MemRL: Self-Evolving Agents via Runtime Reinforcement Learning on Episodic Memory

Jiaqian Wang[1,*] **Shengtao Zhang**[2,*] **Ruiwen Zhou**[3] **Junwei Liao**[2,4] **Yuchen Feng**[5]
Zhuo Li[2] **Yujie Zheng**[2] **Weinan Zhang**[2,4] **Ying Wen**[2,4] **Zhiyu Li**[5]
Feiyu Xiong[5] **Yutao Qi**[1] **Bo Tang**[6,5,†] **Muning Wen**[2,†]

[1]Xidian University [2]Shanghai Jiao Tong University [3]National University of Singapore
[4]Shanghai Innovation Institute [5]MemTensor (Shanghai) Technology Co., Ltd.
[6]University of Science and Technology of China
[*]Equal contribution, order determined by flipping a coin.
[†]Corresponding authors: tangb@memtensor.cn, muningwen@sjtu.edu.cn

Abstract

The hallmark of human intelligence is the self-evolving ability to master new skills by learning from past experiences. However, current AI agents struggle to emulate this self-evolution: fine-tuning is computationally expensive and can introduce destabilizing weight updates, increasing the risk of catastrophic forgetting, while existing memory-based methods rely on passive semantic matching that often retrieves noise. To address these challenges, we propose MEMRL, a non-parametric approach that evolves via reinforcement learning on episodic memory. By decoupling stable reasoning from plastic memory, MEMRL employs a Two-Phase Retrieval mechanism to filter noise and identify high-utility strategies through environmental feedback. Extensive experiments on Humanity's Last Exam, BigCodeBench, ALFWorld, and Lifelong Agent Bench demonstrate that MEMRL consistently outperforms strong baselines, suggesting that MEMRL offers a practical way to balance stability and plasticity during runtime improvement. Code is available at https://github.com/MemTensor/MemRL.

1 Introduction

Cognitive theories describe how humans balance stability and plasticity: complementary learning systems pair slowly consolidated knowledge with fast episodic memory [15, 31, 22], and constructive episodic simulation recombines elements of past episodes to prepare for new situations, supporting adaptation without rewiring neural circuitry [44, 16, 45, 14]. Despite their reasoning capabilities, current AI agents struggle to emulate this decoupled self-evolution [62, 68, 46, 59]. Specifically, fine-tuning internalizes experience by modifying weights [35, 52, 41, 8] but incurs computational cost and risks catastrophic forgetting [21, 24, 64]. Conversely, Retrieval-Augmented Generation (RAG) [23] provides a non-parametric alternative but remains passive, retrieving by semantic similarity rather than helpfulness [20, 13]; this prevents agents from effectively leveraging runtime feedback to choose helpful strategies.

This limitation underscores a critical research question: **How can we enable an agent to continuously improve its performance after deployment, without compromising the stability of its pre-trained backbone?** Our objective is to achieve an agent that evolves with continued usage and rapidly adapts to new tasks after deployment, referred to as Runtime Continuous Learning [18, 50, 37, 64], all while keeping the backbone model frozen to avoid the risk of catastrophic forgetting [10, 63]. To achieve this objective, we propose **MEMRL**, an approach that facilitates self-evolving agents by explicitly decoupling the model's stable reasoning from dynamic episodic memory. The cognitive

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

theories above serve only as high-level inspiration for this architecture; MEMRL does not model the underlying human processes. Following prior non-parametric work [74, 60, 66, 63, 26, 72], we use *self-evolution* to mean memory-level adaptation of the composite agent: the LLM stays frozen, a lightweight RL update of memory utilities is the learning mechanism, and the memory bank and retrieval policy are where adaptation takes place. Drawing on reinforcement learning to estimate expected utilities from trial-and-error feedback [56], we formalize the interaction between the frozen LLM and external memory as a Markov Decision Process (MDP) [40]. Unlike traditional methods that optimize the backbone model, MEMRL only optimizes the retrieval policy of memory.

MEMRL organizes memory into a structured **Intent-Experience-Utility** triplet. This structure transforms retrieval from a passive semantic match task into an active decision-making process: **Two-Phase Retrieval** first uses semantic similarity for recall, then uses learned Q-values (predicted utility) to re-rank candidates by expected utility, moving beyond similarity alone [61]; **Utility-Driven Update** refines these Q-values through environmental feedback, applying Monte Carlo style updates [32]. This closed-loop cycle enables the agent to identify high-value memories while suppressing similar but unhelpful distractors, learning from both success and failure without weight updates. We validate MEMRL on four diverse benchmarks, including Humanity's Last Exam, BigCodeBench, ALFWorld, and Lifelong Agent Bench. Across three runs, MEMRL attains the highest mean on most metrics, with the largest gains in exploration-heavy environments and parity with the strongest baseline on the near-saturated DB task. Our analysis further shows a strong correlation between learned utility and task success. In summary, our contributions are threefold:

- We introduce a runtime learning framework that stores experiences as Intent–Experience–Utility triplets, enabling frozen-backbone adaptation that addresses the stability-plasticity dilemma.
- We introduce MEMRL, a non-parametric approach enabling agent self-evolution via Two-Phase Retrieval and Utility-Driven Update.
- We conduct extensive evaluations and in-depth analyses to validate MEMRL's effectiveness and explain how its value-aware memory retrieval drives performance gains.

2 Related Works

Runtime Learning. Runtime Learning studies post-deployment improvement, where agents adapt through interaction streams while keeping the backbone model frozen, marking a shift toward the "Era of Experience" [50]. This setting differs from continual learning, which updates parameters over task streams and is often affected by catastrophic forgetting [37, 64], and from test-time training or adaptation, which updates parameters or internal statistics at inference to handle distribution shifts [53, 58, 28]. With the backbone frozen, the central question is no longer how to update the model, but how to store and reuse experience. Recent lifelong-agent benchmarks and self-evolving memory systems highlight this direction [75, 63, 77], but reveal a central bottleneck: as memory grows, agents must distinguish useful experience from semantically similar but unhelpful noise.

Memory-Augmented LLM Agents. A first line of work asks how interaction experience can be *stored and organized* for reuse. Building on retrieval-augmented generation [23, 20, 1], reflective and experiential agents store natural-language reflections, cross-task insights, or context-aware guidelines distilled from past trajectories [48, 74, 11], while skill-memory and procedural-memory systems accumulate executable skills or trajectory-distilled scripts as structured retrieval-time representations [59, 9]. Sequential-decision agents further retrieve trajectory-level demonstrations to guide future actions [78, 19, 60, 1, 30]. More recent infrastructures systematize the memory lifecycle—storage, indexing, consolidation, and update—through hierarchical or graph-structured designs [36, 5, 27, 66, 17, 69, 43, 73, 25]. These works clarify what agents can store and how memory can be structured. MEMRL is complementary: it focuses on how environmental feedback assigns utility to stored experiences and decides which ones to reuse at retrieval time.

Memory-based Agentic Reinforcement Learning. Feedback-driven memory use traces back to episodic control, which performs non-parametric value lookup over high-return episodes [2, 39, 29]. Memento formalizes memory-augmented adaptation as a Memory-augmented MDP (M-MDP) [76]. Recent LLM-agent methods combine memory with RL, but usually optimize action selection, memory management, or skill execution: REMEMBERER learns from experience memory [71]; Retrospex

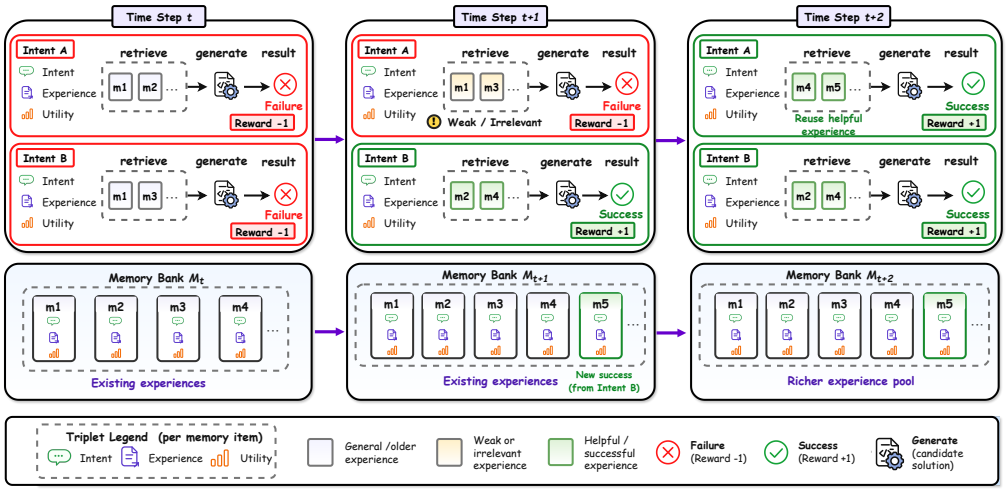


Figure 1: Illustration of memory-augmented decision making under an MDP. At time t, the agent starts with memory set $\mathcal{M}_t$. At $t{+}1$, *Intent A* retrieves past experiences but fails, while *Intent B* succeeds and adds its experience to memory. At $t{+}2$, *Intent A* retrieves this newly stored experience from *Intent B* and succeeds. This example shows how memory retrieval enables knowledge reuse across intents, implicitly supporting self-evolution through shared experiences.

uses a critic to rescore executable actions [65]; Memory-R1 and AgeMem train memory managers via PPO/GRPO [67, 70]; while JitRL, Skill-Pro, and MemSkill modify logits, evolve executable skills, or learn memory-operation skills [26, 33, 72]. These works demonstrate the value of feedback-driven memory, but their learning signals largely act on actions, managers, or skills.

Relationship between MEMRL and Previous Methods. MEMRL inherits Memento's view that treating the retrieval policy as a component of the agent policy, but optimizes the retrieval policy with value-based reinforcement learning. Its closest comparison is REMEMBERER, but it uses the value differently: REMEMBERER prompts LLMs with step-level action–value pairs for action selection, whereas MEMRL applies Q-values for retrieving better in-context examples. MEMRL actively identifies more useful trajectories as transferable experiences to guide agents. MEMRL learns scalar utilities over task-level Intent–Experience–Utility memories without training critics, managers, skill policies, or modifying logits, making experience reuse selective and outcome-aware while keeping the LLM frozen. JitRL [26] is the closest concurrent work: both methods freeze the backbone and learn from runtime feedback without gradient updates, but they act at different points. JitRL estimates advantages over a finite set of candidate actions and adjusts the output logits, as do earlier output-action RL methods such as TWOSOME and GLAM [57, 4]; MEMRL instead learns which experience to place in the input context. JitRL is therefore suited to enumerable action spaces, whereas MEMRL also applies to open-ended generation such as programs, SQL/shell commands, and free-form answers (controlled comparison in Appendix G.2).

3 Problem Formulation

Standard RAG and memory-based agentic methods typically retrieve memories through fixed semantic similarity. While semantic matching can identify relevant experiences, relevance alone does not guarantee utility: a memory may be close to the current query but still lead to a low-reward outcome. To reconcile this gap, we adopt the Memory-Based Markov Decision Process (M-MDP) framework [76], where memory retrieval is treated as a decision variable rather than a passive lookup. Figure 1 illustrates how retrieval outcomes and memory evolution unfold over time.

3.1 Memory-Augmented Agent Policy

The M-MDP framework [76] is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma, \mathcal{M})$. Here, $\mathcal{S}$ and $\mathcal{A}$ represent state and action spaces, P is the transition dynamics, $\mathcal{R}$ is the reward function of state and action,

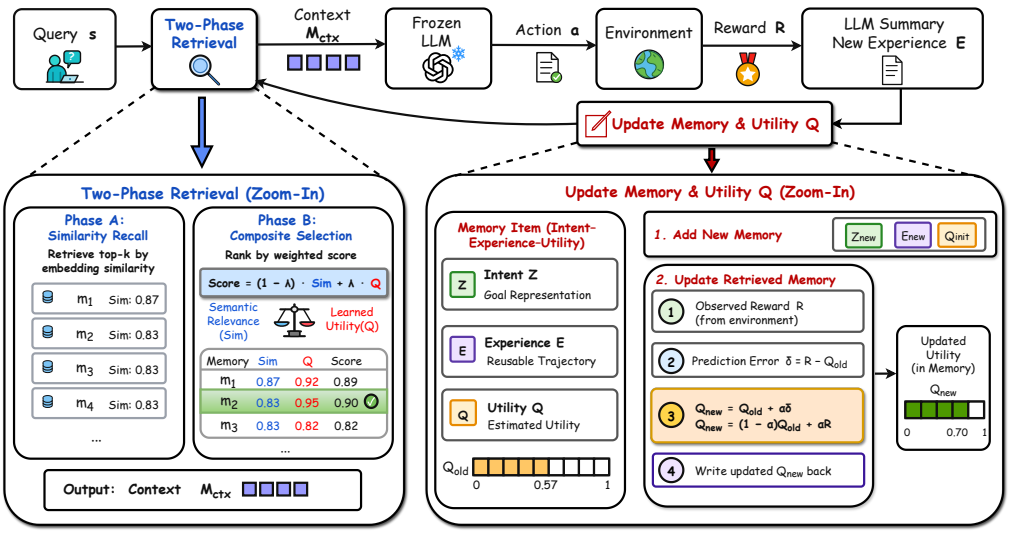


Figure 2: **Overview of MEMRL. The end-to-end learning loop (Top)**: given a query $\mathbf{s}$, the agent retrieves context $\mathbf{M}_{ctx}$ from memory bank $\mathcal{M}$, generates output $\mathbf{y}$, and updates memory value Q based on reward R. **Two-Phase Retrieval (Bottom Left)**: Candidates are recalled via similarity, then re-ranked using learned Q-values. **Update Memory and Utility (Bottom Right)**: MEMRL represents memories as Intent–Experience–Utility triplets, writes new triplets into the memory bank, and updates the utilities of retrieved memories using environmental rewards.

$\gamma \in [0, 1)$ denotes the discount factor, and $\mathcal{M} = (\mathcal{S} \times \mathcal{A} \times \mathbb{R})^*$ constitutes the evolving memory bank of past experiences [76]. At each step t, the agent receives state s_t and leverages $\mathcal{M}_t$ to generate a response a_t maximizing the expected reward. The joint policy $\pi(a_t|s_t, \mathcal{M}_t)$ is formulated as the marginal probability over retrieved items m [76]:

$$\pi(a_t|s_t, \mathcal{M}_t) = \sum_{m \in \mathcal{M}_t} \mu(m|s_t, \mathcal{M}_t) p_{LLM}(a_t|s_t, m), \tag{1}$$

where $\mu(m|s_t, \mathcal{M}_t)$ is the **Retrieval Policy** for selecting memory contexts, and $p_{LLM}(a_t|s_t, m)$ is the **Inference Policy** parameterized by a frozen LLM. This decomposition makes retrieval an explicit decision variable: changing μ changes which memory conditions the frozen generator, and therefore changes the expected outcome.

3.2 Retrieval as a Value-Based Decision Problem

Building on this decision-theoretic view, we formulate memory retrieval as a value-based decision problem. We instantiate the state s as the user intent, represented by the current query, and the retrieval action as selecting a memory item $m \in \mathcal{M}_t$. This maps the M-MDP components to a structured *Intent–Experience–Utility* view: each memory is associated with the intent it addresses, the experience it stores, and the utility it provides for future retrieval.

While the executable action a is generated by the frozen LLM, its quality is conditioned on the retrieved memory. We therefore define $Q(s, m)$ as the expected utility of conditioning the generator on memory m under intent s. Conceptually, the optimal retrieval policy selects memories with the highest expected utility:

$$\mu^*(m|s, \mathcal{M}) = \arg\max_{m \in \mathcal{M}} Q(s, m). \tag{2}$$

In this view, the Q-value acts as a critic for the retrieval mechanism, distinguishing memories that strategically aid the generator from unhelpful noise that merely shares high semantic similarity.

4 MEMRL

Building upon the M-MDP formulation in Section 3, we propose MEMRL, a non-parametric framework that learns how to retrieve useful episodic memories while keeping the LLM backbone frozen.

Instead of modifying the model weights θ, MEMRL optimizes the **retrieval policy** $\mu(m|s, \mathcal{M})$ within an evolving memory space. As illustrated in Figure 2, the framework consists of three core components: (i) a structured **Intent-Experience-Utility** memory bank, (ii) a **Two-Phase Retrieval** mechanism that decouples semantic recall from value-aware selection, and (iii) a **Runtime Utility Update** rule that stabilizes Q-value estimation.

4.1 The Intent-Experience-Utility Triplet

To support value-based decision-making, we structure the external memory bank $\mathcal{M}$ not merely as key-value pairs, but as a set of triplets:

$$\mathcal{M} = \{(z_i, e_i, Q_i)\}_{i=1}^{|\mathcal{M}|}, \tag{3}$$

where z_i represents the **Intent**, e_i stores the raw **Experience** (e.g., a successful solution trace or trajectory), and Q_i denotes the learned **Utility**. Q_i approximates the expected return of applying experience e_i to intents similar to z_i, serving as the *critic* in RL.

4.2 From Semantic Recall to Value-Aware Selection

Standard RAG assumes "similar implies useful," but agentic tasks often involve environment-specific routines with poor generalization [51, 6, 12, 78]. Thus, MEMRL implements a **Two-Phase Retrieval** strategy. **Phase A: Similarity-Based Recall.** Given query s, we isolate a candidate pool $\mathcal{C}(s)$ of relevant experiences by filtering memory bank $\mathcal{M}$ via cosine similarity and a sparsity threshold δ:

$$\mathcal{C}(s) = \text{TopK}_{k_1}(\{i|sim(Emb(s), Emb(z_i)) > \delta\}), \tag{4}$$

where Emb represents the Embedding Model to transfer the raw text to a vector. If $\mathcal{C}(s) = \emptyset$, MEMRL relies solely on the frozen LLM without retrieved memory. **Phase B: Value-Aware Selection.** To determine the final context $\mathcal{M}_{ctx}(s)$, we select top-k_2 items from $\mathcal{C}(s)$ using a composite score balancing semantic relevance (similarity) and learned utility (Q):

$$\text{score}(s, z_i, e_i) = (1 - \lambda) \cdot \hat{sim}(Emb(s), Emb(z_i)) + \lambda \cdot \hat{Q}_i, \tag{5}$$

where $\hat{\cdot}$ denotes z-score normalization and $\lambda \in [0, 1]$ modulates the trade-off. Similarity acts as a relevance prior rather than an exploration mechanism; we do not use explicit exploration such as ϵ-greedy. This mechanism filters out "distractor" memories—those semantically similar but with low historical utility. As detailed in Section 5.4.2, normalization and strict similarity thresholds are essential for noise filtering and maintaining stability during self-evolution.

4.3 Non-Parametric RL on Memory

The core of MEMRL is to continuously refine memory utilities from environmental feedback. During runtime, the agent retrieves context $\mathcal{M}_{\text{ctx}}(s)$, generates an action a with the frozen LLM policy in Eq. 1, and receives a task-level reward r after execution. In the general M-MDP formulation, a retrieved memory utility can be updated by a TD rule [55]:

$$Q(s, m) \leftarrow Q(s, m) + \alpha[r + \gamma \max Q(s', m') - Q(s, m)], \tag{6}$$

whereas our implementation observes task-level feedback and uses a Monte Carlo-style update:

$$Q_{\text{new}} \leftarrow Q_{\text{old}} + \alpha\big(r - Q_{\text{old}}\big). \tag{7}$$

Here, α is the learning rate. This update is a terminal one-step instantiation of Eq. 6: the next state is treated as terminal to balance complexity of tracing multi-step memory retrieval sequence. Under task-level feedback, it also resembles contextual-bandit learning: the intent provides the context, candidate memories are the selectable items, and feedback updates the selected memory's utility. Thus, the M-MDP defines retrieval as part of the full memory-augmented agent policy, while Eq. 7 provides a lightweight memory utility update.

For the memories actually injected into $\mathcal{M}_{\text{ctx}}(s)$, MEMRL updates their utilities using Eq. 7, pushing each Q_i toward the empirical return of using experience e_i under similar intents. After each trajectory, an LLM summarizes the interaction into a new experience [9], which is written back as a new triplet $(z, e_{\text{new}}, Q_{\text{init}})$. Learning therefore occurs entirely in memory space, enabling continual experience accumulation while keeping the LLM parameters unchanged. We set $Q_{\text{init}} = 0$, which encodes *unverified* rather than low utility: Phase A admits a new memory by similarity alone, and the conservative initialization keeps near-duplicate recent memories from displacing memories whose utility has already been validated (Appendix G.3).

Table 1: **Runtime Learning results.** We compare MEMRL against various baselines over 10 epochs. The results are reported as **Last Epoch Success Rate / Cumulative Success Rate (CSR)**. For No Memory, RAG, MemP, and MEMRL on BigCodeBench, Lifelong Agent Bench, and ALFWorld, entries are means over three runs; standard deviations are reported in Table 15 (Appendix G.1). All other entries, including HLE, are single runs. **Average** is the mean over all benchmarks.

Method	BigCodeBench	Lifelong Agent Bench		ALFWorld	HLE	Average
	Code Gen (Last / CSR)	OS Task (Last / CSR)	DB Task (Last / CSR)	Exploration (Last / CSR)	Knowledge Frontier (Last / CSR)	(Last / CSR)
Model	`GPT-4o`	`GPT-4o-mini`	`GPT-4o-mini`	`GPT-5-mini`	`Gemini-3-pro`	-
No Memory	0.481	0.649	0.863	0.777	0.357	0.625
Pass@10	– / 0.577	– / 0.756	– / 0.928	– / 0.928	– / 0.524	– / 0.743
RAG	0.458 / 0.468	0.665 / 0.690	0.916 / 0.917	0.887 / 0.940	0.430 / 0.475	0.671 / 0.698
Self-RAG	0.497 / 0.561	0.646 / 0.732	0.891 / 0.898	0.907 / 0.962	0.423 / 0.475	0.673 / 0.726
Mem0	0.487 / 0.495	0.670 / 0.702	0.920 / 0.926	0.894 / 0.969	0.436 / 0.470	0.681 / 0.712
MemP	0.582 / 0.599	0.752 / 0.776	**0.961** / 0.967	0.884 / 0.922	0.522 / 0.570	0.740 / 0.767
MEMRL	**0.596 / 0.624**	**0.777 / 0.807**	**0.961 / 0.969**	**0.942 / 0.983**	**0.570 / 0.606**	**0.769 / 0.798**

4.4 Theoretical Stability Analysis

We analyze MEMRL's stability in Appendix A. Under a frozen inference policy and stationary task distribution, Eq. 7 provides unbiased and variance-bounded utility estimation for fixed query-memory pairs. At the global level, each memory utility tracks its retrieval-weighted average return over the Phase-A queries that select it. We further view MEMRL as a GEM-style process that maximizes a KL-regularized retrieval objective,

$$\mathcal{J}(\mu, Q) = \mathbb{E}_s\Big[\sum_{m\in\mathcal{C}(s)} \mu(m|s)\, Q(s,m) - \tfrac{1}{\beta}\, D_{\mathrm{KL}}\big(\mu(\cdot|s) \,\|\, \pi_{\mathrm{sim}}(\cdot|s)\big)\Big]. \tag{8}$$

Here, π_{sim} is the semantic prior induced by Phase A and β controls the regularization strength. For fixed Q, the maximizer is $\mu^*(m|s) \propto \pi_{\mathrm{sim}}(m|s)\exp(\beta Q(s,m))$, so $\log \mu^*$ is a weighted sum of a similarity term and a utility term; Eq. 5 is a normalized linear approximation of this policy. The KL term keeps retrieval within the semantically relevant support and limits the influence of noisy early Q estimates. Phase-B re-ranking corresponds to the policy-improvement step and Eq. 7 to the value-estimation step (derivation in Appendix A.4). This provides a theoretical justification for stable memory-side learning and is consistent with the empirical reduction in forgetting observed in Section 5.4.2.

5 Experiments

5.1 Experimental Setup

Baselines & Benchmarks. We compare MEMRL against RAG-based (RAG, Self-RAG), Agentic Memory (Mem0, MemP), and Test-Time Scaling (Pass@k) baselines under a frozen-backbone setting (see Appendix D.1). Evaluations span four domains: BigCodeBench (coding), ALFWorld (navigation), LifelongAgent Bench (OS/DB), and Humanity's Last Exam (HLE). Details are in Appendix D.2. Backbones are selected per benchmark to avoid no-signal or ceiling problems, ensuring valid learning signals, while Appendix E.4 provides a unified comparison to demonstrate cross-task consistency under identical capacity.

Metrics. We employ two metrics: (1) **Success Rate (SR)**, the ratio of tasks completed in an epoch; (2) **Cumulative Success Rate (CSR)**, the proportion of tasks solved at least once across epochs.

We evaluate our MEMRL and baselines under two distinct settings: **Runtime Learning**, which assesses the ability to learn and adapt within a training session, and **Transfer Learning**, which evaluates the generalization capability of memory on unseen tasks. Implementation and reproducibility details, including all prompts used in our experiments, can be found in Appendix E and Appendix K.

Table 2: **Transfer Learning results** on BigCodeBench, Lifelong Agent Bench and ALFWorld. We compare MEMRL against various baselines using the best validation results. Multi-run entries follow Table 1 (means over three runs; standard deviations in Table 15). The **Average** column represents the mean Success Rate across all benchmarks.

Method	BigCodeBench	Lifelong Agent Bench		ALFWorld	Average
	Code Generation (Success Rate)	OS Task (Success Rate)	DB Task (Success Rate)	Exploration (Success Rate)	(Success Rate)
Model	`GPT-4o`	`GPT-4o-mini`	`GPT-4o-mini`	`GPT-5-mini`	-
No Memory	0.482	0.671	0.894	0.824	0.718
RAG	0.478	0.701	0.923	0.933	0.759
Self-RAG	**0.500**	0.653	0.881	0.950	0.746
Mem0	0.485	0.686	0.935	0.950	0.764
MemP	0.488	0.716	0.935	0.914	0.763
MEMRL	**0.500**	**0.745**	**0.940**	**0.981**	**0.792**

Table 3: **Ablation on Retrieval Scope** (CSR, single run, seed 42): MEMRL vs. *Single-Task Reflection*.

Setting	BCB	OS	DB	ALFWorld	HLE	Avg.
Single-Task Reflection	0.614	0.714	0.938	0.930	**0.610**	0.761
MEMRL (Cross-Task)	**0.627**	**0.804**	**0.972**	**0.981**	0.606	**0.798**

5.2 Main Results

Runtime Learning Results. As shown in Table 1, MEMRL attains the highest mean Last and CSR on every benchmark except DB Last, where it ties MemP. Averaged over the five benchmarks, it exceeds the strongest baseline (MemP) by $+2.9$ percentage points (pp) in Last and $+3.1$ pp in CSR. The gains are largest in exploration-intensive environments: on ALFWorld, MEMRL exceeds the strongest three-run baseline (RAG) by $+5.5$ pp Last and $+4.3$ pp CSR, and on OS it exceeds MemP by $+2.5$ pp and $+3.1$ pp. On HLE (single run) the margin over MemP is $+3.6$ pp CSR. On the near-saturated DB task, MEMRL and MemP are statistically indistinguishable. These results are consistent with value-aware selection filtering similar but unhelpful memories; further evidence is provided in Appendix B.4, with failure-mode analysis in Appendix B.5. In a controlled ALFWorld comparison, MEMRL also outperforms the concurrent method JitRL [26] (96.7% vs. 76.2%; Appendix G.2).

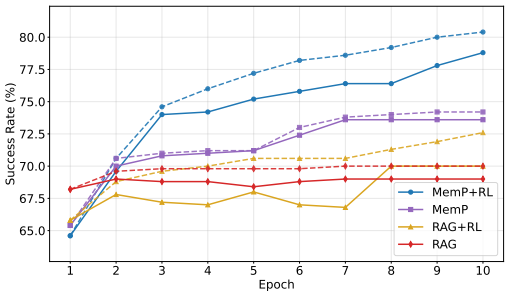
Statistical Reliability. Per-cell standard deviations are reported in Table 15. To quantify reliability beyond them, we macro-average the three-run results over the four multi-run benchmarks and compute approximate 95% confidence intervals for the difference between MEMRL and the strongest baseline, MemP. The differences are $+2.45$ pp $[1.45, 3.44]$ in Last, $+2.96$ pp $[1.09, 4.83]$ in CSR, and $+2.82$ pp $[1.90, 3.75]$ in Transfer SR; all three intervals exclude zero. This supports an aggregate advantage across benchmarks, not a significant gain on every individual benchmark (Appendix G.1).

Transfer Learning Results. We evaluate memory transferability by freezing the memory bank after training and testing on held-out sets. As shown in Table 2, MEMRL attains the highest average transfer success rate, $+2.8$ pp above MemP. The advantage is clearest on ALFWorld ($+4.8$ pp over RAG, the strongest three-run baseline there) and OS ($+2.9$ pp over MemP), whereas on BigCodeBench and DB the mean $\pm$ std intervals of MEMRL and the strongest baselines overlap (Table 15). This suggests that memories selected by learned utility transfer to unseen tasks at least as well as similarity-selected ones, with the largest benefit in structured, multi-step environments.

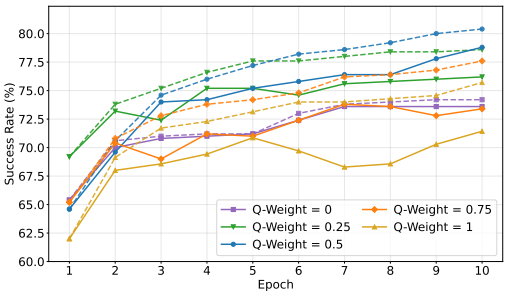
5.3 Ablations

5.3.1 Effectiveness of Runtime RL

To isolate runtime RL, we compare MEMRL and its RAG-based variant against their non-RL counterparts (MemP and standard RAG) in the OS interaction environment. As shown in Figure 3a, while initial performance is comparable, a clear divergence emerges as training progresses: MEMRL achieves a smoother learning curve and superior stability. Crucially, this advantage is most pronounced



(a) **OS Interaction Performance.** Performance comparison of MEMRL vs. baselines (MemP, RAG). **Solid lines** represent the Epoch Success Rate, while **dashed lines** indicate the Cumulative Success Rate (CSR).



(b) **Ablation on Q-value weighting factor λ.** Performance comparison across $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$. **Solid lines** denote Epoch Success Rate, while **dashed lines** indicate Cumulative Success Rate (CSR).

Figure 3: **OS interaction performance and ablation on Q-value weighting.** (a) Comparison between MEMRL and memory-based baselines on OS interaction tasks. (b) Ablation over different Q-value weighting factors λ.

in the Cumulative Success Rate (dashed lines), where the monotonic widening gap indicates that the RL-driven value function effectively filters noisy memories and consolidates successful experiences.

5.3.2 Impact of Q-Value Weighting

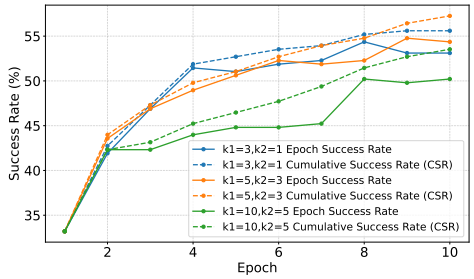
To study the balance between semantic relevance and learned utility, we evaluate the Q-weighting factor $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$. As shown in Figure 3b, performance exhibits a concave trend peaking at $\lambda = 0.5$. Deviating from this balance degrades results: pure semantic retrieval ($\lambda = 0$) plateaus due to an inability to filter unhelpful memories, while excessive RL weight ($\lambda \to 1$), which approximates a standard bandit-style selector relying purely on learned Q-values, induces volatility and context detachment. Thus $\lambda = 0.5$ is the best-performing value among those tested, and we use it as a unified default across benchmarks rather than claiming it is universally optimal; the preferred value may depend on task similarity, memory density and noise, and the reliability of the learned Q-values.

5.3.3 Ablation Analysis: Cross-Task vs. Single-Task Optimization

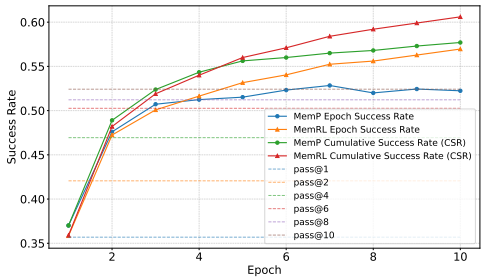
To investigate the source of our performance gains, we conduct an ablation study in Table 3 by restricting the memory retrieval scope. We compare the full MEMRL (*Cross-Task Retrieval*) against an ablated setting that only utilizes feedback from the single task instance, conceptually equivalent to Reflexion [48]. MEMRL demonstrates superior performance in structured environments, particularly on **OS-Agent** (+9.0 pp CSR) and **ALFWorld** (+5.1 pp). These benchmarks exhibit high intra-dataset similarity, allowing MEMRL to effectively perform *horizontal transfer*—retrieving and adapting successful policies from semantically similar historical tasks. While HLE falls into a different regime, the single-task variant (0.610) is comparable to the cross-task MEMRL (0.606). This is expected because HLE has low intra-dataset semantic similarity (0.186; Appendix B.3) and contains diverse, knowledge-intensive questions with few reusable procedural patterns. This contrast shows that MEMRL benefits different task regimes in different ways: it enables cross-task procedural transfer in structured, high-similarity environments, while in low-similarity settings such as HLE, it improves through instance-level utility learning and reuse of high-value experiences for hard questions.

5.3.4 Sensitivity to Retrieval Size (k_1 and k_2).

To investigate retrieval bandwidth, we compare three memory density configurations on the HLE (CS/AI) subset benchmark: sparse ($k_1 = 3, k_2 = 1$), moderate ($k_1 = 5, k_2 = 3$), and dense ($k_1 = 10, k_2 = 5$). As shown in Figure 4a, performance follows an inverted-U trajectory, illustrating the trade-off between information sufficiency and context noise. The sparse setting limits performance due to insufficient guidance, whereas the dense setting degrades success rate by introducing distractions into the reasoning context. Consequently, the moderate configuration ($k_1 = 5, k_2 = 3$) achieves the best result, effectively maximizing the signal-to-noise ratio of the retrieved context.

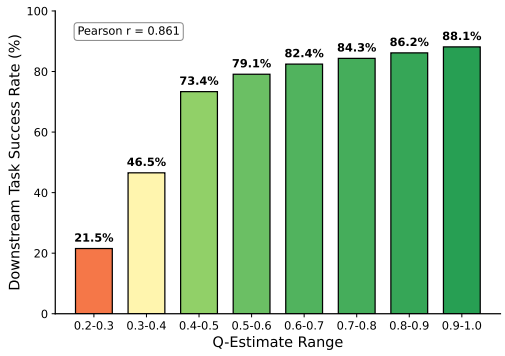


(a) **Ablation on Retrieval Size (k_1, k_2).** Performance comparison on the HLE (CS/AI) subset across sparse, moderate, and dense settings.

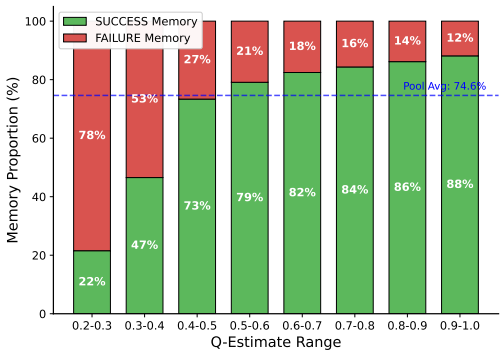


(b) **Runtime Learning curve on HLE.** Epoch Success Rate and Cumulative Success Rate (CSR) of MEMRL and MemP in HLE.

Figure 4: **Retrieval-size ablation and performance comparison on HLE.** (a) Ablation on retrieval size (k_1, k_2) over sparse, moderate, and dense settings. (b) Epoch Success Rate and Cumulative Success Rate (CSR) comparison between MEMRL and MemP on HLE.



(a) Success Rate vs. Q-Range



(b) Memory Composition

Figure 5: **Q-Value Analysis.** (a) Pearson $r = 0.861$ indicates Critic's predictive power. (b) Failure memories ($\sim 12\%$) in high Q-bins indicate latent strategic utility.

5.4 Discussion

In this section, we delve deeper into the mechanisms driving MEMRL's performance, connecting empirical results to the challenge of balancing knowledge retention and adaptation.

5.4.1 Predictive Power of the Q Critic

As shown in Figure 5a, learned Q-values strongly correlate (Pearson $r = 0.861$) with empirical task success rates, rising from 21.5% in the lowest-confidence bin to 88.1% in the highest, which indicates the Critic's ability to effectively rank memories by success likelihood. Beyond simple ranking, memory composition analysis (Figure 5b) reveals that the agent retains a small fraction of "failure" memories ($\sim 12\%$) even in high-Q bins ($0.9 - 1.0$), suggesting that Q-values capture *utility beyond binary outcomes* by recognizing strategically useful near-misses. We further substantiate this with concrete case studies in Appendix J. This indicates that the Critic prioritizes reusable guidance—including transferable procedural lessons from high-utility failures—rather than merely separating success from failure, thereby offering greater robustness than simple success-replay mechanisms.

5.4.2 Stability of MEMRL

We analyze MEMRL through the lens of the *stability-plasticity dilemma*. The superior CSR (Table 1) indicates that MEMRL effectively expands the solution space, enabling the agent to break through local optima. Furthermore, long-term dynamics (Figure 4b) reveal a critical stability advantage: heuristic methods like MemP show stronger empirical forgetting, evidenced by a widening gap between CSR and current Success Rate, whereas MEMRL maintains more synchronized growth. The theoretical stability analysis in Section 4.4 shows that, under fixed-policy and stationary-reward

assumptions, the scalar utility estimator has bounded variance and converges in expectation for a fixed state-memory pair. Empirically, this contributes to more stable retrieval behavior.

We quantitatively validate these insights using the **Forgetting Rate**, defined as $FR = N_{\text{lost}}/N_{\text{fail}}$, where N_{lost} denotes tasks transitioning from previous success to current failure and N_{fail} is the total number of failures in the current epoch (see Figure 6 in Appendix B.1 for the full trajectory). MEMRL achieves the lowest mean forgetting rate (0.041), outperforming MemP (0.051). Additionally, ablation results demonstrate that removing z-score normalization and similarity gating causes the rate to spike to 0.073. This suggests that strict filtering helps manage utility variance, suppresses noisy memory retrieval, and supports stable self-evolution across long interaction streams.

5.4.3 Extended Analysis.

We further analyze the mechanisms and generalization behavior of MEMRL beyond the main results. Appendix B.2 shows that MEMRL acts as a structural trajectory verifier in multi-step tasks, where retrieved memories must remain valid across an entire action sequence rather than merely match the initial instruction. Appendix B.3 also analyzes how task similarity shapes memory efficacy: in structured domains, MEMRL benefits from pattern generalization across similar tasks, while in more diverse domains it supports specific knowledge acquisition through repeated runtime exposure. Appendix C further evaluates cross-model memory transfer and modular multi-task memory merging, showing that learned memory utilities can transfer across backbones and compose across task domains with limited interference. Appendix F analyzes cost, efficiency, and scalability, confirming that these gains are achieved with lightweight retrieval and update overhead. Appendices G.3–G.5 cover new-memory initialization, combination with fine-tuning, and non-language agents.

6 Limitations and Conclusion

Limitations and Future Work. While MEMRL enables non-parametric runtime evolution, several limitations remain. First, its step-wise utility update may introduce noise in long-horizon trajectories, motivating future multi-step updates and memory consolidation. Second, updating multiple retrieved memories creates credit-assignment ambiguity, calling for more precise attribution methods such as Shapley-based analysis [47] or value decomposition [42, 54]. Third, when task similarity is low, memory transfer becomes less effective, suggesting the need for richer experience bases and hierarchical abstraction. Fourth, retrieval-only adaptation has an inherent ceiling set by the frozen model, its context length, and retrieval quality; MEMRL is one component of a self-improving agent rather than a substitute for parametric learning, and a preliminary experiment suggests the two are complementary (Appendix G.4). Fifth, our statistical evidence is limited: three-run statistics cover four methods on four benchmarks, while the other baselines and HLE (stochastic decoding at temperature 0.6) are single runs because repeated full-scale runs exceeded our API budget. Finally, the sensitivity to Q_{init} was tested only on DB; its effect in low-similarity settings such as HLE, where new memories may receive less exposure, remains to be measured. Additional discussions on memory security, dedicated domains, and multi-agent sharing are provided in Appendix H.

Conclusion. We presented MEMRL, a non-parametric framework that formulates memory retrieval as a value-based decision process. Combining Intent–Experience–Utility memories with Monte Carlo-style utility updates, MEMRL lets frozen LLM agents adapt at the memory level: it filters similar but unhelpful memories and reuses high-utility experiences without weight updates. Experiments across diverse benchmarks show consistent gains in runtime adaptation and generalization, supporting episodic memory as a practical substrate for stable continual agent learning.

Acknowledgments and Disclosure of Funding

The Shanghai Jiao Tong University team is supported by the National Key R&D Program of China (2024YFC3505402). This work is also supported by the National Natural Science Foundation of China (62322603) and by the Shanghai Municipal Science and Technology Major Project under Grant 2025SHZDZX025D08.

References

[1] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection, 2023.

[2] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

[3] Weihao Bo, Shan Zhang, Yanpeng Sun, Jingjing Wu, Qunyi Xie, Xiao Tan, Kunbin Chen, Wei He, Xiaofan Li, Na Zhao, Jingdong Wang, and Zechao Li. Agentic learner with grow-and-refine multimodal semantic memory. *arXiv preprint arXiv:2511.21678*, 2025.

[4] Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 3676–3713, 2023.

[5] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025.

[6] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR 2024, page 719–729. ACM, July 2024.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[8] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

[9] Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. *arXiv preprint arXiv:2508.06433*, 2025.

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[11] Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Advances in Neural Information Processing Systems*, 37:119919–119948, 2024.

[12] Chunjing Gan, Dan Yang, Binbin Hu, Hanxiao Zhang, Siyuan Li, Ziqi Liu, Yue Shen, Lin Ju, Zhiqiang Zhang, Jinjie Gu, Lei Liang, and Jun Zhou. Similarity is not all you need: Endowing retrieval augmented generation with multi layered thoughts. *arXiv preprint arXiv:2405.19893*, 2024.

[13] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024.

[14] Mary L Gick and Keith J Holyoak. Analogical problem solving. *Cognitive psychology*, 12(3):306–355, 1980.

[15] Stephen Grossberg. Adaptive resonance theory: How a brain learns to consciously attend, learn, and recognize a changing world. *Neural networks*, 37:1–47, 2013.

[16] Demis Hassabis and Eleanor A Maguire. Deconstructing episodic memory with construction. *Trends in cognitive sciences*, 11(7):299–306, 2007.

[17] Zhengjun Huang, Zhoujin Tian, Qintian Guo, Fangyuan Zhang, Yingli Zhou, Di Jiang, and Xiaofang Zhou. Licomemory: Lightweight and cognitive agentic memory for efficient long-term reasoning. *arXiv preprint arXiv:2511.01448*, 2025.

[18] Khurram Javed, Haseeb Shah, Rich Sutton, and Martha White. Online real-time recurrent learning using sparse connections and selective learning. *Journal of Machine Learning Research*, 2023.

[19] Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *arXiv preprint arXiv:2402.03610*, 2024.

[20] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781, 2020.

[21] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[22] Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

[23] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[24] Hongyu Li, Liang Ding, Meng Fang, and Dacheng Tao. Revisiting catastrophic forgetting in large language model tuning. *arXiv preprint arXiv:2406.04836*, 2024.

[25] Leqian Li, Dianxi Shi, Jialu Zhou, Xinyu Wei, Mingyue Yang, Songchang Jin, and Shaowu Yang. Retrieval feedback memory enhancement large model retrieval generation method. *arXiv preprint arXiv:2508.17862*, 2025.

[26] Yibo Li, Zijie Lin, Ailin Deng, Xuan Zhang, Yufei He, Shuo Ji, Tri Cao, and Bryan Hooi. Just-in-time reinforcement learning: Continual learning in llm agents without gradient updates. *arXiv preprint arXiv:2601.18510*, 2026.

[27] Zhiyu Li, Shichao Song, Hanyu Wang, Simin Niu, Ding Chen, Jiawei Yang, Chenyang Xi, Huayi Lai, Jihao Zhao, Yezhaohui Wang, et al. Memos: An operating system for memory-augmented generation (mag) in large language models. *arXiv preprint arXiv:2505.22101*, 2025.

[28] Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

[29] Zichuan Lin, Tianqi Zhao, Guangwen Yang, and Lintao Zhang. Episodic memory deep q-networks. *arXiv preprint arXiv:1805.07603*, 2018.

[30] Yitao Liu, Chenglei Si, Karthik R Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14179–14198, 2025.

[31] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419–457, 1995.

[32] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[33] Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Procmem: Learning reusable procedural memory from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*, 2026.

[34] Radford M. Neal and Geoffrey E. Hinton. *A View of the Em Algorithm that Justifies Incremental, Sparse, and other Variants*, pages 355–368. Springer Netherlands, Dordrecht, 1998.

[35] 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.

[36] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024.

[37] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

[38] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam, 2025.

[39] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International conference on machine learning*, pages 2827–2836. PMLR, 2017.

[40] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[41] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[42] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

[43] Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. Meminsight: Autonomous memory augmentation for llm agents. *arXiv preprint arXiv:2503.21760*, 2025.

[44] Daniel L Schacter and Donna Rose Addis. On the constructive episodic simulation of past and future events. *Behavioral and Brain Sciences*, 30(3):331–332, 2007.

[45] Daniel L Schacter, Donna Rose Addis, Demis Hassabis, Victoria C Martin, R Nathan Spreng, and Karl K Szpunar. The future of memory: remembering, imagining, and the brain. *Neuron*, 76(4):677–694, 2012.

[46] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

[47] Lloyd S Shapley et al. A value for n-person games. 1953.

[48] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

[49] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning, 2021.

[50] David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1, 2025.

[51] Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. *arXiv preprint arXiv:2505.01441*, 2025.

[52] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

[53] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pages 9229–9248. PMLR, 2020.

[54] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[55] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[56] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018.

[57] Weihao Tan, Wentao Zhang, Shanqi Liu, Longtao Zheng, Xinrun Wang, and Bo An. True knowledge comes from practice: Aligning LLMs with embodied environments via reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024.

[58] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020.

[59] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[60] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[61] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[62] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[63] Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

[64] Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. Continual learning for large language models: A survey, 2024.

[65] Yufei Xiang, Yiqun Shen, Yeqin Zhang, and Nguyen Cam-Tu. Retrospex: Language agent meets offline reinforcement learning critic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 4650–4666, 2024.

[66] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[67] Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z Pan, et al. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. *arXiv preprint arXiv:2508.19828*, 2025.

[68] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[69] Ye Ye. Task memory engine: Spatial memory for robust multi-step llm agents. *arXiv preprint arXiv:2505.19436*, 2025.

[70] Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885*, 2026.

[71] Danyang Zhang, Lu Chen, Situo Zhang, Hongshen Xu, Zihan Zhao, and Kai Yu. Large language models are semi-parametric reinforcement learning agents. *Advances in Neural Information Processing Systems*, 36:78227–78239, 2023.

[72] Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026.

[73] Zeyu Zhang, Quanyu Dai, Rui Li, Xiaohe Bo, Xu Chen, and Zhenhua Dong. Learn to memorize: Optimizing llm-based agents with adaptive memory framework. *arXiv preprint arXiv:2508.16629*, 2025.

[74] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642, 2024.

[75] Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating llm agents as lifelong learners, 2025.

[76] Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, et al. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

[77] Ruiwen Zhou, Wenyue Hua, Liangming Pan, Sitao Cheng, Xiaobao Wu, En Yu, and William Yang Wang. Rulearena: A benchmark for rule-guided reasoning with llms in real-world scenarios. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025.

[78] Ruiwen Zhou, Yingxuan Yang, Muning Wen, Ying Wen, Wenhao Wang, Chunling Xi, Guoqiang Xu, Yong Yu, and Weinan Zhang. TRAD: Enhancing llm agents with step-wise thought retrieval and aligned decision. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 3–13. ACM, 2024.

[79] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, Binyuan Hui, Niklas Muennighoff, David Lo, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions, 2025.

A Theoretical Analysis and Proofs

A.1 Stability Analysis

We analyze the stability of MEMRL from a reinforcement learning perspective, focusing on the convergence behavior of the utility estimates stored in memory. MEMRL performs non-parametric runtime learning using a constant-step-size update. We show that under stated fixed-policy and stationary-distribution assumptions, the learned utility values converge in expectation for fixed state-memory pairs and remain variance-bounded.

Setup. At each time step t, the agent observes an intent state s_t, retrieves a memory item $m_t \in \mathcal{M}_t$, generates an output a_t, and receives a scalar reward $r_t \in [-1, 1]$ indicating task success or failure. The generation policy follows the decomposition defined in Eq. 1, where μ denotes the retrieval policy and p_{LLM} is a frozen inference policy.

For each retrieved memory, MEMRL updates its utility using the exponential moving average rule as formulated in Eq. 7, with learning rate $\alpha \in (0, 1]$. For clarity in this analysis, we consider a fixed state–memory pair (s, m) and write $Q_t \equiv Q_t(s, m)$.

Stationary Reward Assumption. We analyze the learning process on a fixed dataset and posit two key conditions that ensure the stability of the environment:

1. **Frozen Inference Policy.** The parameters of $p_{\mathrm{LLM}}(y|s, m)$ and the evaluator's criteria are fixed.
2. **Fixed Task Distribution.** Tasks s are drawn from a stationary distribution over a fixed dataset.

These assumptions guarantee that the learning target is well-defined: the expected reward for any specific task-memory pair is time-invariant. Thus, we have:

$$\mathbb{E}[r_t|s_t = s,\ m_t = m] = \beta(s, m). \tag{9}$$

where the expectation is taken over the stochastic rewards induced by the inference policy p_{LLM}.

Expected Convergence of Utility Estimates. We now state the main stability result.

Theorem A.1. *Let $\{Q_t\}$ be updated according to the rule in Eq. 7 with constant step size $\alpha \in (0, 1]$. If Eq. 9 holds and the pair (s, m) is updated infinitely often, then:*

$$\lim_{t\to\infty} \mathbb{E}[Q_t] = \mathbb{E}[r_t|s_t = s,\ m_t = m] = \beta(s, m). \tag{10}$$

Moreover, the convergence rate is exponential [56]:

$$\mathbb{E}[Q_t] - \beta(s, m) = (1 - \alpha)^t \big(Q_0 - \beta(s, m)\big). \tag{11}$$

Proof. Define the estimation error $e_t \triangleq Q_t - \beta(s, m)$. Based on the update rule in Eq. 7, the error recurrence relation is:

$$e_{t+1} = (1 - \alpha)e_t + \alpha\big(r_t - \beta(s, m)\big).$$

Taking conditional expectation given the history $\mathcal{F}_t$ and using Eq. 9, we obtain:

$$\mathbb{E}[e_{t+1}|\mathcal{F}_t] = (1 - \alpha)e_t.$$

Taking full expectation yields:

$$\mathbb{E}[e_{t+1}] = (1 - \alpha)\mathbb{E}[e_t].$$

Iterating the recursion gives $\mathbb{E}[e_t] = (1 - \alpha)^t e_0$, which converges to zero as $t \to \infty$. □

We provide the detailed derivation of the convergence proof in Appendix A.2.

Bounded Variance and Stability. If the reward variance $\mathrm{Var}(r_t|s, m) < \infty$, then the variance of Q_t remains bounded:

$$\limsup_{t\to\infty} \mathrm{Var}(Q_t) \le \frac{\alpha}{2 - \alpha} \mathrm{Var}(r_t|s, m). \tag{12}$$

Thus, constant-step-size updates do not induce unbounded oscillations; instead, they yield stable utility estimates that track expected memory effectiveness while filtering high-frequency noise. We explicitly derive the variance bounds to characterize the stability of the estimator under task clustering in Appendix A.3.

Global Stability via EM Convergence. The stability of the local estimate (Theorem A.1) extends to the global memory utility $Q(m)$. By the linearity of expectation, $Q(m)$ acts as a Monte Carlo integrator striving to converge to:

$$\lim_{t\to\infty} \mathbb{E}[Q_t(m)] = \mathbb{E}[r|m] = \sum_{s\in\mathcal{S}(m)} \underbrace{\mathbb{E}[r|s,m]}_{\text{Stationary}} \underbrace{\Pr(s|m)}_{\text{Retrieve-Dependent}} . \tag{13}$$

where $\mathcal{S}(m) \triangleq \{s \in \mathcal{S} \mid \text{sim}(Emb(s), Emb(z_m)) \geq \delta\}$ denotes the *effective support set* of memory m, and z_m is the intent component of $m = (z_m, e_m, Q_m)$.

A theoretical challenge arises here: the weighting term $\Pr(s|m)$ is a latent variable governed by the retrieval policy $\mu(m|s;\mathcal{M})$, which itself shifts as Q-values evolve. To prove convergence despite this dependency, we analyze MEMRL as a **Generalized Expectation-Maximization (GEM)** process [7, 34]. From a variational perspective, the system performs coordinate ascent on a global objective function $\mathcal{J}(Q,\mu)$ (the variational lower bound of expected reward): (i) **E-Step (Policy Improvement):** The Phase-B ranking updates the retrieval policy μ to align with current estimates, monotonically increasing $\mathcal{J}$ with respect to μ; (ii) **M-Step (Value Update):** The utility update (Eq. 7) increases $\mathcal{J}$ with respect to Q. By the *Monotonic Improvement Theorem* [34], this alternating optimization suggests that the system converges to a stationary point when each step sufficiently improves the corresponding objective. At such a stationary point, the retrieval policy stabilizes, making the induced distribution $\Pr(s|m)$ approximately time-invariant. This supports the stability of Eq. 13 and helps explain the empirical reduction in forgetting observed in our experiments. More details can be found in Appendix A.4.

A.2 Proof of Theorem A.1: Convergence of EMA Estimation

We aim to prove that for a fixed task-memory pair (s,m) with a stationary reward distribution, the Q-value estimate $Q_t(s,m)$ converges in expectation to the true mean reward $\beta(s,m)$.

Assumptions.

1. **Stationary Reward.** The reward r_t at step t is drawn from a distribution induced by the stochastic action generation $a \sim p_{LLM}(a_t|s_t,m)$, with a constant mean $\beta(s,m) = \mathbb{E}[r_t|s,m]$ and finite variance σ^2.
2. **Update Rule.** The utility is updated via the linear EMA rule with learning rate $\alpha \in (0,1)$:

$$Q_{t+1} = (1-\alpha)Q_t + \alpha r_t.$$

Derivation of Error Dynamics. Let $e_t \triangleq Q_t - \beta(s,m)$ be the estimation error at time step t. Substituting $Q_t = e_t + \beta(s,m)$ into the update rule:

$$\begin{aligned} e_{t+1} + \beta(s,m) &= (1-\alpha)(e_t + \beta(s,m)) + \alpha r_t \\ e_{t+1} &= (1-\alpha)e_t + (1-\alpha)\beta(s,m) + \alpha r_t - \beta(s,m) \\ e_{t+1} &= (1-\alpha)e_t + \beta(s,m) - \alpha\beta(s,m) - \beta(s,m) + \alpha r_t \\ e_{t+1} &= (1-\alpha)e_t + \alpha(r_t - \beta(s,m)). \end{aligned} \tag{14}$$

Convergence Analysis. We define $\mathcal{F}_t$ as the filtration (history) up to time t. Since the reward r_t depends on the action a_t sampled subsequently from the frozen LLM, taking the conditional expectation of Eq. 14 given $\mathcal{F}_t$:

$$\mathbb{E}[e_{t+1}|\mathcal{F}_t] = (1-\alpha)e_t + \alpha(\underbrace{\mathbb{E}[r_t|\mathcal{F}_t]}_{\beta(s,m)} - \beta(s,m)) = (1-\alpha)e_t.$$

By the Law of Iterated Expectations, taking the full expectation yields:

$$\mathbb{E}[e_{t+1}] = \mathbb{E}[\mathbb{E}[e_{t+1}|\mathcal{F}_t]] = (1-\alpha)\mathbb{E}[e_t].$$

Iterating this recurrence relation from $t = 0$:

$$\mathbb{E}[e_t] = (1-\alpha)^t \mathbb{E}[e_0].$$

Since $0 < \alpha < 1$, we have $|1-\alpha| < 1$. Consequently:

$$\lim_{t\to\infty} \mathbb{E}[e_t] = \mathbb{E}[e_0] \cdot \lim_{t\to\infty} (1-\alpha)^t = 0. \tag{15}$$

This proves that the estimator is unbiased in the limit, i.e., $\lim_{t\to\infty} \mathbb{E}[Q_t] = \beta(s, m)$. □

A.3 Bounded Variance and Estimator Stability

In this section, we provide the formal derivation for the variance bound of the estimator Q_t. We explicitly derive the finite-time variance formula via recursive unrolling and prove its asymptotic convergence, demonstrating how Phase-A clustering contributes to global stability.

Derivation of the Variance Bound. Let $\sigma^2 \triangleq \mathrm{Var}(r_t|s, m)$ be the variance of the reward signal, assumed to be finite. The EMA update rule is given by:

$$Q_{t+1} = (1-\alpha)Q_t + \alpha r_t.$$

Since the reward r_t (current noise) is statistically independent of the current estimate Q_t (which is determined by history $\mathcal{F}_{t-1}$), the variance of the sum is the sum of the variances:

$$\begin{aligned}\mathrm{Var}(Q_{t+1}) &= \mathrm{Var}((1-\alpha)Q_t) + \mathrm{Var}(\alpha r_t) \\ &= (1-\alpha)^2 \mathrm{Var}(Q_t) + \alpha^2\sigma^2.\end{aligned}$$

Let $v_t \triangleq \mathrm{Var}(Q_t)$. We obtain a linear recurrence relation $v_{t+1} = (1-\alpha)^2 v_t + \alpha^2\sigma^2$.

Recursive Unrolling. To solve for v_t, we expand the recurrence relation backward from step t:

$$\begin{aligned}v_t &= (1-\alpha)^2 v_{t-1} + \alpha^2\sigma^2 \\ &= (1-\alpha)^2 \left[(1-\alpha)^2 v_{t-2} + \alpha^2\sigma^2\right] + \alpha^2\sigma^2 \\ &= (1-\alpha)^4 v_{t-2} + \alpha^2\sigma^2 \left[1 + (1-\alpha)^2\right] \\ &\vdots \\ &= (1-\alpha)^{2t} v_0 + \alpha^2\sigma^2 \sum_{k=0}^{t-1} \left((1-\alpha)^2\right)^k.\end{aligned} \tag{16}$$

Eq. 16 explicitly shows that the variance at time t consists of two components: the decayed initial variance (first term) and the accumulated noise variance (second term).

Asymptotic Convergence. As $t \to \infty$, since the learning rate $\alpha \in (0, 1)$, the term $(1-\alpha)^{2t}$ vanishes. The summation term is a geometric series $\sum_{k=0}^{\infty} r^k = \frac{1}{1-r}$ with ratio $r = (1-\alpha)^2$. Thus:

$$\lim_{t\to\infty} v_t = \alpha^2\sigma^2 \cdot \frac{1}{1-(1-\alpha)^2}.$$

Evaluating the denominator:

$$1 - (1-\alpha)^2 = 1 - (1 - 2\alpha + \alpha^2) = 2\alpha - \alpha^2 = \alpha(2-\alpha).$$

Substituting this back yields the tight variance bound:

$$\limsup_{t\to\infty} \mathrm{Var}(Q_t) = \frac{\alpha^2\sigma^2}{\alpha(2-\alpha)} = \frac{\alpha}{2-\alpha}\sigma^2. \tag{17}$$

Connection to Phase-A Clustering. This result provides the theoretical justification for the stability of MEMRL. While tasks within a memory cluster $\mathcal{S}(m) \triangleq \{s \in \mathcal{S} \mid \mathrm{sim}(Emb(s), Emb(z_m)) \geq \delta\}$ may vary, the Smoothness Assumption implies their rewards are drawn from a distribution with bounded variance $\sigma^2_{\mathcal{S}(m)}$. The derived bound $\frac{\alpha}{2-\alpha}\sigma^2_{\mathcal{S}(m)}$ implies that the memory utility $Q(m)$ will not diverge but will instead oscillate within a controlled range around the true expected utility. This mechanism effectively filters out high-frequency noise from diverse task instances while retaining the stable generalized value.

A.4 GEM-Style Stability via Variational Inference

In this section, we provide a theoretical foundation for MEMRL, showing a variational perspective on why the retrieval strategy and update rule support stable value estimation.

A.4.1 The Convergence Objective

Our ultimate goal is to ensure that the estimated utility $Q(m)$ converges to the true expected return of memory m. This target value is defined as:

$$\lim_{t\to\infty} \mathbb{E}[Q_t(m)] = \mathbb{E}[r|m] = \sum_{s\in\mathcal{S}(m)} \underbrace{\mathbb{E}[r|s,m]}_{\text{Stationary}} \underbrace{\Pr(s|m)}_{\text{Retrieve-Dependent}} . \tag{18}$$

The challenge lies in the term $\Pr(s|m)$—the probability that a specific state s triggers the retrieval of m. This distribution depends on the retrieval policy $\mu_t(m|s)$, which itself evolves during training, creating a circular dependency that threatens stability.

A.4.2 Variational Objective with Trust Region

To resolve this, we formulate the problem as maximizing a **global variational objective** $\mathcal{J}(\mu, Q)$. This objective serves as a tractable lower bound for the global expected return defined in Eq. 18, balanced by a semantic trust region:

$$\mathcal{J}(\mu, Q) = \mathbb{E}_{s\sim\mathcal{D}}\left[\underbrace{\sum_{m\in\mathcal{S}(s)} \mu(m|s)Q(s,m)}_{\text{Expected Utility} \approx \mathbb{E}[Q_t(m)]} - \frac{1}{\beta}\underbrace{D_{\text{KL}}\Big(\mu(\cdot|s)\Big\|\pi_{\text{sim}}(\cdot|s)\Big)}_{\text{Semantic Trust Region}}\right] \tag{19}$$

This is the objective in Eq. 8 of the main text, with $\mathcal{S}(s) = \mathcal{C}(s)$ the Phase-A support set. Here, the expectation is taken over the state distribution $\mathcal{D}$. The first term directly corresponds to the expected utility $\mathbb{E}[Q_t(m)]$ we aim to converge, while π_{sim} represents the fixed semantic prior (derived from Phase-A). The KL-divergence term acts as a regularizer crucial for two reasons:

1. **Trust Region:** It constrains the policy to the support set $\mathcal{S}$, preventing the agent from retrieving high-Q but semantically irrelevant memories (out-of-distribution errors).
2. **Regularization:** It stabilizes the learning dynamics during the "cold start" phase when Q-estimates are noisy.

A.4.3 Optimization via Generalized Expectation-Maximization (GEM)

We treat the optimization of $\mathcal{J}$ as a GEM process alternating policy improvement and value evaluation:

E-Step (Policy Optimization). We assume the utility estimates $Q(s,m)$ are fixed and seek the optimal retrieval policy μ^* that maximizes the global variational objective $\mathcal{J}(\mu, Q)$. Since the expectation is taken over the state distribution $\mathcal{D}$, we can maximize the objective for each state s pointwise. The optimization problem for a specific state s is:

$$\max_{\mu(\cdot|s)} \left[\sum_{m\in\mathcal{S}(s)} \mu(m|s)Q(s,m) - \frac{1}{\beta}D_{\text{KL}}\Big(\mu(\cdot|s)\Big\|\pi_{\text{sim}}(\cdot|s)\Big)\right] \tag{20}$$

subject to the probability simplex constraint $\sum_{m\in\mathcal{S}(s)} \mu(m|s) = 1$.

Expanding the KL-divergence term, the objective function becomes:

$$\mathcal{L}(\mu) = \sum_{m\in\mathcal{S}(s)} \mu(m|s)\left(Q(s,m) + \frac{1}{\beta}\log\frac{\pi_{\text{sim}}(m|s)}{\mu(m|s)}\right). \tag{21}$$

To enforce the normalization constraint, we introduce the Lagrange multiplier λ and construct the Lagrangian:

$$L(\mu, \lambda) = \sum_{m \in \mathcal{S}(s)} \mu(m|s) \left(Q(s,m) + \frac{1}{\beta} \log \frac{\pi_{\text{sim}}(m|s)}{\mu(m|s)} \right) + \lambda \left(1 - \sum_{m \in \mathcal{S}(s)} \mu(m|s) \right).$$

Taking the derivative with respect to $\mu(m|s)$ and setting it to zero:

$$\begin{aligned} \frac{\partial L}{\partial \mu(m|s)} &= Q(s,m) + \frac{1}{\beta} \left(\log \frac{\pi_{\text{sim}}(m|s)}{\mu(m|s)} + \mu(m|s) \cdot \frac{\mu(m|s)}{\pi_{\text{sim}}(m|s)} \cdot \frac{-\pi_{\text{sim}}(m|s)}{\mu(m|s)^2} \right) - \lambda \\ &= Q(s,m) + \frac{1}{\beta} \left(\log \frac{\pi_{\text{sim}}(m|s)}{\mu(m|s)} - 1 \right) - \lambda = 0. \end{aligned}$$

Rearranging terms to solve for $\mu(m|s)$:

$$\begin{aligned} \log \frac{\pi_{\text{sim}}(m|s)}{\mu(m|s)} &= \beta\lambda - \beta Q(s,m) + 1 \\ \frac{\mu(m|s)}{\pi_{\text{sim}}(m|s)} &= \exp\left(\beta Q(s,m) - (\beta\lambda + 1)\right) \\ \mu(m|s) &= \pi_{\text{sim}}(m|s) \exp(\beta Q(s,m)) \cdot \exp(-(\beta\lambda + 1)). \end{aligned}$$

Since $\exp(-(\beta\lambda + 1))$ is independent of m, it acts as a normalization constant $1/Z(s)$. Thus, we recover the closed-form Boltzmann distribution used in our Phase-B retrieval:

$$\mu^*(m|s) = \frac{\pi_{\text{sim}}(m|s) \exp(\beta Q(s,m))}{Z(s)} \propto \pi_{\text{sim}}(m|s) \exp(\beta Q(s,m)).$$

This derivation motivates our heuristic scoring function: the optimal retrieval policy naturally balances the semantic prior π_{sim} and the learned utility Q. By taking the logarithm, we recover the specific scoring function used in our **Phase-B Retrieval** (Eq. 5):

$$\log \mu^*(m|s) \propto \underbrace{\log \pi_{\text{sim}}(m|s)}_{\approx \text{sim}(s,m)} + \beta Q_t(s,m)$$

Thus, the practical scoring rule in Eq. 5 can be viewed as a normalized linear approximation to this utility-regularized retrieval policy.

M-Step (Policy Evaluation via Error Minimization). While the E-step improves the policy based on current estimates, the M-step ensures these estimates are grounded in reality. Fixing the policy μ_{t+1}, our goal is to align the variational parameter Q with the true environmental returns. We formulate this as minimizing the Mean Squared Error (MSE) between the estimated utility and the observed reward target $y = r$ (in our Monte Carlo style modeling):

$$\min_Q \mathcal{L}(Q) = \mathbb{E}_{\tau \sim \mu_{t+1}} \left[\frac{1}{2} \left(y - Q(s,m) \right)^2 \right] \tag{22}$$

Minimizing this error is critical because it tightens the variational bound: it ensures that the expectation term $\mathbb{E}[Q]$ in the global objective $\mathcal{J}$ (Eq. 19) converges to the true expected return $\mathbb{E}[r]$. The update rule used in our approach (Eq. 7) corresponds exactly to a Stochastic Gradient Descent (SGD) step on this objective:

$$Q_{t+1}(s,m) \leftarrow Q_t(s,m) - \alpha \nabla_Q \mathcal{L}(Q) = Q_t(s,m) + \alpha(y - Q_t(s,m))$$

By iteratively minimizing $\mathcal{L}(Q)$, the M-step propagates the environmental feedback into the utility estimates, ensuring that the subsequent E-step optimization occurs on a reliable value landscape.

A.4.4 Proof of Convergence

By the *Monotonic Improvement Theorem* of GEM [34], the sequence (μ_t, Q_t) is expected to approach a stationary point when each step sufficiently improves its corresponding objective. At

stationarity, the policy stabilizes ($\mu_{t+1} \approx \mu_t$), which implies that the inverse retrieval probability $\Pr(s|m)$ becomes **time-invariant**:

$$\Pr(s|m) = \frac{\mu^*(m|s)\Pr(s)}{\sum_{s'} \mu^*(m|s')\Pr(s')}$$

Consequently, the "Retrieve-Dependent" term in Eq. 18 is anchored. With a fixed data distribution, the $Q_t(m)$ converges to the unique fixed point:

$$\lim_{t\to\infty} Q_t(m) \to \mathbb{E}_{\mu^*}[r|m] \tag{23}$$

Thus, under the stated assumptions, the GEM-style view provides a sufficient explanation for why memory utilities can stabilize around the expected returns induced by the learned retrieval policy.

B Extended Analysis and Insights

B.1 Detailed Analysis of Forgetting Dynamics

In the main text, we reported the mean forgetting rate to quantify the stability of our method. Here, we provide a detailed visual analysis of how the forgetting rate evolves throughout the learning process on the HLE benchmark.

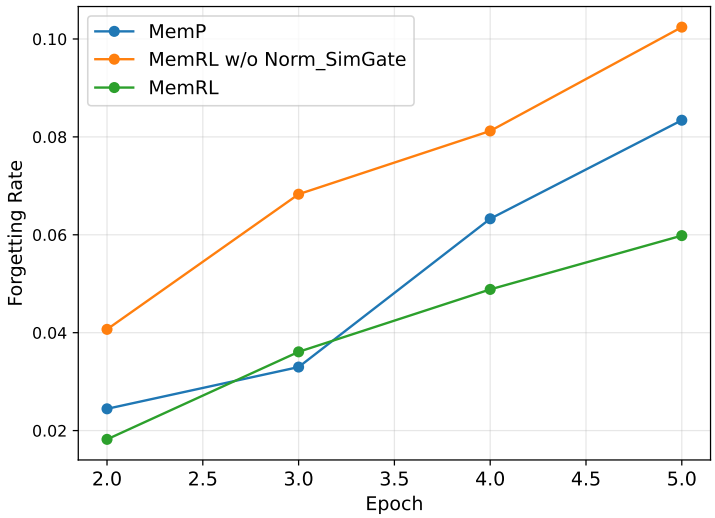


Figure 6: **Forgetting Rate Trajectory on HLE.** We compare the cumulative forgetting rate of the MEMRL against the strong baseline MemP and an ablated version of MEMRL (w/o Normalization & Gating). A lower curve indicates better stability (less empirical forgetting).

As illustrated in Figure 6, MEMRL demonstrates superior stability compared to MemP. Specifically:

- **Stability vs. Plasticity:** While MemP shows a gradual upward trend in forgetting rate as the number of episodes increases, MEMRL maintains a consistently low rate. This indicates that our dual-retrieval mechanism effectively balances the acquisition of new strategies without overwriting stable, high-utility memories.
- **Impact of Filtering:** The ablation curve (denoted as *w/o Norm & SimGate*) exhibits significant higher overall forgetting rate (0.073 mean). The visible spikes in the curve suggest that without z-score normalization and similarity gating, the agent frequently retrieves and reinforces "noisy" strategies that work for specific instances but degrade general performance on previously mastered tasks.

B.2 MEMRL as a Trajectory Verifier.

Table 4 reveals a correlation between task structural complexity and performance gain. The gains are most profound in multi-step sequential tasks (e.g., ALFWorld $+6.2\%$ Points(pp)) compared to

Table 4: **Impact of Task Structure.** Comparison of Cumulative Success Rate (CSR) gains. Multi-step tasks benefit significantly more from MEMRL.

Benchmark	Interaction	MemP (%)	MEMRL (%)	Gain (pp)
ALFWorld	Multi-step	91.9	**98.1**	**+6.2**
OS Task	Multi-step	74.2	**80.4**	**+6.2**
HLE	Single-step	57.0	60.6	+3.6
BigCodeBench	Single-step	60.2	62.7	+2.5

single-turn tasks (e.g., BigCodeBench $+2.5\%$ pp). In sequential tasks, a retrieved memory must be valid for the *entire* trajectory. Standard semantic retrieval often fetches memories that match the initial instruction but fail in later steps. By propagating the final reward backward to the memory utility Q, MEMRL effectively learns to verify the *whole trajectory*, filtering out brittle policies that look correct only on the surface.

This analysis indicates that MEMRL transcends the role of a simple retrieval enhancer to function as a *Trajectory Verifier*. Its value is maximized in tasks with complex temporal dependencies, where it learns to select memories that ensure the structural integrity of the entire interaction process.

B.3 Impact of Task Similarity on Memory Efficacy

To understand the underlying conditions where MEMRL thrives, we analyze the correlation between the intra-dataset semantic similarity (Sim_{intra}) and the absolute performance gain provided by our method ($\Delta = \text{Success Rate}_{\text{MemRL}} - \text{Success Rate}_{\text{NoMem}}$). As illustrated in Figure 7, we analyze the

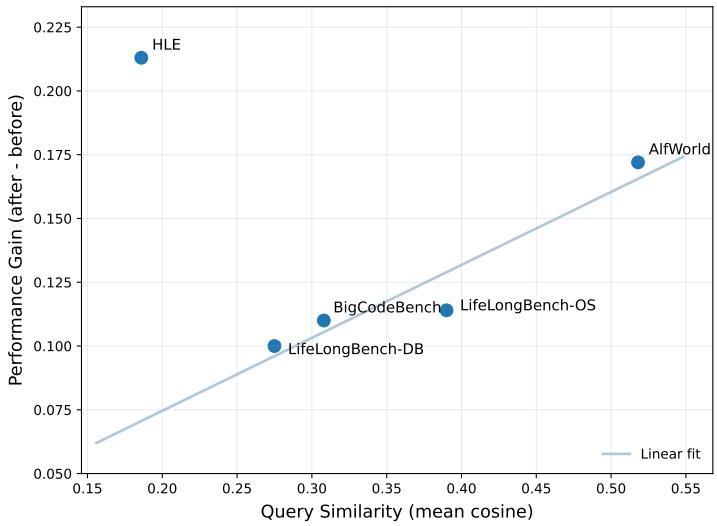


Figure 7: **Similarity-based Generalization.**

correlation between intra-dataset semantic similarity and the absolute performance gain (Δ) provided by MEMRL. The linear regression trend reveals a general positive correlation: environments with higher structural repetition allow the agent to retrieve and reuse optimal policies more effectively. At the upper extreme, **ALFWorld** (similarity 0.518) acts as a strong anchor point for this trend, exhibiting the highest repetition and a corresponding maximum performance boost ($\Delta = +0.172$). This indicates that for highly repetitive procedural tasks, memory serves as an effective shortcut to optimal trajectories. Following the regression line, benchmarks with moderate similarity—such as **Lifelong-OS** (0.390) and **BigCodeBench** (0.308)—cluster in the middle region, showing steady improvements ($\Delta \approx +0.10 \sim +0.11$) where the agent successfully generalizes coding patterns or OS commands across related instructions.

The HLE Anomaly: Generalization vs. Memorization.

Table 5: Entropy analysis for the ALFWorld task "put a hot tomato in sidetable". Lower entropy indicates stronger discrimination among recalled memories.

Metric	Value
$H(\lambda = 0)$	8.866 bits
$H(\lambda = 0.5)$	4.011 bits
ΔH	4.856 bits (54.8% reduction)

HLE presents a unique outlier. Despite having the lowest similarity (0.186) due to its diverse, multi-disciplinary nature, it exhibits a surprisingly high runtime gain ($0.357 \rightarrow 0.570, \Delta = +0.213$). This gain operates on a different mechanism than ALFWorld. In high-similarity benchmarks, MEMRL succeeds via *Positive Transfer*—generalizing shared patterns to new instances. In contrast, the gain in HLE stems from *Runtime Memorization*. Since HLE questions are distinct and domain-specific, the agent relies on the Runtime Learning phase to "memorize" specific solutions to difficult problems through repeated exposure. This distinction highlights MEMRL's versatility: it supports both *pattern generalization* in structured domains and *specific knowledge acquisition* in diverse domains.

B.4 Quantitative Evidence for Noise Reduction

In MEMRL, retrieval noise refers to memory items that pass the Phase-A semantic recall criterion (i.e., $\cos(Emb(s), Emb(z_i)) > \delta$) but are functionally unhelpful for solving the current task. Such noise is especially harmful in agentic settings, where memories can be semantically close to the query while differing in goal-critical attributes such as object state, target receptacle, or action template. To quantify how value-aware retrieval suppresses this form of noise, we conduct a controlled analysis on ALFWorld, where semantic overlap between tasks is high and the distinction between "similar" and "useful" is particularly important.

Entropy-based discrimination metric. We first measure how sharply the retrieval policy distinguishes among the recalled candidate memories. Given the candidate pool produced by Phase-A, we convert the composite retrieval scores in Eq. 5 into a probability distribution using a softmax:

$$p_i = \frac{\exp(\text{score}_i)}{\sum_j \exp(\text{score}_j)}, \tag{24}$$

and compute the Shannon entropy:

$$H = -\sum_i p_i \log_2 p_i. \tag{25}$$

A higher entropy indicates a flatter score distribution, where candidate memories are difficult to distinguish; a lower entropy indicates that the retrieval policy assigns a concentrated preference to a smaller set of high-utility memories.

Table 5 shows that pure semantic retrieval produces a high-entropy candidate distribution. In this case, when $\lambda = 0$, ten exact-goal memories receive identical scores of 1.67, making them indistinguishable under similarity alone. In contrast, with $\lambda = 0.5$, the memory with the highest learned utility ($Q = 0.996$) receives a score of 7.54, whereas the other exact-goal candidates receive scores around 0.81. Thus, the Q-value term creates clear discrimination in precisely the setting where semantic similarity fails to do so.

Directional validation of the entropy reduction. The entropy analysis above shows that MEMRL increases discrimination among candidate memories. We further verify that this discrimination is directionally correct, i.e., that the memories promoted by value-aware retrieval are indeed more useful. We analyze the ALFWorld contrastive set where $\lambda = 0.5$ solves 138 additional tasks beyond those solved by $\lambda = 0$.

As shown in Table 6, the retrieval budget is identical under the two settings, with an average of 2.99 retrieved memories per record. Nevertheless, the retrieved-memory success rate increases from 25.19% under pure similarity retrieval to 90.47% under value-aware retrieval. More importantly, even exact-goal matches are not necessarily reliable: under $\lambda = 0$, their success rate is only 13.53%,

Table 6: Retrieval quality on the 138-task advantage set in ALFWorld. Both settings use the same retrieval budget, but value-aware retrieval selects substantially more useful memories.

Metric	$\lambda = 0$	$\lambda = 0.5$	Interpretation
Avg. retrieved memories	2.99	2.99	Same budget
Retrieved-memory success rate	25.19%	90.47%	3.6× more useful memories
Exact-goal match success rate	13.53%	85.24%	Exact similarity alone is unreliable
Task-level success	0/1,880	1,415/1,880	0% → 75.3% completion

Table 7: Failure-mode breakdown of memories retrieved by pure similarity retrieval ($\lambda = 0$). Categories are non-exclusive.

Failure mode	Count / Share	Description
Destination mismatch	2,988 / 53.20%	Same object, wrong target receptacle
Object mismatch	936 / 16.70%	Different object
State mismatch	854 / 15.20%	Wrong object state, e.g., hot/cool/clean
Verb mismatch	717 / 12.80%	Different action template

because the memory bank may contain many failed or brittle attempts for hard tasks. By incorporating learned utility, $\lambda = 0.5$ raises the exact-goal match success rate to 85.24%. At the task level, although $\lambda = 0$ retrieves memories on every record, it fails all $1{,}880$ evaluated cases in this contrastive set, whereas $\lambda = 0.5$ completes $1{,}415$ of them. This indicates that Q-value re-ranking does not merely sharpen the score distribution; it sharpens it toward functionally helpful memories.

B.5 Systematic Analysis of Similarity-Retrieval Failures

We further analyze when and why pure similarity retrieval fails using the same 138-task contrastive set from Appendix B.4. These tasks are concentrated in state-sensitive ALFWorld scenarios, especially `put` tasks (60.9%) and `heat` tasks (32.6%). Such tasks require the correct conjunction of object identity, object state, and target receptacle. They often share substantial lexical overlap in room layouts, objects, and intermediate actions, which maximizes surface similarity while obscuring goal-critical differences.

Table 7 shows that the dominant failure mode is destination mismatch, accounting for 53.20% of the retrieved memories. That is, pure similarity retrieval often selects memories involving the same object but the wrong target receptacle. Object mismatch, state mismatch, and verb mismatch further account for 16.70%, 15.20%, and 12.80%, respectively. These errors illustrate why semantic similarity is insufficient for embodied procedural tasks: the retrieved trajectory can look highly relevant at the lexical or embedding level, yet still encode an action sequence that is incompatible with the current goal.

The failure occurs at two levels. First, similarity retrieval often retrieves the wrong memories: as Table 7 shows, the most common retrieved distractors preserve surface overlap but violate goal-critical constraints. Second, even when similarity retrieval retrieves exact-goal matches, these memories can still be unreliable. As shown in Table 6, the exact-goal match success rate under $\lambda = 0$ is only 13.53%, because repeated failures on difficult tasks can accumulate in the memory bank and remain indistinguishable from successful attempts under similarity alone. In contrast, value-aware retrieval with $\lambda = 0.5$ uses environmental feedback to down-rank these unhelpful exact matches and up-rank memories with higher expected utility, yielding an exact-goal match success rate of 85.24%.

Overall, these results provide direct evidence for the mechanism behind MEMRL's noise reduction. Phase-A similarity recall preserves semantic relevance, while Phase-B Q-value re-ranking filters functional distractors within the recalled pool. This explains why the balanced setting ($\lambda = 0.5$) achieves the best empirical trade-off: similarity prevents context detachment, whereas learned utility suppresses memories that are semantically plausible but behaviorally harmful.

C Advanced Capabilities: Transfer and Merging

C.1 Cross-Model Memory Transferability

We investigate whether the procedural knowledge captured by MEMRL is specific to the training policy or if it generalizes across different architectures. To test this, we take a memory bank fully trained for 10 epochs on the HLE benchmark using our strongest agent, `Gemini-3-pro`, and directly transfer it—without any fine-tuning—to three distinct inference models: `Qwen3-235B`, `GPT-5.2(High)`, and `Gemini-3-flash`.

As summarized in Table 8, the transferred memory yields substantial zero-shot performance gains across all models. Notably, smaller or distilled models experience the largest relative improvements; for instance, `Qwen3-235B` improves by over $3\times$ ($0.150 \rightarrow 0.531$) and `Gemini-3-flash` nearly doubles its success rate ($0.347 \rightarrow 0.583$). Even `GPT-5.2(High)`, a highly capable reasoning model, sees a significant boost ($0.354 \rightarrow 0.571$).

These results suggest that MEMRL captures *model-agnostic* problem-solving patterns—such as efficient code skeletons and reasoning templates—rather than model-specific artifacts. This effectively allows the memory bank to function as a portable knowledge base, enabling weaker models to "inherit" the capabilities of a stronger teacher model through simple retrieval.

Table 8: **Cross-Model Transfer Results on HLE.** Performance of various agents using a frozen memory bank originally trained by `Gemini-3-pro`. Δ denotes the absolute improvement over the base model.

Inference Model	Base Score	Transfer Score	Gain (Δ)
`Qwen3-235B`	0.150	0.531	**+0.381**
`Gemini-3-flash`	0.347	0.583	+0.236
`GPT-5.2(High)`	0.354	0.571	+0.217

C.2 Multi-Task Memory Merging and Interference Analysis

To evaluate the composability and robustness of MEMRL in multi-task scenarios, we conducted a memory merging experiment. Specifically, we aggregated the finalized memory banks learned from the last epoch of two distinct domains within the Lifelong Agent Bench: Operating System Control (M_{OS}) and Database Management (M_{DB}). The agent was then evaluated on each respective benchmark using this unified, heterogeneous memory bank ($M_{Unified} = M_{OS} \cup M_{DB}$), without any further training or fine-tuning.

As presented in Table 9, the results demonstrate that merging memories introduces negligible interference. The performance on the DB Task remains identical (0.960), while the OS Task exhibits only a marginal fluctuation ($0.788 \rightarrow 0.784$). This robustness is intrinsic to our **Two-Phase Retrieval** mechanism. Since the semantic spaces of OS commands and SQL queries are largely orthogonal, the Phase-A similarity filter effectively acts as a semantic gate, automatically excluding irrelevant cross-task memories before they enter the value-based ranking stage. This indicates that MEMRL supports modular memory composition, allowing agents to scale capabilities by simply merging memory modules without suffering from negative transfer or catastrophic interference.

Table 9: **Memory Merging Results.** Performance comparison between using task-specific individual memory banks versus a merged unified memory bank. The minimal delta indicates strong resistance to cross-task interference.

Benchmark Task	Individual Memory (Original)	Merged Memory ($M_{OS} \cup M_{DB}$)	Performance Delta (Δ)
Lifelong-OS (OS Task)	0.788	0.784	-0.004
Lifelong-DB (DB Task)	0.960	0.960	0.000

D Baseline and Benchmark Details

To ensure a rigorous evaluation, we compare MEMRL against a diverse set of baselines ranging from simple sampling strategies to advanced agentic memory systems. All baselines are evaluated under a **unified frozen-backbone setting** to isolate the contribution of the memory and retrieval mechanisms.

D.1 Baselines

We categorize the baselines into three groups based on their interaction with memory and environment:

I. Test-Time Scaling Strategy

- **Pass@k:** This is a standard sampling-based baseline. It generates k independent candidate solutions for a given query and selects the best one based on the benchmark's verifier (if available) or reports the success rate if at least one candidate passes. This baseline serves as a measure of the inherent capability of the frozen LLM without any memory persistence.

II. Retrieval-Augmented Generation (RAG) Approaches

- **RAG [23]:** Represents the standard semantic retrieval paradigm. It utilizes an embedding model to encode the current query and retrieves the top-k most semantically similar past experiences (or documents) from the external memory. These retrieved contexts are then prepended to the prompt to guide the LLM's generation.
- **Self-RAG [1]:** An advanced RAG variant that incorporates a self-critique mechanism. Unlike standard RAG, Self-RAG performs selective retrieval and uses a critique model (or self-prompting) to verify the relevance and factual correctness of the retrieved content before integrating it into the generation process.

III. Agentic Memory Systems

- **Mem0 [5]:** A recently proposed memory layer for LLMs that manages memory through structured operations. It employs specific APIs for adding, retrieving, and updating memory, aiming to maintain a personalized and persistent context across interactions.
- **MemP [9]:** A framework focused on procedural memory. It distills past successful trajectories into reusable, procedure-like memory entries. MemP maintains a memory repository using a build-retrieve-update cycle, allowing the agent to recall high-level plans rather than raw trajectory data.

D.2 Benchmark Datasets

We evaluate performance across four benchmarks selected to cover diverse domains: code generation, OS interactions, embodied decision-making, and multidisciplinary reasoning.

- **BigCodeBench [79]:** A challenging benchmark for library-oriented code generation that requires agents to implement complex functionalities using diverse third-party libraries. Unlike traditional benchmarks focused on algorithmic snippets, BigCodeBench emphasizes practical software engineering capabilities. We evaluate on the **BigCodeBench-Instruct (Full)** split, which tasks the agent with synthesizing complete functional code from natural language instructions across the full range of difficulty levels.
- **Lifelong Agent Bench [75]:** Designed to evaluate agents in a continuous learning setting involving Operating System (OS) and Database (DB) interactions. It tests the agent's capacity to adapt to new tools and commands over a long horizon without forgetting previous skills.
- **ALFWorld [49]:** An embodied navigation and manipulation benchmark. It requires the agent to solve textual logic puzzles within a simulated household environment (e.g., "put a clean apple in the fridge"). This tests the agent's ability to learn and retrieve multi-step plans.
- **Humanity's Last Exam (HLE) [38]:** A rigorous multidisciplinary reasoning benchmark featuring hard problems from mathematics, humanities, and sciences. It serves as a stress test for the agent's general reasoning capability and its ability to retrieve relevant knowledge for disparate tasks.

E Implementation and Reproducibility Details

To facilitate reproducibility, we provide the exact model versions, hyperparameter settings, and environmental configurations used in our experiments.

E.1 Model Specifications

All LLM reasoning and generation tasks were performed using the models listed in Table 10. We used the official APIs with a fixed temperature to ensure deterministic evaluation where possible.

Table 10: **Model and API Configurations.**

Component	Configuration / Version	Notes
Backbone LLM	`GPT-4o`	Used for BigCodeBench
	`GPT-4o-mini`	Used for Lifelong Bench
	`GPT-5-mini`	Used for ALFWorld
	`Gemini-3-pro`	Used for HLE
Embedding Model	`Text-Embedding-3-Large`	Used for Intent and Query encoding
Generation Params	Temperature $= 0.0$	General (Greedy decoding)
	Temperature $= 0.6$	Specific for HLE (`gemini-3-pro`)
	Top-p $= 1.0$	Default

E.2 Hyperparameter Settings

Table 11 details the specific hyperparameters used for MEMRL and the baselines. The similarity threshold δ is adaptive to the dataset density; specifically, we determine δ by calculating the pairwise cosine similarity distribution of task descriptions within each benchmark and selecting the threshold at the top 20% quantile. This ensures that only the most relevant historical experiences are considered for retrieval.

Table 11: **Hyperparameter Settings across Benchmarks.**

		Benchmark Setting				
Parameter	**Description**	**BigCodeBench**	**Lifelong Bench (OS)**	**Lifelong Bench (DB)**	**ALFWorld**	**HLE**
MEMRL *(Ours)*						
α	Learning Rate (Eq. 7)	0.3	0.3	0.3	0.3	0.3
λ	Q-Weight Balance (Eq. 5)	0.5	0.5	0.5	0.5	0.5
δ	Similarity Threshold	0.38	0.50	0.37	0.62	0.25
k_1	Phase-A Recall Size	10	10	10	5	5
k_2	Phase-B Select Size	5	5	5	3	3
Q_{init}	Initial Q-value	0.0	0.0	0.0	0.0	0.0
Baselines						
k_{RAG}	Retrieval Top-k	5	5	5	3	3
$k_{SelfRAG}$	Retrieval Top-k	5	5	5	3	3
k_{Mem0}	Retrieval Top-k	5	5	5	3	3
k_{MemP}	Retrieval Top-k	5	5	5	3	3

E.3 Data Partitioning

To evaluate the effectiveness of MEMRL, we categorize our experiments into *Runtime Learning* and *Transfer Learning* settings. Table 12 summarizes the dataset sizes and partitioning strategies used for each benchmark.

For benchmarks utilizing random splits (`OS`, `DB`, and `BCB`), we use a fixed random seed of 42 to ensure reproducibility. In `ALFWorld`, the *Transfer Learning* phase specifically tests generalization to new instances within known categories, ensuring the agent learns procedural patterns rather than specific trajectories.

Table 12: **Summary of Data Partitioning and Dataset Sizes.**

Benchmark	Runtime Learning	Transfer Learning	Split / Note
HLE	2,500 tasks	–	Full set (Runtime only)
Lifelong Agent (OS)	500 tasks	500 tasks	7:3 Split (Seed 42)
Lifelong Agent (DB)	500 tasks	500 tasks	7:3 Split (Seed 42)
BigCodeBench-I (Full)	1,140 tasks	1,140 tasks	7:3 Split (Seed 42)
ALFWorld	3,553 tasks	140 tasks	`valid_seen` (Novel instances)[†]

[†] For ALFWorld Transfer Learning, the agent uses the same 3,553 tasks as memory but is evaluated on 140 novel instances of seen task types.

E.4 Model Selection and Performance Analysis

We select the backbone model for each benchmark to ensure a *valid learning signal* relative to task complexity. Since **MemRL** relies on high-value trajectories to perform utility update over retrieved memories, extremely low initial competence can make the feedback effectively unusable. For instance, on challenging benchmarks such as HLE, weaker models may start at $\approx 4\%$ success rate (e.g., `GPT-4o-mini`), yielding too few successful trajectories for stable utility estimation; in this scenario, environmental feedback is dominated by noise, which can prevent meaningful learning and hinder convergence.

At the other extreme, using the strongest available models on simpler benchmarks can cause *performance saturation* (ceiling effects), leaving little headroom to quantify the marginal gains attributable to the memory mechanism. Therefore, we match model capability to each task's difficulty to avoid both the *no-signal situation* (insufficient successes) and the *ceiling effect* (insufficient headroom). This design choice enables a more faithful evaluation of improvements that are intrinsic to **MemRL** rather than artifacts of an ill-posed performance condition.

Importantly, our evaluation spans backbones of different scales—from "mini" to "pro" tiers (Table 10)—to test whether **MemRL** remains effective across capability levels. The resulting performance trends support the **scale-invariant robustness** of **MemRL**. Moreover, the cross-model transfer results (Table 8) further indicate that the procedural knowledge captured in memory is portable across backbones, suggesting that the learned utility over memories generalizes beyond any single model's inherent strength.

To further explore the generality of **MemRL** across different model capabilities and task complexities, we also deployed the `GPT-4o-mini` model in the ALFWorld benchmark. This model is identical to the backbone used in the Lifelong Agent Bench, allowing for a direct comparison of its performance across environments with varying exploration intensities. As an exploration-intensive environment, ALFWorld poses significant demands on the base model's reasoning and planning capabilities. The Table 13 and Table 14 present the runtime learning and knowledge transfer abilities of `GPT-4o-mini`.

Table 13: Runtime Learning Performance of `GPT-4o-mini` Across Benchmarks

Method	Lifelong Agent (OS)		Lifelong Agent (DB)		ALFWorld		Average	
	Last Epoch SR	CSR	Last Epoch SR	CSR	Last Epoch SR	CSR	Last Epoch SR	CSR
No Memory	0.674	N/A	0.860	N/A	0.278	N/A	0.604	N/A
Pass@10	N/A	0.756	N/A	0.928	N/A	0.462	N/A	0.715
MemP	0.736	0.742	**0.960**	0.966	0.299	0.413	0.665	0.707
MEMRL	**0.788**	**0.804**	**0.960**	**0.972**	**0.440**	**0.680**	**0.730**	**0.819**

Table 14: Transfer Learning Performance of `GPT-4o-mini` Across Benchmarks

Method	Lifelong Agent (OS)	Lifelong Agent (DB)	ALFWorld	Average
No Memory	0.673	0.841	0.314	0.609
MemP	0.720	0.928	0.314	0.654
MEMRL	**0.746**	**0.942**	**0.457**	**0.715**

These results, using `GPT-4o-mini` as a consistent backbone, further indicate that MEMRL's gains are consistent across benchmarks under a shared backbone. In runtime learning (Table 13), MEMRL

achieves the highest average Last Epoch Success Rate and Cumulative Success Rate, with clear gains: its average Last Epoch SR is 12.6 percentage points (pp) higher than "No Memory", and its average CSR surpasses MemP by 11.2 pp. This advantage is particularly pronounced in the challenging ALFWorld environment. Similarly, in transfer learning (Table 14), MEMRL secures the highest average Success Rate, 10.6 pp higher than "No Memory" and 6.1 pp higher than MemP. These consistent improvements across tasks and learning paradigms confirm that MEMRL effectively enhances the `GPT-4o-mini` backbone's decision-making capabilities, leveraging learned utility over memories for more robust performance.

F Cost, Efficiency, and Scalability Analysis

Real-world deployment of autonomous agents requires balancing performance gains with computational costs. In this section, we analyze the token consumption and runtime latency of MEMRL compared to the strong baseline MemP on the compute-intensive **Humanity's Last Exam (HLE)** benchmark.

F.1 Token Consumption

Since MEMRL operates as a non-parametric approach without gradient-based fine-tuning, the primary cost arises from LLM API calls. We compare the average token usage per question (Q) across the entire learning trajectory (10 epochs).

MEMRL's token consumption is comparable to that of MemP, as both methods utilize identical interaction loops (reasoning + summarization). On the HLE benchmark, the average total token consumption per question for MEMRL is approximately **32K**. This similarity in token usage is because the complexity of MEMRL lies in *how* memories are retrieved and updated (via Q-values), not in *how much* context is fed to the LLM. Therefore, the significant performance gains of MEMRL reported in the main text are achieved without increasing the inference budget.

F.2 Runtime Latency and Stability

A common concern with two-stage retrieval and reinforcement learning components is the potential for increased latency. We empirically validate the wall-clock time required to complete each epoch (2,500 questions) in Figure 8.

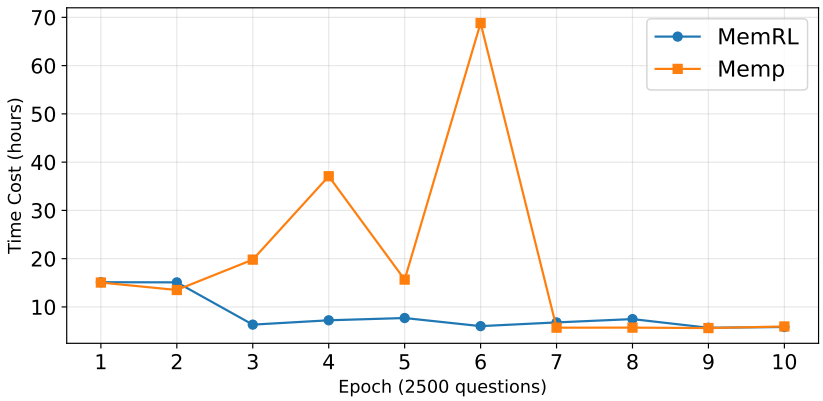


Figure 8: **Wall-Clock Time per Epoch on HLE.** The x-axis represents learning epochs (2,500 questions per epoch), and the y-axis represents the total time in hours. MEMRL exhibits a stable runtime profile, confirming that the algorithmic overhead is negligible compared to system variances (e.g., network latency).

Negligible Algorithmic Overhead. Figure 8 demonstrates that the runtime of MEMRL is commensurate with, and often more stable than, that of MemP. The fluctuations observed (e.g., the spikes

in the MemP curve at Epochs 4 and 6) are attributed to external factors such as API network latency and throughput variability, rather than algorithmic complexity.

Specifically, the additional components in MEMRL introduce minimal computational cost:

- **Dual-Stage Retrieval:** This involves basic vector dot-products and scalar score weighting, operating in milliseconds.
- **RL Update:** The Q-value update in a Monte Carlo style on scalars, which is an $O(1)$ operation.

Compared to the hours required for LLM generation over 2,500 questions, these millisecond-level operations are virtually imperceptible. The stability of the MEMRL curve also indicates that our method adds no complex blocking operations likely to induce network delays.

F.3 Scalability with Large Memory Banks

MEMRL is designed to scale with a growing memory bank. Our experiments already operate in a large-memory regime: HLE ($2{,}500$ tasks $\times$ 10 epochs) and ALFWorld ($3{,}553$ tasks $\times$ 10 epochs) both accumulate more than $10{,}000$ memory entries, while maintaining stable performance (e.g., ALFWorld CSR $= 0.983$ in Table 1). This scalability comes from the two-stage retrieval design. Phase A uses the similarity threshold δ to filter the full bank into a small semantically relevant candidate set, and Phase B re-ranks only the recalled candidates ($k_1 = 5$ or 10 in our experiments). The utility update is a scalar Monte Carlo-style operation with $O(1)$ cost per selected memory.

The system is also self-regulating: memories with poor downstream outcomes receive decreasing Q-values and are naturally down-ranked by Phase-B selection. Therefore, as the memory bank grows, low-utility memories remain stored but are effectively silenced during retrieval. As shown in Appendix F, the retrieval and update components introduce negligible runtime overhead compared with LLM generation.

F.4 Compute Resources.

All experiments use API-based LLM backbones rather than local model training or fine-tuning. Thus, the main compute cost comes from LLM inference calls, while local computation consists of CPU-based embedding retrieval, score re-ranking, and scalar Q-value updates. These memory-side operations require negligible compute compared with LLM generation. We report token consumption and wall-clock latency in Appendix F.1 and Appendix F.2.

G Statistical Reliability and Additional Analyses

This appendix reports multi-run statistics for the main results (Appendix G.1), a controlled comparison with JitRL (Appendix G.2), the cold start of newly written memories (Appendix G.3), the relation between retrieval-based and parametric learning (Appendix G.4), and the applicability of MEMRL beyond language agents (Appendix G.5).

G.1 Multi-Run Results and Aggregate Confidence Intervals

Protocol. For No Memory, RAG, MemP, and MEMRL, we repeat the full runtime-learning and transfer protocol on BigCodeBench, Lifelong Agent Bench (OS and DB), and ALFWorld with three run seeds (42, 43, 44). The data split is fixed across runs, so the seeds vary only the run randomness. Backbones and hyperparameters follow Appendix E. Table 15 reports mean $\pm$ standard deviation across the three runs on a 0–1 scale; the means are the multi-run entries of Tables 1 and 2, which report means only. Due to API cost, HLE (Gemini-3-pro) and the remaining baselines are single runs.

Aggregate analysis. Table 16 reports macro-averages over the four benchmarks (in percentage points) with approximate 95% confidence intervals. Because overlap between separate intervals is not a test of difference, Table 17 additionally reports intervals for the difference between MEMRL and the strongest baseline on each metric (MemP in all three cases). All three intervals exclude zero. The analysis supports a consistent aggregate advantage; it does not establish a significant gain on each benchmark. In particular, on DB the intervals of MEMRL and MemP overlap on all three metrics, and on BigCodeBench Last and Transfer they overlap as well.

Table 15: Three-run results (mean ± sample standard deviation, seeds 42/43/44). Last and CSR are runtime-learning metrics; Transfer is the best held-out success rate. No Memory has no cross-epoch memory, so CSR is not applicable.

Benchmark	Method	Last Epoch SR	CSR	Transfer SR
BigCodeBench	No Memory	0.481 ± 0.006	–	0.482 ± 0.005
	RAG	0.458 ± 0.021	0.468 ± 0.019	0.478 ± 0.002
	MemP	0.582 ± 0.005	0.599 ± 0.004	0.488 ± 0.008
	MEMRL	0.596 ± 0.011	0.624 ± 0.011	0.500 ± 0.008
LLAB-OS	No Memory	0.649 ± 0.022	–	0.671 ± 0.008
	RAG	0.665 ± 0.023	0.690 ± 0.011	0.701 ± 0.014
	MemP	0.752 ± 0.015	0.776 ± 0.030	0.716 ± 0.005
	MEMRL	0.777 ± 0.011	0.807 ± 0.007	0.745 ± 0.006
LLAB-DB	No Memory	0.863 ± 0.004	–	0.894 ± 0.046
	RAG	0.916 ± 0.014	0.917 ± 0.014	0.923 ± 0.004
	MemP	0.961 ± 0.001	0.967 ± 0.002	0.935 ± 0.007
	MEMRL	0.961 ± 0.008	0.969 ± 0.005	0.940 ± 0.004
ALFWorld	No Memory	0.777 ± 0.001	–	0.824 ± 0.009
	RAG	0.887 ± 0.007	0.940 ± 0.011	0.933 ± 0.015
	MemP	0.884 ± 0.004	0.922 ± 0.002	0.914 ± 0.015
	MEMRL	0.942 ± 0.006	0.983 ± 0.001	0.981 ± 0.003

Table 16: Macro-averaged success rates (%) over BigCodeBench, LLAB-OS, LLAB-DB, and ALFWorld, with approximate 95% confidence intervals.

Method	Last Epoch SR	CSR	Transfer SR
MEMRL	81.91 [81.13, 82.69]	84.57 [83.91, 85.24]	79.15 [78.66, 79.65]
MemP	79.47 [78.50, 80.43]	81.61 [79.41, 83.81]	76.33 [75.42, 77.24]
RAG	73.15 [71.63, 74.67]	75.39 [74.18, 76.59]	75.88 [74.90, 76.87]
No Memory	69.26 [67.77, 70.75]	–	71.77 [68.59, 74.96]

G.2 Comparison with JitRL

Methodological distinction. JitRL [26] and MEMRL both keep the backbone frozen and use runtime feedback without parameter updates, but they optimize different objects. JitRL stores state–action–return tuples, estimates the advantages of executable actions, and adjusts the output logits. MEMRL treats retrieval as the RL decision: it learns Q to select experiences and conditions frozen-LLM generation on them. In short, JitRL learns *which action to execute*, whereas MEMRL learns *which experience should guide generation*. This affects applicability. JitRL's policy-consistency result is stated for a finite candidate action set (Theorem 4.3 and Appendix D of [26]), and its value-consistency assumption requires repeated action-matched neighbours for a fixed state–action pair (Assumption C.4 of [26]). Accordingly, its implementation builds a finite candidate set, for example restricting Jericho to game-engine valid actions. Related output-action methods share this interface: TWOSOME [57] scores all valid actions, and GLAM [4] evaluates every environment action. We do not claim that extending JitRL beyond finite action sets is impossible; rather, its guarantees and implementation rely on enumerable candidates with repeated support. HLE requires open-ended answers, BigCodeBench unrestricted programs, and LLAB open-ended SQL or shell commands, so these benchmarks do not naturally satisfy this assumption.

LLAB with candidate abstraction. To probe this boundary, we adapted JitRL to LLAB by abstracting free-form generations into finite candidate commands while retaining its state–action values (Table 18, GPT-4o-mini, three seeds). JitRL underperforms both No Memory and MEMRL on DB and OS. We read this result as evidence about the applicability boundary rather than as a head-to-head comparison: SQL and shell commands are dynamically generated strings without stable cross-task action identity, so the adaptation places JitRL outside the setting it was designed for.

Table 17: Difference between MEMRL and the strongest baseline on macro-averaged metrics.

Metric	MEMRL	Strongest baseline	Difference	Approx. 95% CI
Last Epoch SR	81.91	MemP 79.47	+2.45	[1.45, 3.44]
CSR	84.57	MemP 81.61	+2.96	[1.09, 4.83]
Transfer SR	79.15	MemP 76.33	+2.82	[1.90, 3.75]

Table 18: JitRL on LLAB with candidate abstraction (success rate, %). No Memory and MEMRL are given for reference.

Task	Seed 42	Seed 43	Seed 44	JitRL	No Memory	MEMRL
DB	77.70	80.58	79.14	79.14 ± 1.44	89.40 ± 4.60	94.00 ± 0.40
OS	52.00	51.33	47.33	50.22 ± 2.53	67.10 ± 0.80	74.50 ± 0.60

Controlled comparison on ALFWorld. Among our benchmarks, ALFWorld is closest to JitRL's assumptions because it provides admissible actions. We therefore ran a controlled comparison with the same GPT-5-mini backbone, a 710-task (20%) training subset, 10 epochs, and seeds 42/43/44, evaluating all methods on the complete 140-task valid-seen set (Table 19). Because of the reduced training subset, these numbers are not directly comparable to Table 2. JitRL scores below No Memory, while MEMRL improves over both. A diagnostic explains the gap: ALFWorld grounds its discrete actions in instance-specific objects, receptacles, and locations, so exact action identities rarely recur across tasks. For JitRL, the action-match rate was 3.7–4.6% and the retrieval coverage 9.6–11.0%, so most retrieved trajectories cannot reliably estimate the advantage of the current action, and inaccurate transfer can interfere with a strong backbone. MEMRL transfers complete experiences by utility and does not require exact action-string reuse.

Table 19: Controlled comparison on ALFWorld valid-seen, success rate (%) with GPT-5-mini on 20% of the training set.

Method	Seed 42	Seed 43	Seed 44	Mean $\pm$ Std
No Memory	83.60	81.43	82.14	82.39 ± 1.11
JitRL	80.71	75.00	72.86	76.19 ± 4.06
MEMRL	95.00	97.14	97.86	96.67 ± 1.49

Summary. JitRL reweights executable actions by value, whereas MEMRL selects transferable experiences by value. We expect JitRL's direct correction of the action distribution to be more efficient in tasks with small, stable candidate action sets, and MEMRL to be more broadly applicable when the output space is open-ended. The two are complementary, and combining input-level experience selection with output-level action correction is a natural direction for future work.

G.3 Cold Start of Newly Written Memories

Why $Q_{\text{init}} = 0$. A new memory's initial value encodes *unverified* transfer utility, not unhelpfulness. Because Q does not enter Phase A, similarity alone can admit a new memory into the candidate pool, and Phase B combines similarity with utility, so a highly relevant new memory can still be selected and is updated after a single outcome. MEMRL writes a memory after every trajectory, so recurring tasks produce many near-duplicate experiences. We avoid a recency priority because new memories may be noisy or come from failed trajectories, whose outcomes do not establish future utility. If a validated, high-utility memory already covers a recurring intent, a recent near-duplicate should first demonstrate additional utility before receiving equal priority. This reduces redundancy in the retrieved context and leaves room for distinct experiences, although it does not guarantee diversity across the whole bank.

Exposure dynamics. There is no fixed warm-up period: a new memory can be selected on the next sufficiently similar query, and its first selection depends on how often such intents recur and on the local competitors. Bank growth alone does not enlarge Phase-B competition, because Phase

A fixes a candidate set of size at most k_1. The remaining risk is local near-duplication: many mature, high-utility candidates for the same intent can delay the exposure of a new one. This is the stability–plasticity trade-off at the level of individual intents.

Sensitivity. On DB, replacing $Q_{\text{init}} = 0$ with an optimistic $Q_{\text{init}} = 0.5$ changes the success rate from 0.960 to 0.966 (0.6 pp). On this task, strongly prioritizing new memories gives only a modest aggregate benefit, and the default initialization does not materially degrade performance. This is a single task; low-similarity settings such as HLE, where new memories may receive less exposure, are not covered, and a systematic study of initialization strategies across benchmarks is left for future work.

G.4 Retrieval-Based and Parametric Learning

Retrieval-only adaptation has a ceiling: a frozen model can use experience only through its context, so improvement is bounded by the base model, context length, and retrieval quality. We also note that fine-tuning risks, but does not necessarily cause, catastrophic forgetting [21, 24, 64]. In a preliminary experiment on ALFWorld Task 6, MEMRL reached a final success rate of 95.83%, whereas one epoch of supervised fine-tuning (SFT) on all accumulated memories reached 75.00%. Alternating MEMRL-based experience collection, selection of high-value memories, and SFT reached 100% after four rounds, without memory retrieval at inference. This single-task result is preliminary, but it suggests that retrieval and fine-tuning are complementary: MEMRL provides immediate in-context adaptation and identifies experiences worth consolidating, while SFT distills them into parameters and can exceed the retrieval-only ceiling. A dual-timescale framework that combines fast memory-level learning with periodic, utility-guided parameter consolidation is a promising direction; it would also realize the episodic-to-semantic consolidation suggested by complementary learning systems theory [31, 22].

G.5 Beyond Language Agents

MEMRL is not restricted to language in principle. It requires (i) a representation space for recalling related experiences, (ii) an episodic representation that can condition the agent, and (iii) environmental feedback for updating utility. In a multimodal or embodied agent, memories could contain visual observations or sensor–action trajectories, with a modality-specific encoder replacing the text encoder while the utility update stays unchanged. ViLoMem [3], which accumulates and refines visual and logical experience in a multimodal memory, provides initial evidence that experience-based memory adaptation extends beyond text, although its update mechanism differs from ours.

H Extended Limitations and Future Work

This appendix provides a more detailed discussion on the limitations of MEMRL and outlines promising avenues for future research, building upon the foundations established in the main paper.

H.1 Update Protocols and Memory Consolidation

The current step-wise update protocol of MEMRL, while enabling rapid adaptation, can introduce high-variance noise in long-horizon trajectories. This presents a challenge for stabilizing value estimation over extended sequences. A valuable future direction involves exploring more robust multi-step update mechanisms, which could offer slower but more stable learning. Furthermore, combining these with periodic consolidation of similar intentions and experiences within the memory bank could significantly improve the spatial efficiency of the memory, reducing redundancy and enhancing retrieval quality.

H.2 Precise Credit Assignment in Multi-Memory Updates

A significant challenge arises from the credit-assignment ambiguity encountered when multiple experiences from the memory are referenced and updated simultaneously. Determining the precise contribution of each referenced memory to the final outcome is complex and affects the efficiency of learning. Future research could investigate methods for more precise experience attribution, drawing inspiration from techniques like Shapley values [47], commonly used in cooperative game theory,

or value decomposition methods prevalent in multi-agent reinforcement learning [42, 54]. Such approaches could lead to more accurate updates and faster convergence.

H.3 Task Similarity and Generalization

MEMRL's effectiveness is observed to improve with the number of encountered tasks, aligning well with the characteristics of runtime learning. This is because the algorithm leverages past experiences, and a richer, more diverse set of similar experiences directly enhances its ability to retrieve relevant knowledge. However, when task similarity in the experience base is low, the method may inherently degrade into a less efficient reflection-like behavior, as direct experience transfer is limited. This is not a deficiency of the method but rather a characteristic of its reliance on learned utility from memory. For industrial deployment and real-world applications, this implies that maintaining a sufficiently high "task similarity density" within the agent's operating environment or its collected memory is crucial. Strategies to achieve this could include active curriculum learning, environment design that promotes diverse yet related tasks, or techniques that facilitate hierarchical abstraction to enable generalization beyond direct, low-level experience matches. Enhancing retrieval mechanisms to proactively identify and adapt to novel task structures, or integrating hierarchical abstraction, are key for improving cross-task generalization.

H.4 Memory Security and Robustness to Attack

MEMRL is sensitive to the quality of feedback, particularly vulnerable to "reward hacking" if the verifier produces false positives. Incorrectly learned high Q-values from spurious feedback can quickly solidify and propagate erroneous behavioral patterns, leading to systemic failures. This highlights a critical challenge: memory security. A maliciously injected sample into the memory bank could rapidly diffuse pollution, potentially causing an intelligent agent to collapse. Future research must address the safety and trustworthiness of agent memories. Fortunately, the mutable nature of experience offers a silver lining: once contamination or attack is identified, polluted experiences can be swiftly pruned, allowing for recovery without disrupting the utility of previously learned valid experiences.

H.5 Multi-Agent Collaboration and Shared Memory

As many enterprises transition from single agents to multi-agent clusters (swarms), the question of "shared memory" becomes important. Does the MEMRL approach support knowledge sharing, allowing lessons learned and updated Q-values from one agent to immediately benefit others? We've explored scenarios where the memory, treated as a persistent resource, can indeed be shared among different models or agents (Appendix C.1). Furthermore, adopting a multi-agent paradigm, akin to distributed parameter updates, could be highly beneficial. Each agent could accumulate experience and save it as memory during its operation, contributing to a large, collective memory pool. This would enable different agents to implicitly leverage each other's learned experiences, fostering a more collaborative and efficient learning ecosystem.

Beyond this collective pooling, a significant and promising frontier involves investigating the selective nature of such knowledge diffusion through the lens of transfer learning. As the number of agents increases, particularly beyond simple pairings, the challenge of "what to share" and "with whom to share" becomes a critical research direction. Future work could explore mechanisms that differentiate between universally applicable procedural insights and task-specific noise, ensuring that memory transfer remains contextually relevant and avoids the risks of negative transfer. Navigating these trade-offs between global collective intelligence and targeted knowledge distribution will be essential for building scalable, self-evolving swarms.

H.6 Dedicated Domains and Hybrid Architectures

While this work freezes large language models (LLMs) to avoid the risk of catastrophic forgetting, a question arises regarding highly specialized domains where the base model may not comprehend foundational terminology. Is MEMRL alone sufficient in such cases? We anticipate a future hybrid model where companies periodically fine-tune foundational models (e.g., annually) to update their core vocabulary and understanding, while simultaneously leveraging MEMRL for daily behavioral adapta-

tion and runtime learning. This approach requires the base model to possess a relatively high level of "intelligence" to initially grasp and subsequently learn new terms through interaction and feedback.

I Licenses

We list the licenses and usage terms of the main datasets, benchmarks, and environment assets used in our experiments. For proprietary LLMs and embedding models, we use the official APIs under their respective service terms.

Asset	License / Terms	Notes
BigCodeBench	Apache-2.0	HuggingFace dataset page.
ALFWorld	MIT	Official GitHub repository.
TextWorld	MIT	Dependency used by ALFWorld.
Fast Downward	GPL-3.0	Planner dependency used by ALFWorld.
HLE	MIT	HuggingFace dataset page; evaluation only, not redistributed.
LifelongAgentBench	Not explicitly specified	HuggingFace dataset page; used for OS/DB tasks.
OpenAI / Google API models	API service terms	OpenAI terms; Google Gemini API terms.

Table 20: Licenses and usage terms for the datasets, benchmarks, and environments, and for the model APIs used in our experiments.

J Case Study: High-Utility Failure Analysis (Near-Misses)

This appendix section provides qualitative case studies of high-value near-miss memories mined from the 10-epoch OS-Interaction run. CS denotes *Case Study*. Each box below contains: origin task, retrieved reflection memory, a short explanation, and the target task where it was retrieved.

Case Study 1 (CS1)

Utility. selected=19, success=19 (100.0%).

Origin task.

```
Update the SSH daemon configuration using sed to set: Port=2222,
    PermitRootLogin=no, X11Forwarding=no, ClientAliveInterval=300,
    ClientAliveCountMax=2, MaxAuthTries=2, MaxSessions=2,
    PasswordAuthentication=no, AllowUsers='user1 user2', UseDNS=no, and
    Protocol=2.
```

Retrieved reflection memory.

```
- #1: [MEMORY TYPE] FAILURE_REFLECTION
[TASK]
Update the SSH daemon configuration using sed to set: Port=2222,
    PermitRootLogin=no, X11Forwarding=no, ClientAliveInterval=300,
    ClientAliveCountMax=2, MaxAuthTries=2, MaxSessions=2,
    PasswordAuthentication=no, AllowUsers='user1 user2', UseDNS=no, and
    Protocol=2.

[REFLECTION]
- ROOT CAUSE: The `sed` command did not modify the commented lines in the
    configuration file, which is why the new settings were not applied as
    expected.
- PATTERN TO AVOID: Avoid assuming that `sed` will replace commented lines
    without explicitly handling them.
- CORRECT APPROACH: Use `sed` to either uncomment the lines before replacing
     them or add the new configurations if they are commented out.
```

Explanation. This reflection captures a subtle config-edit pitfall: naive sed replacement may miss commented defaults. The transferable lesson is to explicitly handle commented lines (uncomment-or-append), which generalizes across configuration-editing tasks.

Target task (retrieval rank 1; run outcome: correct).

```
Modify the SSH daemon configuration file at /etc/ssh/sshd_config to change
    the port to 2222, disable root login, set max sessions to 2, and set
    client alive interval to 300. Create a backup file before making changes
    .
```

Case Study 2 (CS2)

Utility. selected=20, success=20 (100.0%).

Origin task.

```
Replace 'DEBUG=True' with 'DEBUG=False' in /app/settings.cfg, append '
    LOG_LEVEL=INFO' after the modified line, and create a backup file.
```

Retrieved reflection memory.

```
- #2: [MEMORY TYPE] FAILURE_REFLECTION
[TASK]
Replace 'DEBUG=True' with 'DEBUG=False' in /app/settings.cfg, append '
    LOG_LEVEL=INFO' after the modified line, and create a backup file.

[REFLECTION]
- ROOT CAUSE: The `sed` command added the line `LOG_LEVEL=INFO` twice due to
     the way it was structured to append after the modified `DEBUG` line.
- PATTERN TO AVOID: Avoid using multiple `-e` options in `sed` that can lead
     to unintended duplications when appending lines.
- CORRECT APPROACH: Instead, use a single `sed` command that ensures the
    line is only added once, or check for the existence of the line before
    appending it.
```

Explanation. The originating attempt nearly completes the edit but fails due to a non-idempotent text transformation (duplicating an appended line). The memory teaches an idempotent editing pattern (single-pass edit or existence check), which transfers to many file-edit tasks.

Target task (retrieval rank 2; run outcome: correct).

```
Replace 'DEBUG=True' with 'DEBUG=False' in /app/settings.cfg, append '
    LOG_LEVEL=INFO' after the modified line, and create a backup file.
```

Case Study 3 (CS3)

Utility. selected=48, success=44 (91.7%).

Origin task.

```
Create a user 'testuser' with a home directory, set password expiration
    policy (max 60 days, min 7 days, warning 7 days), add to 'testgroup',
    create '/home/testuser/private' with 770 permissions, create '/shared'
    directory accessible only by 'testgroup', add 'export PATH=$PATH:/custom
    ' to .bashrc, and create '.hushlogin' file with 644 permissions.
```

Retrieved reflection memory.

```
- #4: [MEMORY TYPE] FAILURE_REFLECTION
[TASK]
Create a user 'testuser' with a home directory, set password expiration
    policy (max 60 days, min 7 days, warning 7 days), add to 'testgroup',
    create '/home/testuser/private' with 770 permissions, create '/shared'
    directory accessible only by 'testgroup', add 'export PATH=$PATH:/custom
    ' to .bashrc, and create '.hushlogin' file with 644 permissions.

[REFLECTION]
- ROOT CAUSE: The output of the commands was empty, indicating successful
    execution without any visible confirmation or errors.
- PATTERN TO AVOID: Relying solely on silent execution without feedback can
    lead to uncertainty about whether tasks were completed successfully.
- CORRECT APPROACH: Always check the status of commands using conditional
    statements or output messages to confirm successful execution and
    provide clarity on the task's completion.
```

Explanation. This memory highlights a common near-miss failure mode: assuming silent command output implies completion without verification. The transferable practice is to add explicit post-checks (exit codes, file/permission/ownership checks) after critical steps.

Target task (retrieval rank 4; run outcome: correct).

```
Create a user 'testuser' with a home directory, set password expiration
    policy (max 60 days, min 7 days, warning 7 days), add to 'testgroup',
    create '/home/testuser/private' with 770 permissions, create '/shared'
    directory accessible only by 'testgroup', add 'export PATH=$PATH:/custom
    ' to .bashrc, and create '.hushlogin' file with 644 permissions.
```

Case Study 4 (CS4)

Utility. selected=26, success=23 (88.5%).

Origin task.

```
Run a sleep process in the background for 3 seconds, log its start and end
    times (in epoch format) to '/var/log/sleep.log', store the process ID in
     '/tmp/sleep_pid', and ensure the PID file is removed after the process
    completes.
```

Retrieved reflection memory.

```
- #1: [MEMORY TYPE] FAILURE_REFLECTION
[TASK]
Run a sleep process in the background for 3 seconds, log its start and end
    times (in epoch format) to '/var/log/sleep.log', store the process ID in
     '/tmp/sleep_pid', and ensure the PID file is removed after the process
    completes.

[REFLECTION]
- ROOT CAUSE: The PID file was not created because the command to capture
    the PID may not have executed correctly, leading to multiple executions
    of the sleep process without proper PID storage.
- PATTERN TO AVOID: Avoid executing commands in a loop or without proper
    checks, which can lead to unintended repeated executions and failures to
     capture necessary outputs.
- CORRECT APPROACH: Implement a structured approach with error handling and
    verification after each critical command to ensure that each step
    completes successfully before proceeding to the next.
```

Explanation. The failure comes from missing control/verification around process management (PID capture and cleanup), which leads to repeated execution and missing artifacts. The memory transfers as a general recipe: capture PID once, verify artifacts, and sequence commands with error handling.

Target task (retrieval rank 1; run outcome: correct).

```
Run a sleep process in the background for 3 seconds, log its start and end
    times (in epoch format) to '/var/log/sleep.log', store the process ID in
     '/tmp/sleep_pid', and ensure the PID file is removed after the process
    completes.
```

Case Study 5 (CS5)

Utility. selected=94, success=88 (93.6%).

Origin task.

```
Create a secure configuration file '/etc/appconfig/settings.conf' using vi,
    set ownership to root:configgroup, permissions to 660, and add 'appuser'
     to 'configgroup'. Ensure the file contains '# Configuration file' and '
    key=value' lines.
```

Retrieved reflection memory.

```
- #2: [MEMORY TYPE] FAILURE_REFLECTION
[TASK]
Create a secure configuration file '/etc/appconfig/settings.conf' using vi,
    set ownership to root:configgroup, permissions to 660, and add 'appuser'
     to 'configgroup'. Ensure the file contains '# Configuration file' and '
    key=value' lines.

[REFLECTION]
- ROOT CAUSE: The initial assumption that the group `configgroup` and the
    user `appuser` already existed led to errors when attempting to set
    ownership and modify group membership.
- PATTERN TO AVOID: Avoid making assumptions about the existence of users or
     groups in the system without verifying first.
- CORRECT APPROACH: Always check for the existence of necessary users and
    groups before performing operations that depend on them, and create them
     if they do not exist.
```

Explanation. This reflection encodes a high-frequency near-miss: assuming required users/groups exist. The transferable heuristic is to preflight environment state (e.g., id/getent) and create prerequisites before applying ownership/permission changes.

Target task (retrieval rank 2; run outcome: correct).

```
Create a configuration file '/etc/appconfig/settings.conf' using 'vi'
    containing the lines 'SERVER_IP=192.168.1.100' and 'DEBUG_MODE=false',
    set its group ownership to 'appgroup', and permissions to 640.
```

Case Study 6 (CS6)

Utility. selected=59, success=55 (93.2%).

Origin task.

```
Create a user 'testuser', add them to group 'testgroup', configure '/data'
    with group ownership 'testgroup' and permissions 770, create '/data/
    notes.txt' using 'vi' with content 'Hello from vi', and create an
    executable script '/check.sh' using 'vi' to verify group ownership of '/
    data'.
```

Retrieved reflection memory.

```
- #5: [MEMORY TYPE] FAILURE_REFLECTION
[TASK]
Create a user 'testuser', add them to group 'testgroup', configure '/data'
    with group ownership 'testgroup' and permissions 770, create '/data/
    notes.txt' using 'vi' with content 'Hello from vi', and create an
    executable script '/check.sh' using 'vi' to verify group ownership of '/
    data'.

[REFLECTION]
- ROOT CAUSE: The output being empty does not indicate a failure, but rather
     that the commands executed successfully without any output.
- PATTERN TO AVOID: Assuming that an empty output signifies an error can
    lead to misunderstandings about command execution results.
- CORRECT APPROACH: Always verify the success of commands by checking their
    exit status or using logging to confirm that actions were completed as
    intended.
```

Explanation. The lesson is that empty output usually signals success, not failure; therefore success should be validated via exit status and direct state checks. This reduces false negatives across shell automation tasks.

Target task (retrieval rank 5; run outcome: correct).

```
Create a file '/shared/config/settings.conf' containing 'USER=testuser' and
    'GROUP=testgroup', then programmatically create a user and group using
    the values from the file. Finally, ensure '/shared/data' is owned by the
     group with 770 permissions.
```

K Prompts

We provide the exact prompt strings and message templates used by our MEMRL implementation across all benchmarks. To minimize ambiguity, we separate prompts used to summarize experiences into memories from prompts used at task time for generation/inference.

K.1 Experience Summarization Prompts

HLE: Experience Summarization Prompts

Trajectory serialization (stored as the episode trajectory).

```
QUESTION
{question}

SOLUTION
{model_output_stripped}
```

High-level script generation prompt.

```
Analyze the following detailed task trajectory and create a concise,
high-level script that captures the essential steps and decision points.

The script should be:
1. Generic enough to apply to similar tasks
2. Specific enough to provide useful guidance
3. 3-5 high-level steps maximum
4. Focus on the strategy and key decisions, not detailed actions

Trajectory:
{trajectory}

High-level script:
```

Failure reflection prompt.

```
Task: {task_description}

Failed trajectory:
{failed_trajectory}

This task failed. Analyze what went wrong and suggest improvements for
    future similar tasks.
Focus on:
1. Incorrect assumptions
2. Steps to improve
3. What to avoid next time

Provide a brief reflection:
```

Stored memory content templates.

```
# Successful memory
Task: {task_description}

SCRIPT:
{script}

TRAJECTORY:
{trajectory}

# Failure memory
TASK REFLECTION:
Task: {task_description}

What went wrong:
{reflection}

Failed approach:
{failed_trajectory}
```

ALFWorld: Experience Summarization Prompts

Trajectory serialization (stored as the episode trajectory).

```
The full ALFWorld dialogue history is stored as the trajectory:
  List[{"role": ..., "content": ...}, ...]
When interpolated into the summarization prompts, it is treated as a string
    representation.
```

High-level script generation prompt.

```
Analyze the following detailed task trajectory and create a concise,
high-level script that captures the essential steps and decision points.

The script should be:
1. Generic enough to apply to similar tasks
2. Specific enough to provide useful guidance
3. 3-5 high-level steps maximum
4. Focus on the strategy and key decisions, not detailed actions

Trajectory:
{trajectory}

High-level script:
```

Failure reflection prompt.

```
Task: {task_description}

Failed trajectory:
{failed_trajectory}

This task failed. Analyze what went wrong and suggest improvements for
    future similar tasks.
Focus on:
1. Incorrect assumptions
2. Steps to improve
3. What to avoid next time

Provide a brief reflection:
```

Stored memory content templates.

```
# Successful memory
Task: {task_description}

SCRIPT:
{script}

TRAJECTORY:
{trajectory}

# Failure memory
TASK REFLECTION:
Task: {task_description}

What went wrong:
{reflection}

Failed approach:
{failed_trajectory}
```

BCB (BigCodeBench): Experience Summarization Prompts

Trajectory serialization (stored as the episode trajectory).

```
[BCB] epoch={epoch} phase={phase} task_id={task_id}
[PROMPT]
{bcb_task_prompt}
[GENERATED CODE]
```python
{model_code}
```
[EVAL]
{"status": "...", "error": "..."}
```

High-level script generation prompt.

```
Analyze the following detailed task trajectory and create a concise,
high-level script that captures the essential steps and decision points.

The script should be:
1. Generic enough to apply to similar tasks
2. Specific enough to provide useful guidance
3. 3-5 high-level steps maximum
4. Focus on the strategy and key decisions, not detailed actions

Trajectory:
{trajectory}

High-level script:
```

Failure reflection prompt.

```
Task: {task_description}

Failed trajectory:
{failed_trajectory}

This task failed. Analyze what went wrong and suggest improvements for
    future similar tasks.
Focus on:
1. Incorrect assumptions
2. Steps to improve
3. What to avoid next time

Provide a brief reflection:
```

Stored memory content templates.

```
# Successful memory
Task: {task_description}

SCRIPT:
{script}

TRAJECTORY:
{trajectory}

# Failure memory
TASK REFLECTION:
Task: {task_description}

What went wrong:
{reflection}

Failed approach:
{failed_trajectory}
```

LLB (LifelongAgentBench): Experience Summarization Prompts

Trajectory serialization (stored as the episode trajectory).

```
{role_1}: {content_1}
{role_2}: {content_2}
...
```

High-level script generation prompt.

```
Analyze the following detailed task trajectory and create a concise,
high-level script that captures the essential steps and decision points.

The script should be:
1. Generic enough to apply to similar tasks
2. Specific enough to provide useful guidance
3. 3-5 high-level steps maximum
4. Focus on the strategy and key decisions, not detailed actions

Trajectory:
{trajectory}

High-level script:
```

Failure reflection prompt.

```
Task: {task_description}

Failed trajectory:
{failed_trajectory}

This task failed. Analyze what went wrong and suggest improvements for
    future similar tasks.
Focus on:
1. Incorrect assumptions
2. Steps to improve
3. What to avoid next time

Provide a brief reflection:
```

Stored memory content templates.

```
# Successful memory
Task: {task_description}

SCRIPT:
{script}

TRAJECTORY:
{trajectory}

# Failure memory
TASK REFLECTION:
Task: {task_description}

What went wrong:
{reflection}

Failed approach:
{failed_trajectory}
```

K.2 Generation and Inference Prompts

HLE: Generation and Inference Prompts

System prompt (exact-match).

```
Your response should be in the following format:
Explanation: {your explanation for your final answer}
Exact Answer: {your succinct, final answer}
Confidence: {your confidence score between 0% and 100% for your answer}
```

System prompt (multiple-choice).

```
Your response should be in the following format:
Explanation: {your explanation for your answer choice}
Answer: {your chosen answer}
Confidence: {your confidence score between 0% and 100% for your answer}
```

Retrieved memory injection (system message).

```
=== Successful Memories ===
{memory_full_content_1}

{memory_full_content_2}

=== Failed Memories (for caution) ===
{memory_full_content_3}
...
```

Optional repeat-attempt reflection note (system message).

```
You attempted this question before.
Result: {CORRECT|INCORRECT}
Question: {question}
Previous attempt (solution only):
{solution_only}
Reflect on mistakes or gaps, then solve the problem again with a better
    solution.
```

User message text block (when images exist).

```
Now solve the following question:

[Image IDs: {question_image_ids}]
{question}

Attached images:
1. [{img_id_1}] ({source_1})
2. [{img_id_2}] ({source_2})
...
```

Message ordering.

```
1) system: exact-match OR multiple-choice format prompt
2) system: optional reflection note (if enabled)
3) system: optional retrieved memory context
4) user: question content (text + optional images)
```

ALFWorld: Generation and Inference Prompts

Base system prompt (ReAct format + action space).

```
Interact with a household to solve a task. Imagine you are an intelligent
    agent in a household environment and your target is to perform actions
    to complete the task goal. At the beginning of your interactions, you
    will be given the detailed description of the current environment and
    your goal to accomplish.
For each of your turn, you will be given the observation of the last turn.
    You should first think about the current condition and plan for your
    future actions, and then output your action in this turn. Your output
    must strictly follow this format:"Thought: your thoughts.\nAction: your
    next action".

The available actions are:
1. go to {recep}
2. take {obj} from {recep}
3. move {obj} to {recep}
4. open {recep}
5. close {recep}
6. use {obj}
7. clean {obj} with {recep}
8. heat {obj} with {recep}
9. cool {obj} with {recep}
where {obj} and {recep} correspond to objects and receptacles.
After your each turn, the environment will give you immediate feedback based
     on which you plan your next few steps. if the envrionment output "
    Nothing happened", that means the previous action is invalid and you
    should try more options.

Your response should use the following format:

Thought: <your thoughts>
Action: <your next action>
```

Retrieved memory injection (system message).

```
In addition to the example, you have the following memories from your own
    past experiences. Use them to help you if they are relevant:

--- SUCCESSFUL MEMORIES (Examples to follow) ---
{formatted_success_memories_joined}

--- FAILED MEMORIES (Examples to avoid or learn from) ---
{formatted_failed_memories_joined}
```

Current task prompt (user message).

```
Now, it's your turn to solve a new task.
{task_description}
```

Per-step observation prompt (user message).

```
Observation: {observation}
```

Message ordering (high level).

```
1) system: base ALFWorld system prompt
2) user/assistant: selected few-shot example dialogue (sequence of messages)
3) system: optional retrieved memory context
4) user: new task prompt
5) loop: append user Observation: ..., model replies with Thought/Action
```

BCB (BigCodeBench): Generation and Inference Prompts

Retrieved memory injection (system message).

```
[Retrieved Memory Context]

### Memory 1 (id={mem_id_1}, sim={similarity_1})
{memory_content_1}

### Memory 2 (id={mem_id_2}, sim={similarity_2})
{memory_content_2}
...
```

Dataset-provided task prompt selection (user message).

```
if split == "instruct":
    return task["instruct_prompt"]
if split == "complete":
    return task["complete_prompt"]
```

Message ordering.

```
1) system: optional [Retrieved Memory Context] ...
2) user: {bcb_task_prompt}
```

LLB (LifelongAgentBench): Generation and Inference Prompts

Base system prompt.

```
You are an execution-focused AI agent solving database and operating-system
    tasks.

You may receive a [Retrieved Memory Context] block with past experiences
    from similar problems.
These are **references for learning**, not guaranteed solutions:
- [MEMORY TYPE] SUCCESS_PROCEDURE: A successful approach from a similar task
     learn the pattern.
- [MEMORY TYPE] FAILURE_REFLECTION: A failed attempt with lessons avoid
    similar mistakes.

Use the memories as inspiration, but always analyze your current task
    independently and
adapt your approach based on its specific requirements.
```

Strict output constraint (DB tasks).

```
STRICT OUTPUT FORMAT (LLB:DB, do not violate):
1) After your reasoning, include exactly ONE action line:
   - Action: Operation
   - Action: Answer
2) If Action: Operation, put exactly ONE SQL statement in the FIRST fenced
    code block using ‘‘‘sql, on a single line. Do not add any extra text
    after that block.
3) If Action: Answer, include ‘Final Answer: ...‘ on the next line and do
    not add extra text after that.
```

Strict output constraint (OS tasks).

```
STRICT OUTPUT FORMAT (LLB:OS, do not violate):
1) After your reasoning, include exactly ONE action line:
   - Act: bash
   - Act: finish
2) If Act: bash, the next lines MUST be a ‘‘‘bash fenced code block with
    your Bash commands. Do not include any other code blocks.
3) If Act: finish, it must be the last line (no code blocks, no extra text).
4) Do NOT use ‘Action:‘ in OS tasks (use ‘Act:‘ only).
```

Retrieved memory injection block.

```
[Retrieved Memory Context]

=== SUCCESSFUL EXPERIENCES (Learn from these) ===
[SUCCESS 1] [TYPE: {mem_type}]
{content}

=== FAILED EXPERIENCES (Avoid these mistakes) ===
[FAILURE 1] [TYPE: {mem_type}]
{content}
```

Prompt assembly ordering (system prompt).

```
1) base system prompt
2) optional [Retrieved Memory Context] ...
3) strict output format block appended at the end (task-aligned)
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All claims accurately follow the paper contributions.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations have been clearly discussed in the paper. Computational efficiency can be found in Appendix F.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Theory assumptions and proofs are given in the main text and Appendix A.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We follow established training procedures as stated in the main text. Code is included in the supplemental material to replicate various experiments. Training details and hyperparameters are reported in Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in the supplemental material?

Answer: [Yes]

Justification: Code is publicly available at `https://github.com/MemTensor/MemRL`; code and data are also provided in the supplemental material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: We report the experimental details in Appendix E.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: The four main methods (No Memory, RAG, MemP, MEMRL) are repeated for three run seeds (42, 43, 44) on BigCodeBench, Lifelong Agent Bench (OS and DB), and ALFWorld. Tables 1 and 2 report the means of those runs; the corresponding standard deviations are in Table 15 (Appendix G.1). Aggregate confidence intervals for the difference between MEMRL and the strongest baseline are reported in Table 17. HLE uses stochastic decoding at temperature 0.6; repeated full-scale runs exceeded our API budget, so HLE entries are single runs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the API-based LLM backbones, decoding settings, and runtime cost analysis in the Appendix E and Appendix F. Our experiments do not require local GPU training; the main compute comes from LLM API calls, while retrieval and scalar utility updates introduce negligible local overhead.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research follows the NeurIPS Code of Ethics. It uses public benchmarks and API-based models, does not involve human subjects or private user data, and discusses relevant risks such as memory security and robustness in the limitations.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: The paper discusses potential positive impacts, such as efficient frozen-backbone adaptation, and potential risks of self-evolving memory systems, including memory pollution, reward hacking, and robustness concerns. These are primarily technical safety risks, but may have broader implications when agent memories are deployed in real-world workflows. Details can be found in Appendix H

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk pretrained models, image generators, or scraped datasets. The method is evaluated on existing benchmarks and uses external LLM APIs.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original papers and sources for all datasets, benchmarks, and environment assets used in our experiments. Their licenses or usage terms are summarized in Appendix I; proprietary LLMs are accessed through official APIs under their service terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide the implementation of MEMRL as supplementary material for review, including code and instructions needed to reproduce the main experiments.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This paper involves neither crowdsourcing experiments nor any research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject studies, so IRB approval is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: LLMs are central to the method and experiments: frozen LLMs serve as the agent backbones for generation and reasoning, and are also used to summarize trajectories into episodic memories. The paper reports the model configurations and prompts used in Appendix E and Appendix K.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.