aligned}
\mathcal{J}_{\mathrm{GRPO}}(\theta) &= \mathbb{E}_{(Q,C,y)\sim\mathcal{D},\,\{\tau^{(g)}\}_{g=1}^{G}\sim\pi_{\theta_{\mathrm{old}}}} \\
&\left[\frac{1}{\sum_{g=1}^{G}\sum_{t=1}^{T_g}|o_t^{(g)}|}\sum_{g=1}^{G}\sum_{t=1}^{T_g}\sum_{i=1}^{|o_t^{(g)}|}\left(\ell_{g,t,i}^{\mathrm{clip}}(\theta)-\beta D_{\mathrm{KL}}[\pi_\theta\|\pi_{\mathrm{ref}}]\right)\right], \\
\ell_{g,t,i}^{\mathrm{clip}}(\theta) &= \min\left\{\rho_{g,t,i}(\theta)\widehat{A}_t^{(g)},\,\mathrm{clip}(\rho_{g,t,i}(\theta),1-\epsilon,1+\epsilon)\,\widehat{A}_t^{(g)}\right\}, \\
\rho_{g,t,i}(\theta) &= \frac{\pi_\theta\left(o_{t,i}^{(g)}\mid h_t^{(g)},o_{t,<i}^{(g)}\right)}{\pi_{\theta_{\mathrm{old}}}\left(o_{t,i}^{(g)}\mid h_t^{(g)},o_{t,<i}^{(g)}\right)}.
\end{aligned} \tag{17}$$

Here, $\widehat{A}_t^{(g)}$ is the advantage assigned to conversation t in trajectory g, and $o_{t,<i}^{(g)}$ denotes the preceding generated tokens within that conversation. The policies $\pi_{\theta_{\mathrm{old}}}$ and π_{ref} denote the rollout and reference policies, respectively. The coefficients ϵ and β control policy clipping and KL regularization. The input prompt $h_t^{(g)}$ depends on the conversation type. For $t < T_g$, the memory-update prompt contains the question Q, the current chunk c_t, and the previous memory $M_{t-1}^{(g)}$. For $t = T_g$, the answer-generation prompt contains the question and terminal memory.

B.3 Algorithm

Algorithms 1 and 2 summarize the overall workflow and training framework of GraftMem, respectively. The overall workflow processes context chunks sequentially through Update–Evidence–Next actions, combining selective evidence acquisition, incremental memory integration, and optional early exit. The training framework computes trajectory-level and state-level advantages from grouped rollouts and combines them for policy optimization. Auxiliary supervision of Update and Next decisions complements these reward signals through a joint training objective.

Algorithm 1 Long-Context Reasoning with GraftMem

Require: Question Q, context C, chunk size s, memory budget L, memory agent ϕ_θ, answer agent ψ_θ, early-exit flag b_{exit}
Ensure: Predicted answer $\hat{y}$

Partition C into consecutive chunks $(c_1,\ldots,c_T)$ of at most s tokens
$M_0 \leftarrow \varnothing$
$t_{\mathrm{last}} \leftarrow 0$
for $t = 1,\ldots,T$ **do**
  $(U_t, E_t, V_t) \leftarrow \phi_\theta(Q, c_t, M_{t-1})$ ▷ Update–Evidence–Next
  if $U_t =$ `True` **then**
    $\widetilde{M}_t \leftarrow M_{t-1} \oplus E_t$ ▷ Append new evidence
    $M_t \leftarrow \Pi_L(Q, \widetilde{M}_t)$ ▷ Consolidate only if over budget
  else
    $M_t \leftarrow M_{t-1}$ ▷ Retain previous memory
  end if
  $t_{\mathrm{last}} \leftarrow t$
  if $b_{\mathrm{exit}} =$ `True` **and** $V_t =$ `True` **then**
    break ▷ Terminate after the memory update
  end if
end for
$\hat{y} \leftarrow \psi_\theta(Q, M_{t_{\mathrm{last}}})$
return $\hat{y}$

Algorithm 2 GraftMem Training

Require: π_θ, π_{ref}, $\mathcal{D}$, G, α, λ_{sup}, η

while not converged **do**
  Sample annotated $(Q, C, y) \sim \mathcal{D}$
  $\theta_{\text{old}} \leftarrow \theta$
  $\{\tau^{(g)}\}_{g=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid Q, C)$
    $R^{(g)}_{\text{traj}} \leftarrow R^{(g)}_{\text{out}} + R^{(g)}_{\text{format}} + R^{(g)}_{\text{exit}}$
    $R^{(g)}_{\text{state},t} \leftarrow r^{(g)}_{\text{gate},t} + r^{(g)}_{\text{mem},t}$
  $\mathcal{G}_t \leftarrow \{g : t < T_g\}$
    $\widehat{A}^{(g)}_{\text{traj}} \leftarrow R^{(g)}_{\text{traj}} - \frac{1}{G}\sum_{k=1}^{G} R^{(k)}_{\text{traj}}$
    $\widehat{A}^{(g)}_{\text{state},t} \leftarrow R^{(g)}_{\text{state},t} - \frac{1}{|\mathcal{G}_t|}\sum_{k\in\mathcal{G}_t} R^{(k)}_{\text{state},t}$
  $\widehat{A}^{(g)}_{t} \leftarrow \alpha \widehat{A}^{(g)}_{\text{traj}} + (1-\alpha)\widehat{A}^{(g)}_{\text{state},t}$
  Construct Update/Next targets ▷ Mask Next for $t > t^*$
  Compute $\mathcal{J}_{\text{GRPO}}(\theta)$ using Eq. (17)
    $\mathcal{L} \leftarrow -\mathcal{J}_{\text{GRPO}}(\theta) + \lambda_{\text{sup}}(\mathcal{L}_{\text{update}} + \mathcal{L}_{\text{next}})$
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
end while
return π_θ

B.4 Training Settings

We implement GraftMem using the verl framework[1], with efficient trajectory generation powered by the SGLang engine[2]. We use Fully Sharded Data Parallelism (FSDP) for distributed training and bfloat16 precision for both training and evaluation.

Each training sample contains approximately 30K input tokens. We train both models with a chunk size of 5,000 tokens and a maximum response length of 2,048 tokens. We adopt a clip ratio of 0.2 and a learning rate of 1×10^{-6}. During training, responses are sampled with a temperature of 1.0 and top-p of 0.999. The training batch size is 64, with a rollout group size of $G = 8$ and a Micro training batch size of 32. Table 5 summarizes the training hyperparameters.

The 3B model is trained on 4 NVIDIA A800 GPUs for approximately 90 hours, while the 7B model is trained on 8 NVIDIA A800 GPUs for approximately 150 hours.

Table 5: Primary hyperparameters used in training.

Hyperparameter	**Value**
Chunk size	5,000
Memory budget L	1,024
Maximum response length	2,048
Training batch size	64
Micro training batch size	32
Rollout group size G	8
Sampling temperature	1.0
Top-p	0.999
Clip ratio ϵ	0.2
KL coefficient β	0.001
Learning rate	1×10^{-6}
Warmup steps	20
Early-exit activation step	30
Decision supervision weight λ_{sup}	0.001

[1] https://github.com/volcengine/verl

[2] https://github.com/sgl-project/sglang

B.5 Benchmarks and Baselines

Benchmarks. We evaluate GraftMem on long-context question-answering tasks constructed from HotpotQA, 2WikiMultiHopQA, and SQuAD.

- **HotpotQA** (Yang et al., 2018) is a Wikipedia-based multi-hop question-answering benchmark with sentence-level supporting-fact annotations. It includes bridge questions, which require connecting information through an intermediate entity, and comparison questions, which require comparing attributes of two entities.
- **2WikiMultiHopQA** (Ho et al., 2020) combines Wikipedia passages with structured information from Wikidata to construct questions requiring multiple reasoning steps. In addition to answers and supporting facts, it provides explicit evidence paths that describe the relations connecting the question to the answer.
- **SQuAD** (Rajpurkar et al., 2016) is a reading-comprehension benchmark containing human-written questions whose answers are text spans in Wikipedia passages. Unlike the two multi-hop benchmarks, its questions are grounded in a single passage.

Baselines. We compare GraftMem with the following memory-augmented LLM agents:

- **MemAgent** (Yu et al., 2026a) processes documents sequentially in chunks, updating a textual memory through overwriting and generating the final answer from the accumulated memory.
- **ReMemR1** (Shi et al., 2026a) augments recurrent memory updates with retrieval from historical memories, allowing the agent to revisit previously processed information. Its training combines final-answer rewards with intermediate rewards.
- **LycheeMemory** (Chen et al., 2026) compresses document chunks into KV-cache representations that a reasoner processes with an evolving working memory. We evaluate the publicly released 7B checkpoint without the Gate module, denoted as LycheeMemory-7B (w/o Gate).
- **InfMem** (Wang et al., 2026) combines selective memory compression with retrieval from the source document to refine its memory. We use its publicly released Qwen3-4B-based checkpoint.
- **MemReread** (Ji et al., 2026) augments streaming reading with question decomposition and additional reading passes when the current memory is insufficient, enabling the recovery of previously omitted evidence. We use its publicly released Qwen3-4B-based checkpoint.

We evaluate the released LycheeMemory, InfMem, and MemReread checkpoints without additional training. Since these models differ in backbone, model size, and training procedure, their results provide system-level comparisons rather than controlled comparisons of memory mechanisms.

B.6 Evaluation Settings

Following Yu et al. (2026a), we randomly sample 128 examples from each benchmark using a random seed of 919. We report normalized sub-EM scores for all benchmarks. Before scoring, we normalize both the extracted predictions and reference answers by lowercasing, removing articles and punctuation, and normalizing whitespace. A prediction is considered correct if it contains all elements of the reference answer. For answers consisting of multiple required parts, the score is the proportion of correctly predicted parts.

Inference variants. We evaluate two inference variants: GraftMem (w Exit) and GraftMem (w/o Exit). With early exit enabled, reading terminates when the memory agent outputs `<next>end</next>`, and the answer agent generates an answer from the terminal memory. Without early exit, the workflow ignores this termination signal and continues processing all context chunks before answering. Both variants retain selective AcIn updates and differ only in whether the termination signal is executed. Unless otherwise stated, subsequent experiments use GraftMem (w Exit).

Table 6: Extended long-context QA results on HotpotQA, 2WikiMultiHopQA, and SQuAD. Values are normalized sub-EM accuracy (%). Gray-shaded rows report reference baselines and are excluded from the comparative ranking. **Bold** and underlined values denote the best and second-best results, respectively.

(a) HotpotQA (In-Distribution)

Scale	Method	7K	14K	28K	56K	112K	224K	448K	896K
3B	Qwen2.5-Instruct-3B	53.91	50.00	38.28	35.94	23.44	21.88	12.50	8.59
	MemAgent-3B	62.50	58.59	53.91	55.47	53.13	56.25	52.34	46.88
	ReMemR1-3B	60.94	59.38	56.25	57.81	54.69	56.25	53.13	50.00
	GraftMem-3B (w/o Exit)	63.28	63.28	60.16	**60.94**	**61.72**	**60.16**	**58.59**	**55.47**
	GraftMem-3B (w Exit)	**65.63**	**64.84**	**60.94**	57.81	58.59	58.59	54.69	52.34
$>$ 3B	Qwen3-4B	70.31	67.97	60.16	42.97	31.25	28.13	20.31	17.19
	R1-Distill-Qwen-7B	36.72	19.53	10.16	1.56	0.78	2.34	1.56	0.00
	Qwen2.5-Instruct-7B-1M	60.94	60.16	60.94	54.69	51.56	50.78	28.91	11.72
	MemReread-4B	58.59	59.38	60.16	52.34	56.25	53.13	50.78	51.56
	InfMem-4B	70.31	68.75	71.09	69.53	67.97	67.19	67.97	68.75
	LycheeMemory-7B (w/o Gate)	70.31	70.31	65.63	67.19	65.63	68.75	67.19	65.63
	GraftMem-7B (w/o Exit)	68.75	69.53	65.63	**67.97**	**68.75**	**67.19**	65.63	**66.41**
	GraftMem-7B (w Exit)	**71.88**	**72.66**	**66.41**	65.63	67.19	66.41	**67.97**	64.84

(b) 2WikiMultiHopQA (Out-of-Distribution)

Scale	Method	7K	14K	28K	56K	112K	224K	448K	896K
3B	Qwen2.5-Instruct-3B	44.53	39.84	33.59	25.78	14.06	10.94	7.81	5.47
	MemAgent-3B	52.34	49.22	47.66	49.21	43.75	40.63	37.50	35.16
	ReMemR1-3B	**56.25**	51.56	52.34	48.44	45.31	42.19	40.63	37.50
	GraftMem-3B (w/o Exit)	54.69	50.78	**53.13**	**51.56**	**49.22**	**46.88**	41.41	**42.19**
	GraftMem-3B (w Exit)	55.47	**52.34**	50.78	50.78	47.66	44.53	**42.97**	39.06
$>$ 3B	Qwen3-4B	**67.97**	61.72	53.13	28.91	19.53	14.84	15.63	12.50
	R1-Distill-Qwen-7B	36.72	27.34	23.44	4.69	3.13	3.13	2.34	2.34
	Qwen2.5-Instruct-7B-1M	66.41	**62.50**	54.69	53.91	48.44	48.44	20.31	16.41
	MemReread-4B	58.59	59.38	61.72	54.69	58.59	55.47	55.47	53.13
	InfMem-4B	69.53	71.09	67.97	69.53	67.19	62.50	63.28	59.38
	LycheeMemory-7B (w/o Gate)	62.50	60.16	64.84	57.81	59.38	57.03	56.25	53.91
	GraftMem-7B (w/o Exit)	63.28	60.94	**65.63**	**63.28**	**58.59**	**57.81**	**54.69**	**53.13**
	GraftMem-7B (w Exit)	64.84	61.72	62.50	59.38	56.25	54.69	53.13	51.56

(c) SQuAD (Out-of-Distribution)

Scale	Method	7K	14K	28K	56K	112K	224K
3B	Qwen2.5-Instruct-3B	74.22	71.88	65.63	39.06	23.44	15.63
	MemAgent-3B	76.56	**77.34**	73.44	70.31	68.75	70.31
	ReMemR1-3B	78.91	76.56	74.22	72.66	71.09	67.97
	GraftMem-3B (w/o Exit)	78.13	75.00	75.00	**77.34**	**74.22**	**72.66**
	GraftMem-3B (w Exit)	**80.47**	76.56	**75.78**	73.44	71.88	71.88
$>$ 3B	Qwen3-4B	80.47	78.13	58.59	48.44	43.75	45.31
	R1-Distill-Qwen-7B	30.47	17.19	8.59	1.56	3.13	2.34
	Qwen2.5-Instruct-7B-1M	83.59	**82.03**	80.47	74.22	70.31	54.69
	MemReread-4B	72.66	67.19	68.75	66.41	62.50	57.03
	InfMem-4B	82.81	82.03	80.47	82.03	77.34	78.13
	LycheeMemory-7B (w/o Gate)	81.25	82.81	79.69	81.25	79.69	78.91
	GraftMem-7B (w/o Exit)	83.59	79.69	**82.81**	**82.81**	**79.69**	**78.13**
	GraftMem-7B (w Exit)	**84.38**	80.47	81.25	80.47	78.91	77.34

C Additional Experiments

C.1 Main Results

Table 6 extends the main evaluation with additional comparisons at the 3B scale and above. At 3B, we report results for Qwen2.5-Instruct, MemAgent, ReMemR1, and GraftMem. The $>$ 3B group includes Qwen3-4B, R1-Distill-Qwen-7B, Qwen2.5-Instruct-7B-1M, MemReread-4B, InfMem-4B,

LycheeMemory-7B (w/o Gate), and GraftMem. We evaluate the publicly released MemReread, InfMem, and LycheeMemory checkpoints without additional training.

At the 3B scale, both GraftMem variants achieve strong results across the three datasets. On HotpotQA, GraftMem-3B (w/o Exit) achieves maximum absolute accuracy gains of 8.59% and 7.03% over MemAgent-3B and ReMemR1-3B, respectively; the corresponding gains for GraftMem-3B (w Exit) are 7.03% and 5.46%. On 2WikiMultiHopQA, the gains are 7.03% and 4.69% for the w/o Exit variant, and 5.47% and 2.35% for the w Exit variant. On SQuAD, they are 7.03% and 4.69% for w/o Exit, and 3.91% and 3.91% for w Exit. In the $>$ 3B group, both GraftMem-7B variants achieve higher accuracy than Qwen2.5-Instruct-7B-1M at most evaluated context lengths. The maximum absolute gains for w/o Exit and w Exit are 54.69% and 53.12% on HotpotQA, 36.72% and 35.15% on 2WikiMultiHopQA, and 23.44% and 22.65% on SQuAD, respectively.

C.2 Distant Evidence on 2WikiMultiHopQA

We extend the distant-evidence evaluation to the out-of-distribution 2WikiMultiHopQA dataset. Following the HotpotQA setting, we insert distractor documents between the two supporting documents so that the gap between them exceeds half the total context length.

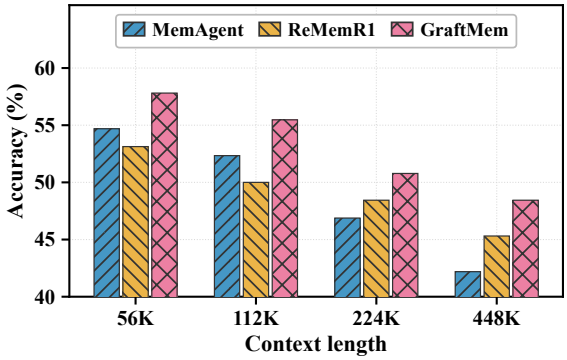


Figure 8: Accuracy on 2WikiMultiHopQA.

As shown in Figure 8, GraftMem achieves the highest accuracy at all evaluated context lengths, as it does on HotpotQA in the main text. At the two longest lengths, ReMemR1 overtakes MemAgent, yet GraftMem remains ahead of both. Together, these results suggest that selective AcIn updates help mitigate information loss when supporting documents are widely separated.

C.3 Memory-Budget Overflow Analysis

To examine GraftMem when its evidence memory exceeds the token budget, we construct a controlled overflow setting on HotpotQA. We preserve the original questions and answers, repeat one supporting document with a different excerpt from the original context on each occurrence, and place the other supporting document later in the context. We use a 512-token memory budget and count a case as overflow-triggered only when a consolidation call actually occurs. Following the main-text analysis in Figure 5, we measure memory length after each processed chunk at 896K for MemAgent, ReMemR1, and GraftMem (w/o Exit), as shown in Figure 9. We further evaluate answer accuracy from 56K to 448K for MemAgent, ReMemR1, and GraftMem, as summarized in Table 7.

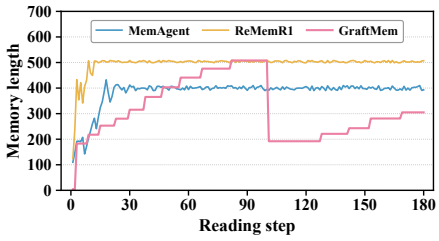


Figure 9: Memory size dynamics on the HotpotQA overflow stress set at 896K.

Table 7: Accuracy (%) on HotpotQA overflow stress sets.

Method	56K	112K	224K	448K
MemAgent-7B	68.75	66.41	63.28	62.50
ReMemR1-7B	64.84	63.28	60.94	61.72
GraftMem-7B (w/o Exit)	**70.31**	**69.53**	**67.97**	**66.41**
GraftMem-7B (w Exit)	65.63	67.19	64.84	64.06

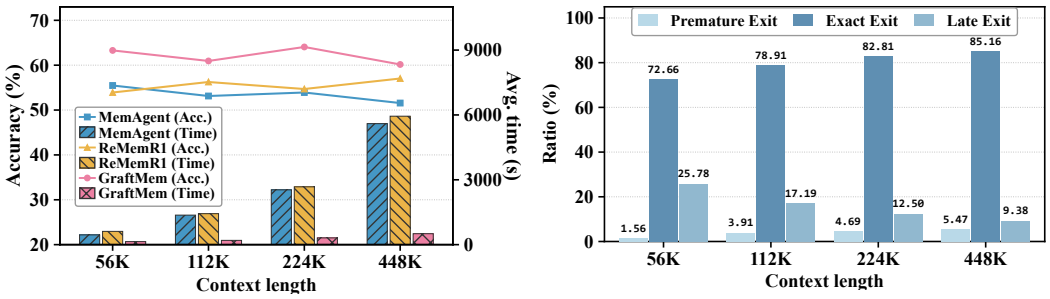
Figure 9 illustrates the intended behavior of GraftMem at the memory-budget boundary: after its memory grows toward the budget, consolidation reduces its length, and reading continues with subsequent memory updates. Table 7 shows that GraftMem (w/o Exit) achieves the highest accuracy at every evaluated length, exceeding MemAgent by 1.56–3.91 percentage points. GraftMem (w Exit),

however, is less accurate than its w/o Exit counterpart. These results suggest that the w/o Exit variant handles the constructed overflow setting more robustly, whereas enabling early exit can reduce accuracy in this setting. The stress set deliberately repeats supporting evidence and delays another supporting document to trigger a boundary condition absent from our standard evaluations.

C.4 Performance under the Top20 Setting

We extend the Top20 evaluation in Section 3.3.2 to the out-of-distribution 2WikiMultiHopQA dataset, again placing the last supporting document within the first 20% of the context.

As shown in Figure 10(a), GraftMem achieves the highest accuracy at all four evaluated context lengths. At 448K, it achieves inference speedups of $11.1\times$ and $11.7\times$ over MemAgent and ReMemR1, respectively. Figure 10(b) shows that the proportion of exact exits increases from 72.66% at 56K to 85.16% at 448K. Consistent with the HotpotQA results, GraftMem usually stops at the last supporting-document step while reducing inference time on this out-of-distribution task.



(a) Performance under the Top20 setting.

(b) The ratios of premature, exact, and late exits.

Figure 10: Early-exit behavior under the Top20 setting on 2WikiMultiHopQA.

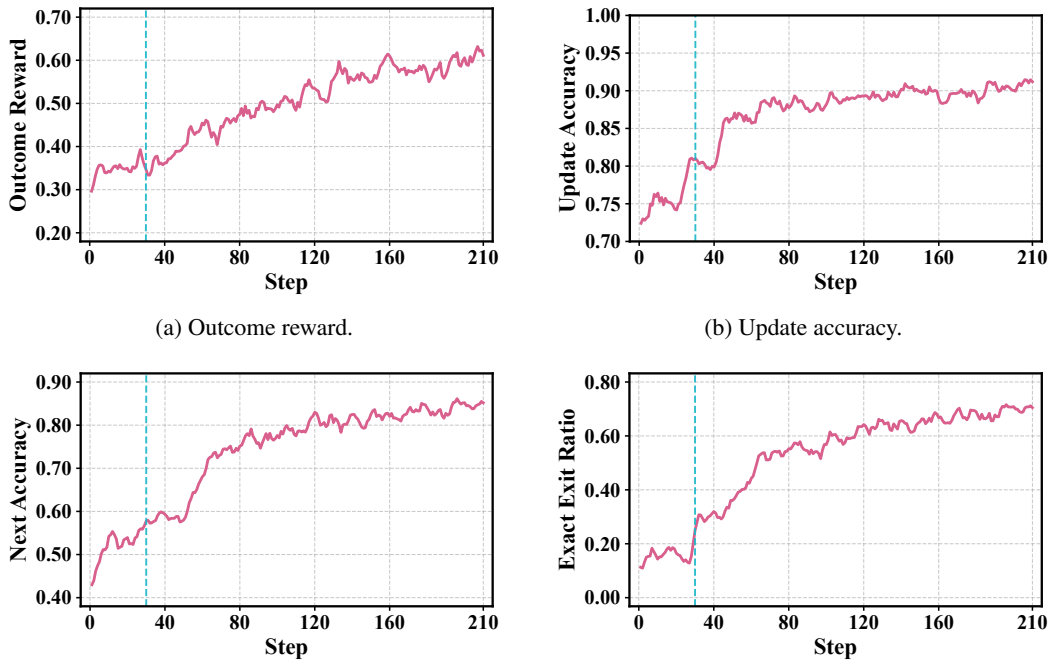
C.5 Training Dynamics

Figure 11 presents the training dynamics of GraftMem with Qwen2.5-7B-Instruct. We track outcome reward, Update accuracy, Next accuracy, and the exact-exit ratio. Update and Next accuracies measure agreement with annotation-derived control targets, while the exact-exit ratio measures the fraction of trajectories terminating at the last supporting-evidence chunk t^*.

Learning selective memory updates. Update accuracy increases from approximately 0.72 to 0.86 within the first 50 training steps, followed by a slower increase to around 0.91. Under the AcIn update mechanism, this decision determines whether the model performs a memory update or directly reuses the memory from the previous turn without updating it. The improvement in accuracy therefore indicates that the selective-update mechanism has been effectively learned and continues to improve steadily.

Learning when to stop reading. Next accuracy measures the accuracy of termination decisions: the model is expected to output `continue` before t^* and `end` at or after t^*. Next accuracy increases from approximately 0.43 to 0.85, with a pronounced rise during the middle phase of training, around steps 50 to 65, followed by slower improvement. Moreover, the exact-exit ratio exhibits a similar trend to Next accuracy. Because early exit was disabled during the first 30 steps, the exact-exit ratio during this period should reflect only trajectories whose final supporting evidence appears in the last chunk, resulting in a value of approximately 15%. After early exit is enabled, the exact-exit ratio increases rapidly from roughly 0.30 to 0.60 between training steps 30 and 60, and then continues to rise more gradually to 0.70.

Outcome reward. Despite fluctuations throughout training, particularly around the two key points at steps 20 and 30, outcome reward increases from approximately 0.30 to 0.65. Its improvement continues after Update accuracy begins to level off, alongside further gains in Next accuracy and the exact-exit ratio. This pattern is consistent with the complementary roles of the training signals: the



(a) Outcome reward.

(b) Update accuracy.

(c) Next accuracy.

(d) Exact-exit ratio.

Figure 11: **Training dynamics of GraftMem at the 7B scale.** The cyan dashed line marks the activation of early exit at step 30.

model gradually learns when to update memory and when to stop reading, while outcome reward, as a trajectory-level reward, captures the substantive gains resulting from improvements in these capabilities.

C.6 Ablation Study

C.6.1 Impact of α

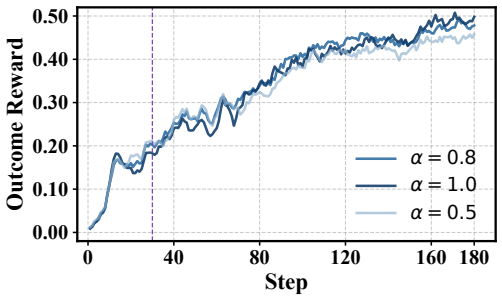
We compare $\alpha \in \{1.0, 0.8, 0.5\}$ to examine how the balance between trajectory-level and state-level advantages affects training. Figures 12 and 13 present the corresponding training dynamics. The purple dashed line marks the activation of early exit at training step 30.

Format compliance and outcome reward. All three settings rapidly acquire the required output format, with format reward approaching 1.0 by step 30. Outcome reward continues to improve over a longer training interval, with clearer differences emerging later in training, when $\alpha = 0.8$ generally achieves higher outcome reward.

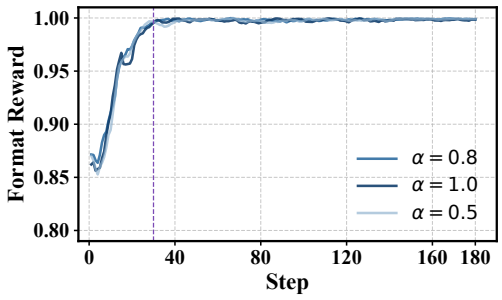
Learning Update and Next decisions. Update and Next accuracies improve throughout training under all three settings. The settings incorporating state-level rewards, $\alpha = 0.8$ and $\alpha = 0.5$, generally achieve higher accuracies than $\alpha = 1.0$, with $\alpha = 0.5$ often improving faster and reaching slightly higher values. These trends suggest that state-level feedback helps the model learn when to update memory and stop reading. However, $\alpha = 0.5$ does not achieve the highest outcome reward, indicating that alignment with local control targets alone does not determine final-answer quality.

The correct exit behavior is gradually learned. We evaluate termination by the exact-, premature-, and late-exit ratios ($t_{\text{exit}} = t^*$, $t_{\text{exit}} < t^*$, and $t_{\text{exit}} > t^*$), together with the mean absolute deviation $\mathbb{E}[|t_{\text{exit}} - t^*|]$ in chunks. For the first 30 steps, early exit is disabled: trajectories read the full context and exit exactly at t^* only when the final supporting evidence appears in the last chunk. After early exit is enabled, all three settings progressively increase their exact-exit ratios and reduce their absolute exit deviations. Premature-exit ratios initially rise and then generally decline, while late-exit ratios decrease.

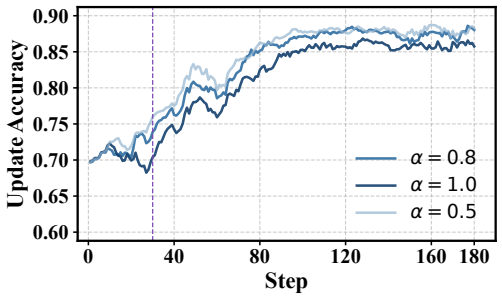
The settings differ in termination behavior. $\alpha = 1.0$ yields more exact exits and smaller timing deviations, but also more premature exits; $\alpha = 0.5$ reduces premature exits at the cost of more delayed ones. Its higher Next accuracy does not necessarily translate into more exits at t^*, as local decision accuracy and trajectory-level timing differ. $\alpha = 0.8$ balances these tendencies and achieves higher outcome reward later in training.



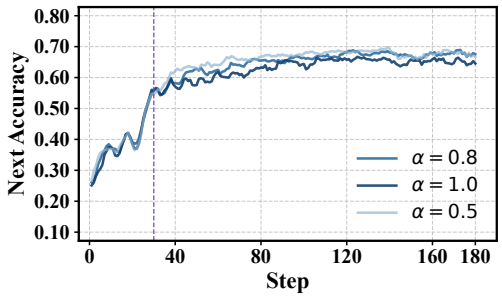
(a) Outcome reward.



(b) Format reward.

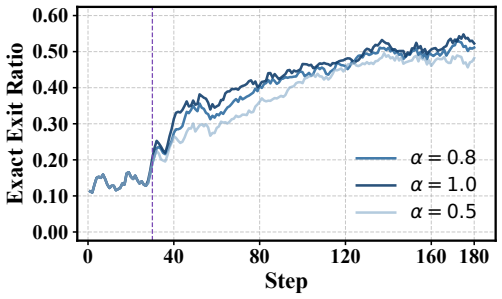


(c) Update accuracy.

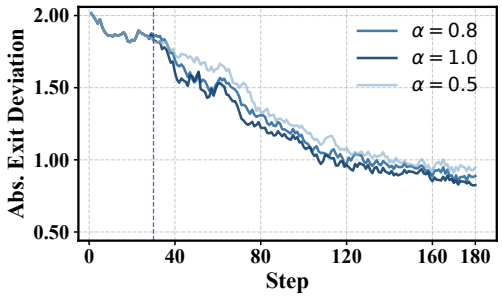


(d) Next accuracy.

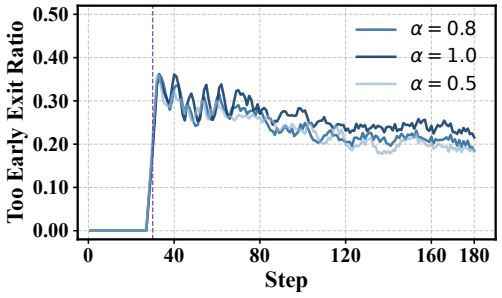
Figure 12: Training dynamics under different α values (part 1).



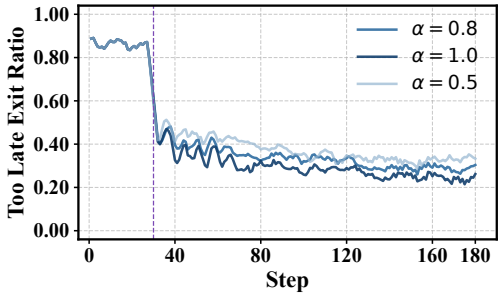
(a) Exact-exit ratio.



(b) Mean absolute exit deviation.

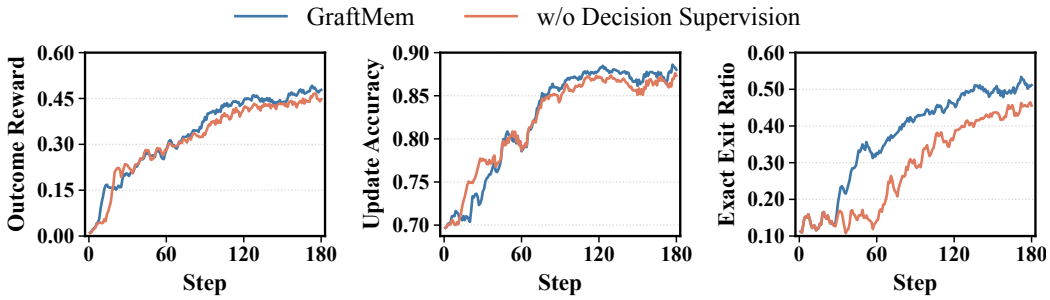


(c) Premature-exit ratio.



(d) Late-exit ratio.

Figure 13: Training dynamics under different α values (part 2).



(a) Outcome reward. (b) Update accuracy. (c) Exact-exit ratio.

Figure 14: Training dynamics with and without decision supervision.

C.6.2 Effect of Decision Supervision

We compare GraftMem with a variant trained without auxiliary decision supervision. Figure 14 shows their outcome reward, Update accuracy, and exact-exit ratio during training.

Decision supervision improves exit timing. The clearest difference appears in the exact-exit ratio. GraftMem begins to improve around training step 30 and reaches approximately 0.35 by step 50. In comparison, the variant without decision supervision fluctuates around 0.15 until sustained improvement begins near step 60. This gap persists through the end of training. These observations suggest that direct supervision of control decisions helps the model learn correct termination behavior earlier in training.

Effects on Update accuracy and outcome reward. Both settings show substantial improvements in Update accuracy and outcome reward. During part of early training, however, the variant without decision supervision achieves higher Update accuracy. This may reflect the additional optimization demands of jointly learning from rewards and auxiliary supervision, requiring more training steps to coordinate the objectives. Later in training, GraftMem generally achieves higher Update accuracy and outcome reward.

These results suggest that decision supervision complements reward-based feedback, supporting the learning of termination and memory-update decisions and contributing to improved outcome reward.

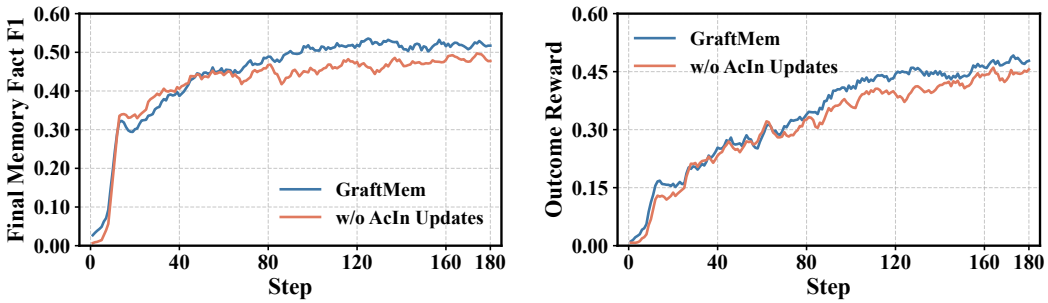
C.6.3 Effect of AcIn Updates

We compare GraftMem with a variant that replaces AcIn updates with overwrite-based memory updates. Figure 15 presents their final-memory fact F1 and outcome reward during training. Final-memory fact F1 measures the overlap between the final memory and annotated supporting facts.

AcIn updates improve final-memory evidence quality. Both settings show rapid initial improvements in final-memory fact F1. The variant w/o AcIn updates achieves slightly higher values during part of early training, but GraftMem subsequently overtakes it and maintains a clearer advantage in the later stages.

This pattern is consistent with the design of AcIn: separating explicit evidence acquisition from incremental memory integration allows newly acquired information to be incorporated while preserving earlier evidence. In contrast, overwrite-based updates require the model to regenerate any previously stored information it needs to retain, creating opportunities for supporting details to be omitted.

Outcome reward. Outcome reward increases under both settings, with the curves remaining relatively close over much of early and middle training. A clearer gap emerges later, when GraftMem generally achieves higher reward. This separation accompanies its growing advantage in final-memory fact F1, suggesting that improved evidence acquisition and retention provide a more useful basis for answering.



(a) Final-memory fact F1.

(b) Outcome reward.

Figure 15: Training dynamics with and without AcIn updates.

Together, these observations support the role of AcIn in retaining supporting facts in the final memory. Separating evidence acquisition from memory integration, rather than repeatedly rewriting the entire memory, helps retain supporting evidence and contributes to improved final-answer correctness.

D Case Study

We present qualitative analyses of GraftMem. We first compare its memory updates and predictions with those of MemAgent and ReMemR1 on the same questions to demonstrate the advantages of GraftMem. We then examine unsuccessful GraftMem trajectories to identify distinct failure modes. For readability, we display selected input passages and model responses, with omissions explicitly marked.

D.1 Comparative Cases

D.1.1 Information Loss during Memory Overwriting

Question: Which of the seven Disney characters who had been given a series of their own, starring in cartoons who opened with their own logo is a fictional character who first appeared in Disney's "Pinocchio"?
Supporting Evidence:
(Turn 8, Document 261: Humphrey the Bear) [...] When the shorts division closed, Humphrey was the last of only seven Disney characters who had been given a series of their own, starring in cartoons who opened with their own logo (the six others were Mickey Mouse, Donald Duck, Goofy, Pluto, Chip 'n' Dale (counting as one), and Figaro).
(Turn 25, Document 872: Figaro (Disney)) Figaro is a fictional character who first appeared in Disney's "Pinocchio".
Ground Truth: Figaro
Predictions:
MemAgent: Jiminy Cricket
ReMemR1: Jiminy Cricket
GraftMem: Figaro

MemAgent — Turn 25

Updated memory:
The problem asks for a Disney character who first appeared in Disney's "Pinocchio" and has had a series of their own cartoons starring them with their own logo. The section provides information about a fictional character named Figaro, who first appeared in Disney's "Pinocchio". This character matches the description in the problem.
Therefore, the relevant information to update the memory is:
- Figaro is a fictional character who first appeared in Disney's "Pinocchio".
- The specific Disney character in question is Figaro, who first appeared in Disney's "Pinocchio" and has had a series of his own cartoons starring him with their own logo.

MemAgent — Turn 27

Updated memory:
The problem asks for a Disney character who first appeared in Disney's "Pinocchio" and has had a series of their own cartoons starring them with their own logo. The section provides information about several characters and topics, but the relevant information is found in the description of Sylvester the Cat.
[...]
Therefore, the relevant information to update the memory is:
- Sylvester the Cat is a fictional character who first appeared in Disney's "Pinocchio".

MemAgent — Turn 44

Updated memory:
The problem asks for a Disney character who first appeared in Disney's "Pinocchio" and has had a series of their own cartoons starring them with their own logo. [...] Since his debut in "Pinocchio," he has become a recurring iconic Disney character and has starred in his own series of cartoons with their own logo.[...]

MemAgent — Fianl Turn (after Turn 45)

Final Memory: The problem asks for a Disney character who first appeared in Disney's "Pinocchio" and has had a series of their own cartoons starring them with their own logo. [...] Jiminy Cricket was appointed by the Blue Fairy to serve as Pinocchio's official conscience and has made numerous other appearances. Since his debut in "Pinocchio," he has become a recurring iconic Disney character and has starred in his own series of cartoons with their own logo. [...]

Answer: Therefore, the answer is Jiminy Cricket.

ReMemR1 – Turn 43

Recalled Memory: Humphrey the Bear is a cartoon character created by the Walt Disney studio in 1950. He first appeared in the Goofy cartoon "Hold That Pose" and later appeared in four Donald Duck cartoons. Humphrey was the last of only seven Disney characters who had their own series of cartoons that opened with their own logo, and he starred in two films: "Hooked Bear" and "In the Bag". Humphrey first appeared in Disney's "Pinocchio" and starred in his own series of cartoons.

[...]
<update>Figaro is a fictional character who first appeared in Disney's "Pinocchio". He is one of the seven Disney characters who had their own series of cartoons that opened with their own logo, and he starred in his own series of cartoons.</update>
<recall>who's the president of the United States?</recall>

ReMemR1 – Turn 44

Recalled Memory: Humphrey the Bear is a cartoon character created by the Walt Disney studio in 1950. He first appeared in the Goofy cartoon "Hold That Pose" and later appeared in four Donald Duck cartoons. Humphrey was the last of only seven Disney characters who had their own series of cartoons that opened with their own logo, and he starred in two films: "Hooked Bear" and "In the Bag". Humphrey first appeared in Disney's "Pinocchio" and starred in his own series of cartoons.

[...]
<update>Jiminy Cricket is the Walt Disney version of The Talking Cricket, a fictional character created by Carlo Collodi for his children's book "The Adventures of Pinocchio". He was transformed into a comical and wise partner who accompanies Pinocchio on his adventures and has made numerous other appearances. Jiminy Cricket first appeared in Disney's "Pinocchio" and starred in his own series of cartoons.</update>
<recall>who's the president of the United States?</recall>

ReMemR1 – Final Turn (after Turn 45)

Final Memory:
Jiminy Cricket is the Walt Disney version of The Talking Cricket, a fictional character created by Carlo Collodi for his children's book "The Adventures of Pinocchio". He was transformed into a comical and wise partner who accompanies Pinocchio on his adventures and has made numerous other appearances. Jiminy Cricket first appeared in Disney's "Pinocchio" and starred in his own series of cartoons.

Answer: \boxed{Jiminy Cricket}

GraftMem — Turn 8

```
<update>yes</update>
<evidence>
Humphrey the Bear is a cartoon character created by the Walt Disney studio in
1950. He first appeared in the Goofy cartoon "Hold That Pose", in which Goofy
tried to take his picture. After that he appeared in four classic Donald Duck
cartoons: "Rugged Bear", "Grin and Bear It", "Bearly Asleep", and "Beezy Bear".
Disney gave him his own series in 1955, but only two films resulted ("Hooked
Bear" and "In the Bag") before Disney discontinued making theatrical short
subjects. When the shorts division closed, Humphrey was the last of only seven
Disney characters who had been given a series of their own, starring in cartoons
who opened with their own logo (the six others were Mickey Mouse, Donald Duck,
Goofy, Pluto, Chip 'n' Dale (counting as one), and Figaro).
</evidence>
<next>continue</next>
```

GraftMem — Turn 25

```
<update>yes</update>
<evidence>
Figaro is a fictional character who first appeared in Disney's "Pinocchio".
</evidence>
<next>end</next>
```

GraftMem — Final Answer (after Turn 25)

Final memory:

```
Humphrey the Bear is a cartoon character created by the Walt Disney studio in
1950. He first appeared in the Goofy cartoon "Hold That Pose", in which Goofy
tried to take his picture. After that he appeared in four classic Donald Duck
cartoons: "Rugged Bear", "Grin and Bear It", "Bearly Asleep", and "Beezy Bear".
Disney gave him his own series in 1955, but only two films resulted ("Hooked
Bear" and "In the Bag") before Disney discontinued making theatrical short
subjects. When the shorts division closed, Humphrey was the last of only seven
Disney characters who had been given a series of their own, starring in cartoons
who opened with their own logo (the six others were Mickey Mouse, Donald Duck,
Goofy, Pluto, Chip 'n' Dale (counting as one), and Figaro).
Figaro is a fictional character who first appeared in Disney's "Pinocchio".
```

Answer: \boxed{Figaro}

This case illustrates how correctly acquired evidence can be lost during subsequent memory updates. MemAgent correctly identified Figaro at turn 25, but replaced it with Sylvester the Cat at turn 27 and later shifted to Jiminy Cricket. ReMemR1 retained the correct Figaro evidence through turn 43, yet replaced it with Jiminy Cricket at turn 44. Both baselines incorrectly attributed the required cartoon-series property to the replacement character. The recalled memory shown for ReMemR1 contained an erroneous attribution to Humphrey rather than the identifying evidence about Figaro, and did not prevent the final incorrect answer.

In contrast, GraftMem preserved the candidate list acquired at turn 8 and appended the identifying fact about Figaro at turn 25. Its final memory therefore contained both conditions needed to answer the question: membership in the specified group of Disney characters and a first appearance in *Pinocchio*. This trajectory illustrates how separating explicit evidence acquisition from incremental memory integration preserves earlier supporting information while incorporating new evidence. GraftMem then terminated reading and returned the correct answer, avoiding the remaining 20 reading turns.

D.1.2 CRITICAL EVIDENCE OMISSION DURING MEMORY ACQUISITION

Question: What is a common feature of both the Iron Rattler's inversion and a barrel roll?
Supporting Evidence:
(Turn 13, Document 447: Barrel roll) A barrel roll is an aerial maneuver in which an airplane makes a complete rotation on both its longitudinal and lateral axes, causing it to follow a helical path, approximately maintaining its original direction. It is sometimes described as a "combination of a loop and a roll."
(Turn 22, Document 751: Iron Rattler) [...] The inversion is a zero-g-roll, which is when the train goes through a combination of a loop and a roll.
Ground Truth: combination of a loop and a roll
Predictions:
MemAgent: that both the Iron Rattler's inversion and a barrel roll involve an inverted or upside-down position during the maneuver
ReMemR1: inversion and rotation
GraftMem: combination of a loop and a roll

MemAgent — Turn 13

Updated memory:
A common feature of both the Iron Rattler's inversion and a barrel roll is that both are likely aerial maneuvers. The section provides a detailed description of a barrel roll, which is an aerial maneuver in which an airplane makes a complete rotation on both its longitudinal and lateral axes, causing it to follow a helical path, approximately maintaining its original direction. [...]
Although the section does not directly mention the Iron Rattler's inversion, it reiterates that the Iron Rattlers inversion is a maneuver in roller coasters where the train inverts in the air. Both maneuvers involve an inverted or upside-down position during the maneuver.[...]

MemAgent — Turn 22

Updated memory:
A common feature of both the Iron Rattler's inversion and a barrel roll is that both are likely aerial maneuvers. The provided section does not contain any information related to the common feature between the Iron Rattler's inversion and a barrel roll. However, it reiterates that the Iron Rattler's inversion is a [...] The provided section does not contain any information related to the common feature between the Iron Rattler's inversion and a barrel roll.

MemAgent — Final Turn (after Turn 45)

Final Memory: A common feature of both the Iron Rattler's inversion and a barrel roll is that both are likely aerial maneuvers. The provided section does not contain any additional information directly related to the common feature between the Iron Rattler's inversion and a barrel roll. However, it reiterates that the Iron Rattler's inversion is a maneuver in roller coasters where the train inverts in the air, and a barrel roll is an aerial maneuver in which an airplane makes a complete rotation on both its longitudinal and lateral axes, causing it to follow a helical path, approximately maintaining its original direction. Both maneuvers involve an inverted or upside-down position during the maneuver.[...]
Answer: Therefore, the answer is that both the Iron Rattler's inversion and a barrel roll involve an inverted or upside-down position during the maneuver.

ReMemR1 – Turn 13

Recalled Memory : Adrian "Wildman" Cenni is the first person to successfully land a true 360 degree barrel roll in a vehicle. The section does not contain information about the Iron Rattler's inversion, but mentions a barrel roll performed by Adrian "Wildman" Cenni. A barrel roll is a maneuver where an aircraft rolls inverted through a horizontal circle.

[...]
<update>A barrel roll is an aerial maneuver in which an airplane makes a complete rotation on both its longitudinal and lateral axes, causing it to follow a helical path, approximately maintaining its original direction. The provided section does not contain information about the Iron Rattler's inversion, but mentions a barrel roll.</update>
<recall>What is the Iron Rattler's inversion and how does it compare to a barrel roll?</recall>

ReMemR1 – Turn 22

Recalled Memory : Adrian "Wildman" Cenni is the first person to successfully land a true 360 degree barrel roll in a vehicle. The section does not contain information about the Iron Rattler's inversion, but mentions a barrel roll performed by Adrian "Wildman" Cenni.

[...]
<update>The Iron Rattler's inversion and a barrel roll both involve an inversion and rotation. The Iron Rattler is a roller coaster with a hybrid wood and steel structure that features a zero-g-roll inversion, while a barrel roll is an aerial maneuver in which an airplane makes a complete rotation on both its longitudinal and lateral axes, causing it to follow a helical path.</update>
<recall>what is the definition of a barrel roll in an airplane?</recall>

ReMemR1 – Final Turn (after Turn 45)

Final Memory:
[...]
The Iron Rattler is a roller coaster with a hybrid wood and steel structure th
at features a zero-g-roll inversion, while a barrel roll is an aerial maneuver
 in which an aircraft makes a complete rotation on both its longitudinal and l
ateral axes, causing it to follow a helical path.
[...]
Answer: \boxed{inversion and rotation}

GraftMem – Turn 13

<update>yes</update>
<evidence>
A barrel roll is an aerial maneuver in which an airplane makes a complete rotation on both its longitudinal and lateral axes, causing it to follow a helical path, approximately maintaining its original direction. It is sometimes described as a "combination of a loop and a roll." The g-force is kept positive (but not constant) on the object throughout the maneuver, commonly between 2–3 g, and no less than 0.5 g. The barrel roll is commonly confused with an aileron roll.
</evidence>
<next>continue</next>

GraftMem — Turn 22

```
<update>yes</update>
<evidence>
Iron Rattler is a steel roller coaster located at Six Flags Fiesta Texas in San
Antonio, Texas. The ride opened in 2013, replacing The Rattler, a wooden roller
coaster, by putting a new steel track on top of The Rattler's wooden support
structure.[...] The resulting modifications increased the speed of the trains fr
om 65 to . It is the first hybrid wood and steel roller coaster to feature an
inversion, in which riders are turned upside-down and then back upright. The
inversion is a zero-g-roll, which is when the train goes through a combination
of a loop and a roll.
</evidence>
<next>end</next>
```

GraftMem — Final Answer (after Turn 22)

```
Fianl memory:
A barrel roll is an aerial maneuver in which an airplane makes a complete
rotation on both its longitudinal and lateral axes, causing it to follow a
helical path, approximately maintaining its original direction. It is sometimes
described as a "combination of a loop and a roll." [...]
Iron Rattler is a steel roller coaster located at Six Flags Fiesta Texas in San
Antonio, Texas. [...]It is the first hybrid wood and steel roller coaster to fea
ture an inversion, in which riders are turned upside-down and then back upright.
The inversion is a zero-g-roll, which is when the train goes through a combinati
on of a loop and a roll.
Answer: \boxed{combination of a loop and a roll}
```

This case illustrates the omission of critical evidence when forming memory. At turn 13, the input explicitly described a barrel roll as a "combination of a loop and a roll." Both baselines retained general descriptions of the maneuver but omitted this specific property. When the matching description of the Iron Rattler's inversion appeared at turn 22, MemAgent incorrectly stated that the section contained no information about the shared feature, while ReMemR1 summarized the relation as "inversion and rotation." The displayed recalled memories did not supply the missing detail. Their final answers consequently remained broader than the reference answer.

GraftMem explicitly acquired the loop-and-roll description at turn 13 and appended the corresponding evidence about the Iron Rattler at turn 22. By retaining the specific property stated in both passages, its final memory supported a direct comparison and the correct answer. Unlike the preceding case, where previously acquired evidence was subsequently replaced, this example highlights the importance of acquiring answer-relevant details before they are omitted from memory. GraftMem preserved these details and terminated reading once the complementary evidence had been integrated.

D.2 Failure Cases

D.2.1 Failure to Acquire Critical Evidence

In this case, GraftMem failed to acquire an explicitly stated fact despite encountering it twice. The passages at turns 8 and 36 identified Jem Finer as a banjo player, but the model declined to update memory on both occasions. Although it stored a biographical passage at turn 15, that passage did not specify his instrument. After reading all 45 chunks, GraftMem returned "whistle" from a memory that lacked the required fact. This case illustrates a limitation of incremental integration: preserving acquired information does not ensure that critical evidence is acquired in the first place.

Question: What intrusment did Jem Finer play in the band The Pogues?
Supporting Evidence:
(Turn 8, Document 255: Misty Morning, Albert Bridge) "Misty Morning, Albert Bridge" is a 1989 single by the British-Irish folk rock band The Pogues. It was composed by banjo player Jem Finer and featured on the band's fourth album, "Peace and Love".
(Turn 36, Document 1210: Once Upon a Time (The Pogues song)) "Once Upon a Time" is a song by The Pogues released as a single in 1993 from their sixth album, "Waiting for Herb". The song was the band's last ever single to chart in the UK, making number 66, before the band broke up in 1996 following the release of their seventh and final album, "Pogue Mahone". The song was composed by Banjo player Jem Finer.
Ground Truth: banjo player
Prediction: whistle

GraftMem — Turn 8

```
<update>no</update>
<evidence>None.</evidence>
<next>continue</next>
```

GraftMem — Turn 15

```
<update>yes</update>
<evidence>
Jeremy Max "Jem" Finer (born 25 July 1955) is an English musician, artist and composer. He was one of the founding members of The Pogues.
</evidence>
<next>continue</next>
```

GraftMem — Turn 36

```
<update>no</update>
<evidence>None.</evidence>
<next>continue</next>
```

GraftMėm — Final Answer (after Turn 45)

Final memory:
Jeremy Max "Jem" Finer (born 25 July 1955) is an English musician, artist and composer. He was one of the founding members of The Pogues.
Answer: \boxed{ whistle }

D.2.2 Premature Exit with Incomplete Evidence

GraftMem identified Rob Marciano as the meteorologist who replaced Ginger Zee at turn 1, but this established only the intermediate relation required by the question. At turn 5, it terminated reading without acquiring evidence about who occasionally fills in for Marciano and returned his name instead. The passage identifying Kait Parker as his substitute appeared in chunk 13 and was therefore never read. This trajectory illustrates a premature sufficiency judgment.

Question: Who fills in occasionally for the meteorologist who replaced Ginger Zee?
Supporting Evidence:
(Turn 1 – Read, Document 15: Rob Marciano) Robert Mark "Rob" Marciano (born June 25, 1968) is an American journalist and meteorologist who is currently employed by ABC News. Marciano is ABC's, and currently provides forecasts for the weekend editions of "Good Morning America", a position Ginger Zee vacated when she was chosen to succeed Champion on the daily editions of "GMA".
(Chunk 13 – Unread, Document 414: Kait Parker) Kait Parker is an atmospheric scientist who currently works for weather.com and The Weather Channel app. She can also occasionally be seen on "Good Morning America" where she fills-in for meteorologist Rob Marciano on weekends.Prior to her role with weather.com, she was an on-camera meteorologist and host of America's Morning Headquarters Weekend Edition alongside Reynolds Wolf. She left The Weather Channel television network in early 2016.
Ground Truth: Kait Parker
Prediction: Rob Marciano

GraftMem – Turn 1

<update>yes</update>
<evidence>
Rob Marciano (born June 25, 1968) is an American journalist and meteorologist who is currently employed by ABC News. Marciano is ABC's, and currently provides forecasts for the weekend editions of "Good Morning America", a position Ginger Zee vacated when she was chosen to succeed Champion on the daily editions of "GMA".
</evidence>
<next>continue</next>

GraftMem – Turn 5

<update>yes</update>
<evidence>
Ginger Renee Colonomos (née Zuidgeest; born January 13, 1981), known by her pseudonym Ginger Zee, is an American television personality. She is the chief meteorologist for ABC News. Previously she was their weekend meteorologist.
Bob Ryan is a retired meteorologist who most recently forecasted for WJLA, the ABC affiliate in Washington, D.C.. From 1980–2010, he served as the chief meteorologist at Washington NBC affiliate WRC-TV. Before serving as the station's chief meteorologist for 30 years, he was previously the "Today Show's" first on-air meteorologist, which was also the first network television meteorologist position. When Willard Scott replaced Ryan on "Today", he and Scott effectively exchanged jobs, with Ryan taking over the meteorologist position vacated by Scott on WRC.
</evidence>
<next>end</next>

GraftMem – Final Answer (after Turn 5)

Final memory:
Rob Marciano (born June 25, 1968) is an American journalist and meteorologist who is currently employed by ABC News. [...]
Ginger Renee Colonomos (née Zuidgeest; born January 13, 1981), known by her pseudonym Ginger Zee, is an American television personality. She is the chief meteorologist for ABC News.[...]
Answer: \boxed{Rob Marciano}