EVOMEMORY: LEARNING FUTURE UTILITY FOR BOUNDED MEMORY REPLACEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) increasingly rely on long-term memory to support interactions that extend beyond a fixed context window. Yet memory capacity is inherently limited: as new experiences continually arrive, a system must decide not only what to store, but also what to retain, replace, or discard. Existing memory systems often rely on local signals such as recency, similarity, or standalone importance, which do not directly capture how a memory update will affect future task performance. We introduce **EvoMemory**, a framework that learns bounded memory replacement based on future utility. We formulate replacement as a sequential decision problem in which each incoming memory induces alternative successor memory states. During training, EvoMemory evaluates these counterfactual states on future examples to construct utility supervision and learns a contextual value model to score their downstream utility. At deployment time, the learned policy updates memory using only the current bank and incoming memory, without access to future examples. We evaluate EvoMemory on two complementary benchmarks, JRE-L and WikiLarge, against heuristic and random memory-management strategies. On JRE-L, EvoMemory achieves a held-out utility of 0.4719, compared with 0.4599 for five-seed random replacement and 0.4590 for the strongest deterministic heuristic. On WikiLarge, EvoMemory achieves a held-out utility of 0.8152, compared with 0.7887 for retaining the initial memory bank and 0.7726 for five-seed random replacement. Across both benchmarks, paired tests on held-out examples confirm significant improvements over all evaluated baselines after multiple-comparison correction. These results support future-utility learning as an effective principle for bounded memory replacement, where useful updates depend on their consequences for the resulting memory state.

1 INTRODUCTION

Large language models (LLMs) increasingly operate in settings that extend beyond isolated interactions, including long-running assistants, adaptive agents, and retrieval- or memory-augmented systems (Lewis et al., 2020; Borgeaud et al., 2022; Park et al., 2023; Packer et al., 2023; Zhong et al., 2024; Zhang et al., 2025). Persistent external memory provides a natural mechanism for retaining useful information across interactions without continually modifying model parameters. However, as experience accumulates, storing all past information becomes impractical. Memory systems must therefore decide not only what to retrieve, but also what to retain, replace, or discard under a limited memory budget.

Existing memory systems often rely on local signals such as recency, similarity, salience, or independently estimated importance (Zhong et al., 2024; Packer et al., 2023; Chhikara et al., 2025). More recent approaches learn memory admission or update policies from downstream supervision (Zhang et al., 2026; Wang et al., 2025). Yet a central question in bounded replacement remains: *how does each possible memory update change the future utility of the complete memory state?* An incoming memory may appear useful in isolation but be redundant with the current bank or displace information that is more valuable for future interactions.

We introduce **EvoMemory**, which formulates bounded memory replacement as a sequential decision problem over alternative successor memory states. For each incoming memory, EvoMemory considers preserving the current bank or replacing one stored item. During offline training, these

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.

counterfactual successor states are evaluated on future examples to construct relative utility supervision, from which a contextual value model learns to compare their downstream consequences. At deployment, the learned policy makes the same decision using only the current bank and incoming memory, without future lookahead. We evaluate EvoMemory on JRE-L and WikiLarge with a frozen target LLM and a fixed memory budget. EvoMemory improves both online memory-update quality and held-out downstream utility over heuristic and random replacement strategies, with statistically significant gains on the held-out test sets. Ablations and robustness analyses further support the benefits of non-myopic supervision and contextual successor-state modeling.

Our main contributions are:

- We formulate bounded memory replacement as a sequential decision problem in which updates are valued by the future utility of their resulting memory states.
- We introduce **EvoMemory**, which learns a contextual value model over candidate successor states from offline counterfactual utility supervision, enabling online memory updates without future lookahead.
- Across JRE-L and WikiLarge, EvoMemory improves online memory-update quality and held-out downstream utility over heuristic and random strategies; controlled analyses support the benefits of non-myopic supervision and contextual successor-state valuation.

2 Related Work

2.1 Long-Term Memory for Language Models

External memory extends language models beyond fixed context windows and supports persistent information across interactions, personalization, and agentic tasks (Park et al., 2023; Packer et al., 2023; Zhong et al., 2024). A large body of prior work focuses on memory representation, retrieval, organization, or incorporation into generation. We instead study how a fixed-capacity external memory should evolve when retaining new information requires displacing existing content.

2.2 Learning Memory Management Policies

Memory management has evolved from heuristic storage and retrieval rules toward adaptive policies for retaining, updating, and discarding information. MemoryBank uses time- and importance-based forgetting (Zhong et al., 2024), while MemGPT manages information across hierarchical storage tiers (Packer et al., 2023). More recent systems jointly adapt multiple components of the memory pipeline: A-MEM constructs and links evolving structured memories (Xu et al., 2025), while Mem0 extracts, consolidates, and retrieves salient conversational information (Chhikara et al., 2025). Direct end-to-end comparisons with such systems can conflate replacement quality with differences in memory construction, representation, organization, and retrieval.

Recent work increasingly studies memory management through downstream consequences. Work on memory addition and deletion shows that future-task evaluations can provide useful quality signals for stored experiences, including under memory constraints (Xiong et al., 2026). A-MAC learns candidate-level admission from features including future utility, confidence, novelty, and recency (Zhang et al., 2026), while Mem-α uses reinforcement learning to construct and update a multi-component memory system from downstream task rewards (Wang et al., 2025). These works move beyond fixed heuristics, but focus on experience quality, admission, or broader memory construction rather than comparing the complete successor states induced by alternative updates under a fixed memory budget.

EvoMemory instead isolates *bounded memory replacement*, holding the memory representation, budget, target model, and downstream use fixed. For each incoming memory, it compares KEEP with all single-item replacements by the future utility of their resulting *successor memory states*, separating replacement quality from other memory-system design choices.

Learned eviction has also been studied for transformer key–value caches. KV Policy ranks cached tokens by predicted usefulness for future decoding (Moschella et al., 2026). Although conceptually related, KV-cache eviction operates over internal token-level inference states rather than explicit

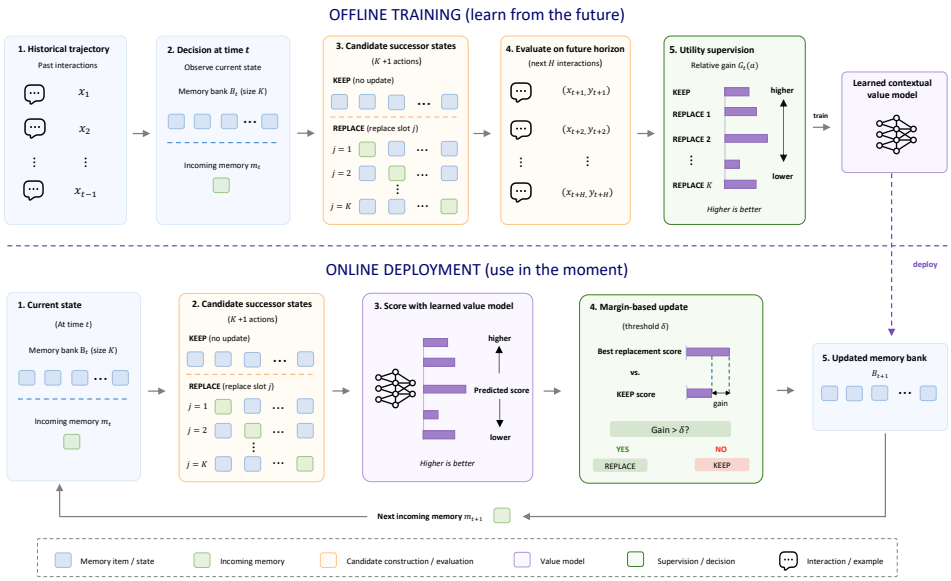

Figure 1: Overview of EvoMemory. **Offline training:** for each memory-management decision, EvoMemory constructs the $K + 1$ candidate successor memory states and evaluates them on future examples to obtain counterfactual utility supervision for learning a contextual value model. **Online deployment:** given the current memory bank and an incoming memory, the learned value model scores the same candidate updates using only information available at decision time; a margin-based policy then either preserves the current bank or applies the highest-valued replacement. Future examples are used only during offline supervision and are never available to the deployed policy.

persistent memories, making it a different memory substrate and action space. The key distinction is that EvoMemory does not assign utility to memories independently; it learns the relative future utility of the complete successor states induced by alternative memory-update actions.

3 Method

We consider a language-model system equipped with a bounded external memory bank that evolves as new information arrives. At each step, the system must decide whether to retain the current memory bank or admit the incoming memory by replacing one stored item. The central challenge is that the usefulness of an update cannot be determined from the incoming memory alone: its value depends on what is already stored, what would be displaced, and how the resulting memory state affects future interactions.

EvoMemory addresses this problem by learning to score complete successor memory states according to their future utility. During offline training, for each incoming memory we enumerate all feasible updates—KEEP and replacement of each existing memory item—and evaluate the resulting successor states on subsequent examples. These counterfactual evaluations provide relative utility-gain supervision for a contextual value model. At deployment, future examples are unavailable; the learned model instead scores the same candidate successor states using only the current memory bank and incoming memory, enabling online memory updates without future lookahead. Figure 1 summarizes the framework.

3.1 Problem Formulation

Let $\mathcal{D} = \{x_1, \ldots, x_T\}$ denote a temporally ordered stream. At time t, the system maintains a bounded memory bank $B_t = \{b_t^{(1)}, \ldots, b_t^{(K)}\}$ and receives an incoming memory m_t. Once the bank is full, the available actions are

$$\mathcal{A}_t = \{\textsc{Keep}\} \cup \{\textsc{Replace}(j) : j = 1, \ldots, K\},$$

where $B_t^{\text{KEEP}} = B_t$ and $B_t^{\text{REPLACE}(j)} = (B_t \setminus \{b_t^{(j)}\}) \cup \{m_t\}$. Thus, each decision induces one of $K+1$ successor memory states.

Let $U(B; \mathcal{F})$ denote the downstream utility obtained when a frozen target language model performs a task on a future example set $\mathcal{F}$ while conditioned on memory bank B. EvoMemory seeks the highest-utility successor state, but must learn to score candidate states without access to future examples at deployment.

3.2 FUTURE UTILITY OF MEMORY STATES

We define the quality of a candidate memory state by its downstream utility over the next H examples,

$$\mathcal{F}_t = \{x_{t+1}, \ldots, x_{t+H}\},$$

We consider only decision points for which the complete H-example future window is available. For a candidate successor bank B_t^a, its future utility is

$$V_t(a) = U(B_t^a; \mathcal{F}_t) = \frac{1}{|\mathcal{F}_t|} \sum_{x \in \mathcal{F}_t} R(f_\theta(x; B_t^a), y_x)\,,$$

where f_θ is a frozen target language model, y_x is the reference output when available, and $R(\cdot)$ is the task-specific reward function. The oracle value at decision t is

$$V_t^* = \max_{a \in \mathcal{A}_t} V_t(a),$$

and the corresponding oracle action is

$$a_t^* = \arg\max_{a \in \mathcal{A}_t} V_t(a).$$

This formulation exposes two quantities that are useful for evaluating memory policies. First, the gain of an action relative to preserving the current memory is

$$G_t(a) = V_t(a) - V_t(\text{KEEP}).$$

Second, the oracle regret incurred by selecting action a is

$$\mathcal{R}_t(a) = V_t^* - V_t(a).$$

3.3 UTILITY SUPERVISION

During offline training, we evaluate all $K+1$ candidate successor states $\{B_t^a : a \in \mathcal{A}_t\}$ on the same future horizon $\mathcal{F}_t$, yielding counterfactual utilities $\{V_t(a) : a \in \mathcal{A}_t\}$. Future examples are used only to construct offline supervision; the deployed policy observes only the current bank and incoming memory.

Using the relative gain $G_t(a)$ defined above, we preserve the graded structure of the counterfactual utilities through a soft target distribution,

$$q_t(a) = \frac{\exp\left(sG_t(a)/\tau_{\text{tar}}\right)}{\sum_{a' \in \mathcal{A}_t} \exp\left(sG_t(a')/\tau_{\text{tar}}\right)},$$

where s is a utility scaling factor and $\tau_{\text{tar}} > 0$ is the target temperature.

Let $\hat{G}_\phi(B_t, m_t, a)$ denote the predicted scaled utility-gain score for action a. We define the predicted action distribution as

$$p_{\phi,t}(a) = \frac{\exp\left(\hat{G}_\phi(B_t, m_t, a)/\tau_{\text{pred}}\right)}{\sum_{a' \in \mathcal{A}_t} \exp\left(\hat{G}_\phi(B_t, m_t, a')/\tau_{\text{pred}}\right)},$$

where $\tau_{\text{pred}} > 0$ is the prediction temperature.

For N offline training decisions, we optimize a soft ranking objective

$$\mathcal{L}_{\text{rank}} = -\frac{1}{N} \sum_t \sum_{a \in \mathcal{A}_t} q_t(a) \log p_{\phi,t}(a).$$

To additionally preserve the magnitude of relative utility differences, we include a Smooth L1 regression objective,

$$\mathcal{L}_{\text{reg}} = \frac{1}{N(K+1)} \sum_{t} \sum_{a \in \mathcal{A}_t} \text{SmoothL1}\left(\hat{G}_\phi(B_t, m_t, a) - sG_t(a)\right).$$

The overall training objective is

$$\mathcal{L} = \mathcal{L}_{\text{rank}} + \lambda \mathcal{L}_{\text{reg}},$$

where λ controls the contribution of utility-gain regression.

3.4 Contextual Value Model

Because the utility of an update depends on both the incoming memory and the existing bank, EvoMemory scores candidate successor states contextually. For each action a, we encode its resulting successor bank B_t^a together with the incoming memory m_t as

$$z_t^a = g_\phi(B_t^a, m_t),$$

and predict its scaled utility-gain score as

$$\hat{G}_\phi(B_t, m_t, a) = h_\phi(z_t^a) \approx sG_t(a).$$

Because all $K+1$ candidates are evaluated for the same decision, the model can account for redundancy within the bank and the information displaced by each replacement. The target language model remains frozen throughout training.

3.5 Online Memory Update

At deployment time, EvoMemory observes only the current bank B_t and incoming memory m_t and predicts a score for each of the $K+1$ candidate successor states. Let the highest-scoring replacement be

$$j^* = \arg \max_{j \in \{1,\dots,K\}} \hat{G}_\phi(B_t, m_t, \textsc{Replace}(j)),$$

and let

$$\hat{G}_{\text{keep}} = \hat{G}_\phi(B_t, m_t, \textsc{Keep}).$$

To avoid updates driven by small prediction differences, EvoMemory replaces a memory only when the predicted replacement advantage over Keep exceeds an update margin δ:

$$\pi_\phi(B_t, m_t) = \begin{cases} \textsc{Replace}(j^*), & \text{if } \hat{G}_\phi(B_t, m_t, \textsc{Replace}(j^*)) - \hat{G}_{\text{keep}} > \delta, \\ \textsc{Keep}, & \text{otherwise.} \end{cases}$$

The margin controls update conservativeness and is selected on validation data and fixed before online evaluation. The selected successor state becomes B_{t+1} and is used for subsequent decisions. Theoretical complexity and empirical runtime and memory overhead are reported in Appendix A.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate on two complementary benchmarks. For JRE-L from MemoryBench (Ai et al., 2026) (`THUIR/MemoryBench`), completed historical interactions are converted into abstract memories while preserving stream order. We use $[0, 120)$ for utility-model training, $[120, 160)$ for validation, and $[160, 200)$ for online evaluation; the 50-example official test set is reserved for final-state evaluation. For WikiLarge (Zhang & Lapata, 2017), we construct a sequential sentence-simplification memory stream with 953 aligned examples, using $[0, 650)$ for utility-model training, $[650, 800)$ for validation, and $[800, 953)$ for online evaluation; the 121-example official test set is reserved for final-state evaluation. For online evaluation, the first K examples initialize the memory bank, and only decision points with a complete H-example future window are evaluated, yielding 26 decisions on JRE-L and 139 on WikiLarge. Dataset construction and preprocessing are detailed in Appendix B.

Target model and memory construction. We use Qwen3-8B as the frozen target model throughout. Stream examples are converted into compact abstract memories while preserving their order. Memory-construction and decoding details are provided in Appendices B and C.

Baselines. We compare against KEEP (no update), FIFO (replace oldest), Similarity (replace the stored item most similar to the incoming memory), Importance (LLM-based selection of the least useful item), and Random (uniform replacement; five seeds). All methods use identical memory representations, budgets, target models, and evaluation protocols. Full definitions are provided in Appendix D. We additionally evaluate a learned item-level utility control on WikiLarge to isolate the effect of contextual successor-state valuation.

Evaluation metrics. We evaluate memory management at two levels. For online decisions, we report oracle action accuracy, chosen future utility, gain relative to preserving the current bank, oracle regret, and regret reduction relative to KEEP. These metrics characterize the quality of online update decisions along each policy's induced memory trajectory. For final-state evaluation, we freeze the memory bank produced by each policy and evaluate all methods on the same held-out examples. On both benchmarks, the downstream reward is

$$R = 0.5\,\text{ROUGE-L} + 0.5\,\text{BERTScore-F1}.$$

We additionally report the two component metrics separately.

Implementation details. Unless otherwise specified, we use a memory budget of $K = 4$ and a future utility horizon of $H = 10$. The utility model uses RoBERTa-base as its text encoder and a two-layer Transformer with a hidden dimension of 256 and four attention heads for contextual bank encoding. Hyperparameters, validation-based margin selection, and additional implementation details are provided in Appendix C.

4.2 Main Results

We organize our main evaluation around two complementary questions. First, does EvoMemory make better memory-update decisions as information arrives (**RQ1**)? Second, do these decisions produce a memory state that improves performance on unseen downstream examples (**RQ2**)? We evaluate the first question using online decision metrics and oracle counterfactual utility, and the second using a disjoint held-out test set.

4.2.1 EvoMemory improves online memory decisions

EvoMemory achieves the highest action accuracy on both benchmarks and substantially lower oracle regret than KEEP (Table 1). Similarity-based replacement yields lower action accuracy and higher oracle regret than EvoMemory on both benchmarks, indicating that semantic redundancy alone is insufficient for bounded memory replacement.

4.2.2 Better memory decisions are associated with higher held-out utility

To assess whether improved online decisions are accompanied by more useful memory states, we freeze the final memory bank produced by each policy and evaluate it without further updates on the held-out test sets.

EvoMemory achieves the strongest final-state performance on both benchmarks (Table 2). On JRE-L, it reaches a reward of 0.4719, exceeding the strongest deterministic heuristic (0.4590) and the five-seed random baseline (0.4599). On WikiLarge, it reaches a reward of 0.8152, improving over KEEP by 0.0264 and the five-seed random baseline by 0.0426, with gains in both BERTScore-F1 and ROUGE-L. Together with the lower online oracle regret in Table 1, these results show that EvoMemory's improved memory-update decisions are accompanied by higher downstream utility on unseen examples.

4.2.3 The held-out improvements are statistically robust

Paired statistical tests confirm that the held-out improvements are statistically significant across both benchmarks. EvoMemory improves reward over all five baselines on both benchmarks, with

Table 1: Online memory-management performance on JRE-L and WikiLarge. Action Acc. measures agreement with the oracle update action; Chosen Future is the realized future utility following the selected action; Gain vs. KEEP is relative to retaining the current policy-specific memory state; and Oracle Regret is the utility gap to the oracle action. Regret Red. measures the reduction in Oracle Regret relative to choosing KEEP at the same policy-specific state. Random reports mean ± standard deviation over five seeds.

Method	Action Acc. ↑	Chosen Future ↑	Gain vs. KEEP ↑	Oracle Regret ↓	Regret Red. ↑
JRE-L					
KEEP	0.2308	0.4591	0.0000	0.0045	0.00%
FIFO	0.1923	0.4581	−0.0005	0.0040	−15.84%
Similarity	0.1538	0.4630	−0.0001	0.0041	−2.38%
Importance	0.2308	0.4591	0.0000	0.0045	0.00%
Random	0.1538±0.0666	0.4579±0.0018	−0.0002±0.0004	0.0042±0.0007	−6.32±11.11%
EvoMemory	**0.4231**	**0.4703**	**+0.0004**	**0.0009**	**+31.61%**
WikiLarge					
KEEP	0.2014	0.7383	0.0000	0.0162	0.00%
FIFO	0.2374	0.7372	−0.0002	0.0143	−1.56%
Similarity	0.2230	0.7441	−0.0007	0.0163	**4.11%**
Importance	0.2014	0.7383	0.0000	0.0162	0.00%
Random	0.1986±0.0308	0.7357±0.0019	0.0000±0.0004	0.0163±0.0007	−0.04±2.48%
EvoMemory	**0.3237**	**0.7470**	**+0.0003**	**0.0067**	+3.61%

Table 2: Held-out downstream performance on JRE-L and WikiLarge. Reward is the predefined utility combining ROUGE-L and BERTScore-F1. Random reports mean ± standard deviation over five seeds. Best results within each benchmark are shown in bold.

Method	Reward ↑	BERT-F1 ↑	ROUGE-L ↑
JRE-L			
KEEP	0.4578	0.8043	0.1113
FIFO	0.4586	0.8067	0.1104
Similarity	0.4590	0.8063	0.1117
Importance	0.4578	0.8043	0.1113
Random	0.4599±0.0034	0.8069±0.0038	0.1130±0.0031
EvoMemory	**0.4719**	**0.8177**	**0.1261**
WikiLarge			
KEEP	0.7887	0.9337	0.6437
FIFO	0.7763	0.9330	0.6197
Similarity	0.7619	0.9282	0.5955
Importance	0.7887	0.9337	0.6437
Random	0.7726±0.0113	0.9311±0.0020	0.6140±0.0209
EvoMemory	**0.8152**	**0.9393**	**0.6910**

gains of 0.0120–0.0141 on JRE-L and 0.0264–0.0533 on WikiLarge; all remain significant after Holm correction ($p < 0.001$). BERTScore-F1 and ROUGE-L show the same pattern, with all comparisons remaining significant after correction. Reward confidence intervals, corrected p-values, and statistical testing details are reported in Appendix E.

4.3 Ablation Studies

We ablate three key components of EvoMemory on WikiLarge: contextual state modeling, non-myopic future-utility supervision, and the regression objective. We report two online metrics—gain over KEEP and regret reduction—together with held-out reward, which measures the downstream quality of the resulting memory bank.

Table 3: Ablation study on WikiLarge. Online performance is summarized by gain over KEEP and regret reduction, while held-out reward evaluates the quality of the resulting memory bank on unseen examples.

Variant	Gain vs. KEEP ↑	Regret Red. ↑	Held-out Reward ↑
EvoMemory	**+0.000253**	**+3.61%**	**0.8152**
w/o Context	+0.000024	+0.29%	0.7946
Myopic Utility ($H = 1$)	−0.000041	−0.31%	0.7924
w/o Regression	+0.000144	+1.33%	0.7984

Removing contextual state modeling substantially weakens the memory-update policy: gain over KEEP decreases from $+0.000253$ to $+0.000024$, regret reduction drops from 3.61% to 0.29%, and held-out reward decreases from 0.8152 to 0.7946 (Table 3). Replacing non-myopic future-utility supervision with a myopic objective ($H = 1$) is similarly detrimental, yielding negative gain over KEEP and negative regret reduction while reducing held-out reward to 0.7924. Together, these results support modeling the memory bank as a contextual state and evaluating memory updates according to their future consequences.

Removing utility-gain regression also reduces all three metrics, with held-out reward decreasing from 0.8152 to 0.7984. This suggests that explicitly modeling utility-gain magnitudes provides useful supervision beyond the soft ranking objective.

Comparison with learned item-level utility. To test whether EvoMemory's gains can be reproduced by learning item-level utility alone, we construct a control on WikiLarge using the same offline counterfactual utility evaluations. Rather than valuing complete successor states, the control independently predicts the utility difference between admitting the incoming memory and retaining each existing item, and replaces an item only when the predicted improvement exceeds a validation-selected margin. Memory representations, budgets, target models, data splits, and evaluation protocols are otherwise identical.

As shown in Table 4, the item-level control makes no updates under its selected margin, yielding zero gain over KEEP and the same held-out reward (0.7887). EvoMemory instead achieves positive online gain (+0.000253), 3.61% regret reduction, and a held-out reward of 0.8152. This suggests that, in this controlled setting, item-level utility alone does not capture the context-dependent consequences of replacement; explicitly valuing complete successor memory states yields more effective memory updates.

Table 4: Controlled comparison with learned item-level utility on WikiLarge. The item-level control scores memories independently, whereas EvoMemory models the contextual value of complete successor memory states.

Method	Action Acc. ↑	Gain vs. KEEP ↑	Regret Red. ↑	Reward ↑
Learned Item Utility	0.3022	0.000000	0.00%	0.7887
EvoMemory	**0.3237**	**+0.000253**	**+3.61%**	**0.8152**

Selective update behavior. EvoMemory replaces memory in only 10 of 139 WikiLarge decisions (7.2%). Although five replacements improve future utility and five reduce it, beneficial updates have more than twice the mean magnitude of harmful ones (0.0123 vs. 0.0053), yielding a positive mean gain of 0.0035 across accepted updates. Thus, the positive average gain arises from the larger magnitude of beneficial updates rather than from a higher frequency of beneficial replacements.

4.4 Sensitivity and Robustness

Future utility horizon. We examine the sensitivity of EvoMemory to the future horizon used for constructing offline utility supervision. We vary the training horizon $H_{\text{train}} \in \{1, 5, 10, 20\}$ on WikiLarge while keeping the memory budget ($K = 4$), model architecture, training procedure, and

random seed fixed. To ensure a controlled comparison, all resulting policies are evaluated using the same online evaluation horizon $H_{\text{eval}} = 10$ and the same held-out test set.

As shown in Table 5, myopic supervision ($H_{\text{train}} = 1$) performs substantially worse, yielding a negative gain over KEEP and a held-out reward of 0.7924. In contrast, all longer horizons ($H_{\text{train}} \in \{5, 10, 20\}$) achieve positive gains over KEEP and substantially higher held-out rewards of 0.8122–0.8155. Performance among these longer horizons is relatively stable: although $H_{\text{train}} = 5$ yields the largest online gain and regret reduction, $H_{\text{train}} = 10$ and 20 obtain slightly higher held-out rewards. These results suggest that the main benefit arises from non-myopic future-utility supervision rather than from a particular choice among the longer horizons.

Table 5: Sensitivity to the future utility horizon on WikiLarge. Models are trained with different H_{train} and evaluated using a common $H_{\text{eval}} = 10$.

H_{train}	Gain vs. KEEP ↑	Regret Red. ↑	Held-out Reward ↑
1	−0.000041	−0.31%	0.7924
5	+0.000478	+6.18%	0.8122
10	+0.000253	+3.61%	0.8152
20	+0.000306	+3.79%	0.8155

Memory-budget sensitivity. We vary the memory budget $K \in \{2, 4, 8\}$ while fixing $H = 10$. Performance is non-monotonic with memory capacity (Table 6): $K = 4$ provides the strongest overall performance, while both smaller and larger budgets yield lower held-out utility. EvoMemory nevertheless maintains positive gain over KEEP and positive regret reduction across all evaluated memory budgets.

Table 6: Sensitivity to memory budget on WikiLarge with $H = 10$.

Memory Budget K	Gain vs. KEEP ↑	Regret Reduction ↑	Held-out Reward ↑
2	+0.000146	+2.42%	0.7927
4	**+0.000253**	**+3.61%**	**0.8152**
8	+0.000224	+2.25%	0.8048

Robustness to the utility-model backbone. We further examine whether EvoMemory's gains depend on the RoBERTa-base utility-model backbone by replacing it with the decoder-only Qwen3-0.6B while keeping all other experimental settings fixed. EvoMemory retains a positive gain over KEEP (+0.000212) and a 3.30% reduction in oracle regret, while achieving a held-out reward of 0.8096 compared with 0.8152 using RoBERTa-base (Appendix F.2). These results indicate that EvoMemory's gains are not limited to the RoBERTa-base utility-model backbone and also persist with a decoder-only utility model.

5 Conclusion and Future Work

We introduced EvoMemory, a framework for learning bounded memory replacement from counterfactual future utility. EvoMemory estimates the relative value of complete successor memory states, enabling online updates without future lookahead. Across two benchmarks, it improves held-out utility over heuristic and random replacement, while controlled analyses support the benefits of non-myopic supervision and contextual successor-state valuation.

Our evaluation is limited to two benchmarks, a single stream ordering per benchmark, fixed-capacity memory with single-item replacement, and a single frozen target LLM; counterfactual supervision also incurs additional offline cost.

Future work could extend EvoMemory beyond fixed-size, replacement-only memory to dynamic and hierarchical memory structures, longer interaction streams, and richer operations such as consolidation, abstraction, and revision. Another promising direction is to jointly learn memory extraction and management and adapt memory utility from online feedback, moving toward memory systems that continuously evolve with experience.

AI USE STATEMENT

Generative AI tools were used to assist with code debugging, result interpretation, and manuscript editing. All AI-assisted outputs were reviewed and verified by the authors, who take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of dataset construction and preprocessing, model architecture and optimization settings, baseline definitions, evaluation protocols, and statistical testing procedures in the appendix. Additional details on computational complexity, hyperparameters, and robustness analyses are also provided to support reproduction of the reported experiments.

REFERENCES

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. Memory-bench: A benchmark for memory and continual learning in LLM systems. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022.

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Luca Moschella, Laura Manduchi, and Ozan Sener. Learning to evict from key-value cache. In *Forty-third International Conference on Machine Learning*, 2026.

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

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. ACM, 2023. doi: 10.1145/3586183.3606763.

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

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.27.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-0593.

Guilin Zhang, Wei Jiang, Xiejiashan Wang, Aisha Behr, Kai Zhao, Jeffrey Friedman, Xu Chu, and Amine Anoun. Adaptive memory admission control for LLM agents. *arXiv preprint arXiv:2603.04549*, 2026.

Xingxing Zhang and Mirella Lapata. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 584–594. Association for Computational Linguistics, 2017.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, et al. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025. doi: 10.1145/3748302.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731, 2024. doi: 10.1609/aaai.v38i17.29946.

A Computational Complexity and Runtime

EvoMemory separates expensive offline utility construction from lightweight online memory management. For a memory budget K, each training decision has $K + 1$ candidate successor states. Evaluating every candidate on a future horizon of H examples therefore requires

$$O((K + 1)H)$$

target-model evaluations per decision in the straightforward implementation. This counterfactual evaluation is performed only when constructing offline utility supervision and can be parallelized across candidate states and future examples.

At deployment time, future examples are never evaluated. EvoMemory only scores the $K + 1$ candidate actions with the learned value model and selects between KEEP and the highest-valued replacement. The online memory-management cost is therefore

$$O(K)$$

value-model evaluations per decision, in addition to the target-model computation required by the underlying task.

The memory footprint remains bounded by

$$O(K),$$

independent of the length of the interaction stream. This distinction is important: future rollouts provide supervision for learning the policy offline, but the resulting memory manager does not require lookahead during deployment.

Empirical runtime and memory overhead. We additionally profile the computational overhead of online memory management under the main WikiLarge setting ($K = 4$, $H = 10$). Latency is measured over the 139 sequential held-out memory decisions after warm-up, using each method's original online decision path. Timing includes candidate-state construction, policy scoring or selection, and the resulting memory-bank transition, while excluding future-window evaluation, reward computation, downstream task generation, and logging. All profiled policies reproduce the final memory bank of their original rollout.

Table 7: Computational overhead on the WikiLarge main setting. Offline preparation denotes additional task-specific computation beyond the shared memory-construction pipeline. Latency is reported as mean ± standard deviation over 139 online memory decisions. Peak GPU memory denotes peak allocated memory during online memory management; CPU-only methods require no GPU for the memory decision.

Method	Offline Prep.	Latency (ms)	P95 (ms)	Peak GPU Mem.	Target LLM Online
KEEP	None	0.578±0.008	0.591	CPU-only	No
FIFO	None	0.576±0.007	0.589	CPU-only	No
Random	None	0.572±0.006	0.579	CPU-only	No
Similarity	None	20.03±5.79	31.99	CPU-only	No
Importance	None	117.51±1.36	118.38	15.343 GiB	Yes
EvoMemory	15.07 GPU-h	24.50±2.06	25.89	0.499 GiB	No

The results illustrate the intended offline–online trade-off. Parameter-free heuristics incur negligible online cost. Under their original deployment configurations, semantic similarity requires 20.0 ms per decision on CPU, whereas EvoMemory requires 24.5 ms per decision on GPU while performing learned future-utility scoring. In contrast, the LLM-based Importance baseline requires online inference with the frozen Qwen3-8B target model, resulting in 117.5 ms per decision and 15.34 GiB of peak GPU memory. EvoMemory therefore trades a one-time task-specific offline preparation cost of 15.07 GPU-hours for lightweight online decisions without target-LLM inference.

B Dataset Construction and Preprocessing

B.1 JRE-L

We use the JRE-L configuration of MemoryBench (`THUIR/MemoryBench`). Each training example contains a completed historical interaction, which is treated as one candidate memory unit. For the main experiments, we use abstract memories generated from these interactions rather than storing the full raw dialogs. The resulting memories retain the same ordering as the official JRE-L training split and define a sequential memory stream. We use $[0, 120)$ for utility-model training, $[120, 160)$ for validation, and $[160, 200)$ for online memory-management evaluation; the official 50-example test split is kept untouched until final-state evaluation.

B.2 WikiLarge

We construct a sentence-simplification memory stream from the training split of WikiLarge. Starting from `eilamc14/wikilarge-clean`, we shuffle the training data with seed 42 and select 1,000 aligned complex–simple sentence pairs after filtering malformed or weakly aligned examples. We require source sentences to contain 8–60 words and targets 5–60 words, constrain the target-to-source length ratio to $[0.30, 1.30]$, require a content-word Jaccard overlap of at least 0.30, reject target-side numbers absent from the source, and remove near-identical pairs with normalized similarity above 0.96.

Each retained source–target pair is converted into a compact abstract memory capturing a reusable simplification principle rather than the complete example. Memories preserve their trajectory order and are assigned timestamps according to their positions in the stream. After abstract-memory construction and validity filtering, 953 aligned examples remain in the memory trajectory. We use $[0, 650)$ for utility-model training, $[650, 800)$ for validation, and $[800, 953)$ for online memory-management evaluation; the official 121-example test set is kept untouched until final-state evaluation.

C Implementation Details

Target language model. We use Qwen3-8B as the frozen downstream language model throughout the main experiments. EvoMemory never updates the parameters of this target model; only the external memory state is changed over time. For WikiLarge, we use the chat template with reasoning disabled and deterministic decoding (`do_sample=False`), with a maximum generation length of 128 tokens. The same frozen target model is used when constructing counterfactual utility labels, running online memory-management policies, and evaluating the final memory bank, ensuring that differences across policies arise from memory management rather than target-model adaptation.

Utility model. The main utility model uses RoBERTa-base to encode individual memories and incoming memories. Memory and incoming texts are truncated to 256 tokens. Memory representations are projected to a 256-dimensional state space and processed by a two-layer Transformer with four attention heads and dropout of 0.1 to obtain a contextual bank representation. The incoming-memory representation is projected separately and combined with the bank representation to predict the candidate state's utility-gain score. Unless otherwise specified, the memory budget is $K = 4$ and the future utility horizon is $H = 10$.

The utility target is the candidate's future utility gain relative to KEEP, $G_t(a)$, scaled by a factor of 100 during training. For the graded ranking objective, we use a soft-target temperature of 0.25 and a prediction temperature of 1.0. The regression component is weighted by $\lambda = 0.05$.

Optimization. We train for 15 epochs with a batch size of 4. The best checkpoint is selected by minimizing mean oracle regret on the validation stream, with ties broken by larger mean gain over KEEP and then by higher action accuracy. We use separate learning rates of 1×10^{-5} for the pretrained text encoder and 1×10^{-4} for the remaining model parameters, with weight decay of 0.01. The learning-rate schedule uses a warm-up ratio of 0.1, and gradient norms are clipped at 1.0.

Update-margin selection. The update margin δ is selected exclusively on the validation stream. For each trained model, we evaluate $\delta \in \{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.5\}$ and select the margin that minimizes mean oracle regret, breaking ties in favor of larger mean gain over KEEP. The selected margin is $\delta = 0.3$ for both JRE-L and WikiLarge and is fixed before online evaluation. At each deployment step, EvoMemory identifies the highest-scoring replacement candidate and performs the replacement only when its predicted advantage over KEEP exceeds δ; otherwise, the incoming memory is ignored.

Table 8: Hyperparameters for the main utility model.

Hyperparameter	Value
Text encoder	RoBERTa-base
Maximum memory length	256
Maximum incoming length	256
Memory budget K	4
Future horizon H	10
Margin candidates δ	$\{0, 0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.5\}$
Selected margin δ	0.3
State hidden dimension	256
State Transformer layers	2
Attention heads	4
Dropout	0.1
Target scaling factor	100
Soft-target temperature	0.25
Prediction temperature	1.0
Regression weight λ	0.05
Training epochs	15
Batch size	4
Encoder learning rate	1×10^{-5}
Non-encoder learning rate	1×10^{-4}
Weight decay	0.01
Warm-up ratio	0.1
Maximum gradient norm	1.0

D BASELINE DETAILS

KEEP never updates the initial memory bank. FIFO replaces the oldest stored memory with each incoming memory. Similarity uses `sentence-transformers/all-MiniLM-L6-v2` to embed the current memories and incoming memory and replaces the stored item with the highest semantic similarity to the incoming memory, treating highly similar items as redundant. Importance uses the frozen target language model to jointly assess the current bank and incoming memory and selects the single memory least useful to retain for future tasks; selecting the incoming memory corresponds to KEEP, whereas selecting an existing memory replaces that slot. Random uniformly selects an existing slot for replacement. We report Random as mean and standard deviation over five seeds.

E STATISTICAL SIGNIFICANCE ANALYSIS

Because every policy is evaluated on the same held-out examples within each benchmark, we use paired statistical tests. For the random baseline, we first average performance across the five random seeds within each test example and then perform the paired comparison, preserving the test example as the unit of analysis.

For each benchmark, we estimate 95% confidence intervals for the mean performance difference using 10,000 paired bootstrap resamples and compute two-sided paired randomization tests using 100,000 sign permutations. We apply Holm correction across the 15 comparisons defined by five baselines and three held-out metrics within each benchmark.

Table 9: Paired analysis of held-out reward on JRE-L and WikiLarge. Differences are EvoMemory minus each baseline and are computed from unrounded per-example scores; summary values elsewhere are rounded. Confidence intervals use 10,000 paired bootstrap resamples. p-values are from two-sided paired randomization tests with Holm correction across 15 baseline–metric comparisons within each benchmark.

Dataset	Comparison	Δ Reward ↑	95% CI	Holm p
JRE-L	EvoMemory vs. KEEP	+0.0141	[0.0102, 0.0181]	< 0.001
	EvoMemory vs. FIFO	+0.0134	[0.0106, 0.0162]	< 0.001
	EvoMemory vs. Similarity	+0.0129	[0.0097, 0.0164]	< 0.001
	EvoMemory vs. Importance	+0.0141	[0.0101, 0.0180]	< 0.001
	EvoMemory vs. Random	+0.0120	[0.0094, 0.0148]	< 0.001
WikiLarge	EvoMemory vs. KEEP	+0.0264	[0.0169, 0.0367]	< 0.001
	EvoMemory vs. FIFO	+0.0388	[0.0241, 0.0540]	< 0.001
	EvoMemory vs. Similarity	+0.0533	[0.0396, 0.0676]	< 0.001
	EvoMemory vs. Importance	+0.0264	[0.0166, 0.0365]	< 0.001
	EvoMemory vs. Random	+0.0426	[0.0314, 0.0538]	< 0.001

F Additional Ablation and Robustness Studies

F.1 Graded vs. Hard Utility Supervision

We additionally examine whether preserving graded differences among counterfactual utilities is beneficial compared with reducing each decision to a single hard oracle-action label. In the hard-target variant, the highest-utility action is treated as the positive target and all remaining actions as negative. All other model, training, and evaluation settings are kept identical to the full EvoMemory model.

Table 10: Effect of graded versus hard utility supervision on WikiLarge. Held-out reward is evaluated on the 121-example test set.

Supervision	Action Acc. ↑	Gain vs. KEEP ↑	Held-out Reward ↑
Hard target	**0.3669**	+0.000092	0.8133
Graded	0.3237	**+0.000253**	**0.8152**

Hard-target supervision achieves higher discrete action accuracy (0.3669 vs. 0.3237), but this does not translate into stronger utility-based performance. Graded supervision produces a larger gain over KEEP (+0.000253 vs. +0.000092) and a higher held-out reward (0.8152 vs. 0.8133). This suggests that preserving relative utility differences among candidate successor states provides a more informative training signal for downstream memory quality than reducing each decision to a one-hot oracle-action target.

F.2 Utility-Model Backbone Robustness

The main EvoMemory experiments use RoBERTa-base as the utility-model backbone. To examine whether the effectiveness of the learned memory-management policy depends on this particular encoder architecture, we replace RoBERTa-base with Qwen3-0.6B, a decoder-only language model. We keep the memory-management protocol, memory budget ($K = 4$), training horizon ($H = 10$), utility supervision, data splits, and frozen target model unchanged. Thus, the utility-model backbone is the only component varied in this comparison.

As shown in Table 11, EvoMemory remains effective when the RoBERTa-base encoder is replaced with the decoder-only Qwen3-0.6B backbone. The Qwen3-0.6B variant retains a positive gain over KEEP (+0.000212) and reduces oracle regret by 3.30%, close to the +0.000253 gain and 3.61% regret reduction obtained with RoBERTa-base. Held-out downstream performance is also comparable: Qwen3-0.6B achieves a reward of 0.8096 compared with 0.8152 for RoBERTa-base, while achieving a slightly higher BERT-F1 (0.9401 vs. 0.9393). These results provide evidence that EvoMemory's

Table 11: Robustness to the utility-model backbone on WikiLarge. All settings other than the utility-model backbone are held fixed.

Backbone	Gain vs. KEEP ↑	Regret Red. ↑	Reward ↑	BERT-F1 ↑	ROUGE-L ↑
RoBERTa-base	**+0.000253**	**+3.61%**	**0.8152**	0.9393	**0.6910**
Qwen3-0.6B	+0.000212	+3.30%	0.8096	**0.9401**	0.6790

gains are not specific to the RoBERTa-base utility-model backbone and persist with a decoder-only utility model.