MemSkill: Learning and Evolving Memory Skills for Self-Evolving Agents

Haozhen Zhang[1] **Quanyu Long**[1] **Jianzhu Bao**[1] **Tao Feng**[2]
Weizhi Zhang[3] **Haodong Yue**[4] **Wenya Wang**[1]
[1]Nanyang Technological University [2]University of Illinois Urbana-Champaign
[3]University of Illinois Chicago [4]Tsinghua University
haozhen001@e.ntu.edu.sg wangwy@ntu.edu.sg

Abstract

Most Large Language Model (LLM) agent memory systems rely on a small set of static, hand-designed operations for extracting memory. These fixed procedures hard-code human priors about what to store and how to revise memory, making them rigid under diverse interaction patterns and inefficient on long histories. To this end, we present **MemSkill**, which reframes these operations as learnable and evolvable memory skills, structured and reusable routines for extracting, consolidating, and pruning information from interaction traces. Inspired by the design philosophy of agent skills, MemSkill employs a *controller* that learns to select a small set of relevant skills, paired with an LLM-based *executor* that produces skill-guided memories. Beyond learning skill selection, MemSkill introduces a *designer* that periodically reviews hard cases where selected skills yield incorrect or incomplete memories, and evolves the skill set by proposing refinements and new skills. Together, MemSkill forms a closed-loop procedure that improves both the skill-selection policy and the skill set itself. Experiments on LoCoMo, LongMemEval, HotpotQA, and ALFWorld demonstrate that MemSkill improves task performance over strong baselines and generalizes well across settings. Further analyses shed light on how skills evolve, offering insights toward more adaptive, self-evolving memory management for LLM agents. Code is available at https://github.com/ViktorAxelsen/MemSkill

1 Introduction

As Large Language Model (LLM) agents engage in longer, open-ended interactions, they must handle growing histories that are essential yet challenging to leverage, motivating memory for retaining experience and maintaining coherence [6]. This need has driven rapid progress in agent memory, including approaches that summarize and retrieve past interactions or manage external memory stores [8, 2, 15, 25]. However, most methods still rely on static, hand-designed memory mechanisms, including fixed operation primitives (e.g., add/update/delete/skip) [19, 26] and heuristic modules that govern what to store, how to revise it [8, 3], and when to prune it. Such designs bake in strong human assumptions and often suffer under diverse interaction patterns, scaling poorly as histories grow.

We argue that this formulation fundamentally limits the adaptability of agent memory. Rather than treating memory as the output of fixed operations or hand-designed modules, we propose to elevate memory extraction itself into a *learnable abstraction*. Concretely, we view memory construction as the outcome of applying a small set of generic, reusable *memory skills*: structured behaviors that specify when and how interaction traces should be transformed into memory and revised over time. This perspective reveals a key bottleneck of prior pipelines: they hard-code memory behaviors into fixed procedural workflows that interleave heuristics with LLM-mediated extraction and revision, making them brittle under distribution shift [3].

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

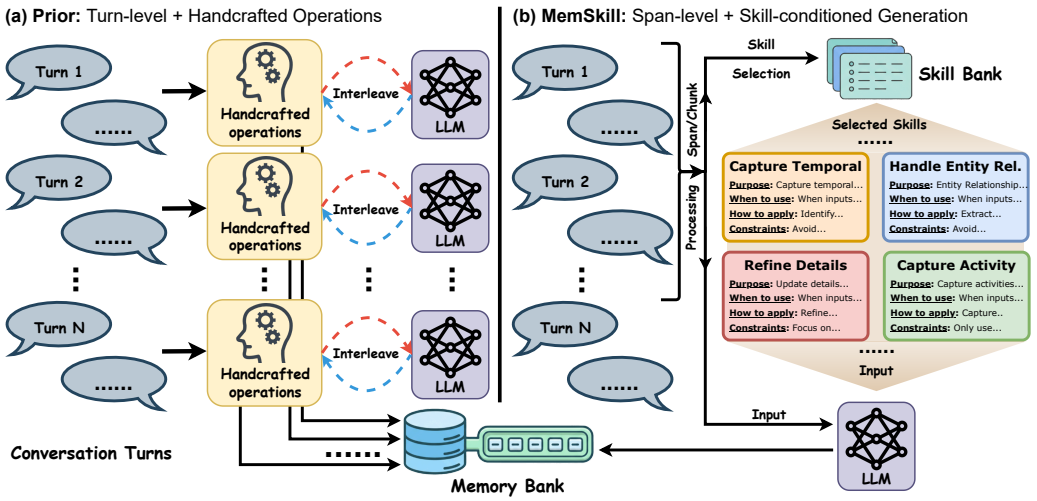

Figure 1: **Comparison between (a) prior turn-level, handcrafted operations and (b) MemSkill's span-level, skill-conditioned generation.** Prior methods interleave handcrafted operations with LLM calls to incrementally extract and revise memory turn by turn, while MemSkill selects a small set of skills from a shared skill bank and applies them in one pass to produce skill-guided memories.

Under this view, an ideal agent memory system should satisfy three properties. (i) *Minimal reliance on human priors.* Instead of manually encoding what is worth remembering for a domain [38], memory behaviors should be shaped by interaction data and updated as task demands evolve. (ii) *Support for larger extraction granularity.* Many approaches are tuned to a fixed unit, such as per-turn processing [3], and can weaken when applied to longer spans. A practical system should be able to operate at larger extraction granularity when needed. (iii) *Skill-conditioned, compositional memory construction.* Existing systems often decompose memory construction into specialized modules [8]. In contrast, we prefer to *select and compose* a small set of relevant skills for the current context and apply them in one generation step, enabling flexible reuse and evolution of memory behaviors.

Based on the above observations, we introduce **MemSkill**, which reframes memory operations as a learnable and evolvable set of memory skills. MemSkill maintains a shared *skill bank*, where each skill captures a reusable way to extract, consolidate, or revise memories from interaction text (Figure 1 shows the template of a memory skill). Given the current context, a *controller* learns to select a small set of relevant skills, and an LLM-based *executor* conditions on these skills to generate skill-guided memories in one pass. This skill-conditioned formulation is not tied to a fixed extraction unit and can be applied to different span lengths when processing long interaction histories.

Crucially, MemSkill goes beyond learning how to use a fixed set of skills. We introduce a closed-loop evolution process that alternates between learning to use the current skill bank and evolving the skill bank itself. Specifically, we train the *controller* with reinforcement learning (RL) using downstream task signals as feedback for skill selection. Periodically, a *designer* aggregates the hardest cases produced during training, selects representative failures, and uses an LLM to refine existing skills and propose new ones. After each evolution step, the controller continues training on the evolved skill bank, with additional exploration to facilitate adopting newly introduced skills. Overall, this process gradually strengthens both the skill selection policy and the evolving skill bank, moving toward a more adaptive memory management system driven by interaction data.

Experiments on LoCoMo, LongMemEval, HotpotQA, and ALFWorld show that MemSkill consistently improves task performance and generalizes well. Further analyses validate key components and showcase representative evolved skills, offering insights toward more adaptive, self-evolving memory management for LLM agents. **Our contributions can be summarized as follows.**

- We propose **MemSkill**, an agent memory method that represents memory operations as an evolving skill bank, where each skill provides reusable guidance for selecting, extracting, and organizing useful memories. This turns memory construction from a fixed handcrafted pipeline into an adaptive skill-conditioned generation process.

- We introduce a closed-loop optimization recipe that combines reinforcement learning for skill selection with LLM-guided skill evolution from hard cases, enabling continual refinement of the skill bank and taking a step toward self-evolving agent memory systems.
- We evaluate MemSkill on LoCoMo, LongMemEval, HotpotQA, and ALFWorld, demonstrating consistent gains and strong transfer ability across conversational QA and embodied interaction settings, offering insights for self-evolving memory in LLM agents.

2 Related Work

2.1 LLM Agent Memory Systems

Prior work on agent memory focuses on constructing external memories from interaction histories and leveraging them to support downstream reasoning and decision making. Typical pipelines periodically extract salient information into a memory store, retrieve relevant entries for a new query, and update the store via consolidation or pruning [8, 38, 25, 15, 2, 3]. More recently, learning-based approaches including Memory-R1 [26] and Mem-α [19] optimize memory management with reinforcement learning (RL) using downstream task signals. Similarly, AgeMem [31] employs RL to manage short- and long-term memory through predefined operations. Despite this progress, memory management is still largely governed by static, hand-crafted routines for extraction, consolidation, and pruning.

Several concurrent works also explore self-evolving memory in agent settings, but differ from our focus. Evo-Memory [22] evaluates streaming memory evolution, MemEvolve [34] optimizes predefined memory architectures, MemGen [33] targets latent memory for reasoning, and ReasoningBank [14] distills reasoning strategies from experience. By contrast, we target the evolution of memory skills themselves, enabling the system to refine and grow reusable memory operations over time.

2.2 Self-Evolving LLM Agents

Recent work on self-evolving LLM agents studies how agents can improve from interaction experience with minimal manual supervision. ExpeL [35] distills trajectories into editable natural-language insights and retrieves relevant experiences to guide future decisions, while EvolveR [24] formalizes an experience lifecycle that consolidates interactions into reusable principles and closes the loop with reinforcement learning updates. A complementary line reduces reliance on curated data via self-play style curricula: Absolute Zero Reasoner [36] trains a proposer and solver with verifiable rewards from a code executor, and Multi-Agent Evolve [1] extends this to a proposer solver judge triad with LLM-based evaluation; R-Zero [7] follows a similar challenger solver co-evolution pattern. Beyond curricula, systems such as AgentEvolver [32] and RAGEN [20] study efficient agent learning dynamics and stabilization in multi-turn RL settings, while ADAS [5] and AlphaEvolve [13] explore automated discovery and evolutionary improvement of agent designs. Finally, SkillWeaver [37] shows that agents can discover and refine reusable skills for web interaction. In contrast, our focus is on self-evolving *memory skills* that govern how agents construct and revise memories over time.

3 Method

In this section, we first overview MemSkill (Section 3.1), followed by the *skill bank* (Section 3.2) and three core components: *controller* (Section 3.3.1), *executor* (Section 3.3.2), and *designer* (Section 3.4). We then describe the closed-loop optimization of skill selection and evolution (Section 3.5).

3.1 Overview

As shown in Figure 2, we propose **MemSkill**, which optimizes agent memory through two intertwined processes. First, it **learns to use a given skill bank**: a controller selects context-relevant skills, and an executor applies them to produce memory updates. Second, it **improves the skill bank itself**: a designer periodically revises existing skills and adds new ones based on challenging training cases.

To disentangle trace-specific memories from reusable memory management knowledge, MemSkill maintains two stores. The *memory bank* is trace-specific and stores memories for each training trace (e.g., a long dialogue). In contrast, the *skill bank* is shared across traces and contains reusable

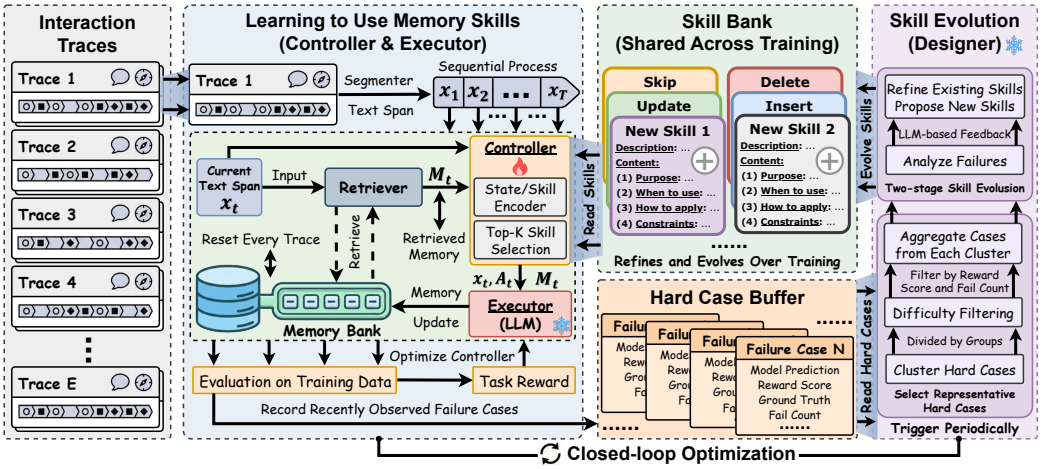

Figure 2: **MemSkill architecture overview.** Given an interaction trace, MemSkill processes it span by span: the controller selects a Top-K subset of skills from a shared *skill bank* conditioned on the current text span and retrieved memories, and an LLM executor applies the selected skills in one pass to update the trace-specific *memory bank*. The constructed memory is then evaluated on memory-dependent training queries to provide task reward for optimizing the controller, while query-centric failures are logged into a sliding hard-case buffer. Periodically, the designer mines representative hard cases to refine existing skills and propose new ones, yielding alternating phases of skill usage and skill evolution. More skill case study can be found in Section 4.5 and Appendix C.

memory skills. During training, the controller and executor build each trace's memory bank, while the designer updates the shared skill bank between phases. This alternating procedure gradually improves both the skill selection policy and the skill bank for memory construction.

3.2 Skill Bank

As shown in Figure 2, a *memory skill* specifies a reusable memory operation as structured guidance, including when it is applicable and how it should be applied to the current context. Concretely, each skill $s \in \mathcal{S}$ contains (i) a short *description* for skill representation and selection, and (ii) a detailed *content* specification that instructs the executor on memory extraction or revision.

We start from a minimal set of general-purpose primitives to ensure a stable and functional initialization. Specifically, we initialize the skill bank with four basic skills corresponding to canonical memory operations: INSERT, UPDATE, DELETE, and SKIP. Starting from this minimal set, the designer progressively refines existing skills and expands the bank by proposing new skills that address uncovered failure modes. (Appendix C details skill description)

3.3 Learning to Use Memory Skills

In this part, we describe how MemSkill learns to use memory skills, covering (i) the skill-selection policy and (ii) skill-conditioned memory construction.

3.3.1 Controller: Skill Selection Policy

To enable effective skill selection as the *skill bank* evolves, we introduce a controller that selects a small set of relevant memory skills for the current context. At each memory construction step, **we update memory at the span level**: we split each interaction trace (e.g., a dialogue) into fixed-length contiguous spans by token count and process them sequentially. For each span, the controller conditions its selection on (i) the current text span and (ii) the retrieved existing memories from the current trace's memory bank (empty for the initial span), rather than operating turn by turn.

To remain compatible with a variable-size skill bank as it continuously evolves, the controller scores each skill by measuring the semantic distance between the current state and skill representations, naturally supporting a changing skill set while staying sensitive to what is already stored in memory.

State and skill representations. Formally, let x_t denote the current text span at step t, and let $M_t = \{m_{t,1}, \ldots, m_{t,R}\}$ be the retrieved memories from the current trace's memory bank. We first encode x_t and M_t with a fixed embedding model $e(\cdot)$, aggregate the retrieved memory embeddings by element-wise averaging, and concatenate it with the span embedding:

$$h_t = f_\theta^{\text{ctx}}([e(x_t); \bar{m}_t]), \qquad u_i = f_\theta^{\text{skill}}(e(\text{desc}(s_i))), \tag{1}$$

where $\bar{m}_t = \frac{1}{R}\sum_{r=1}^{R} e(m_{t,r})$ denotes the aggregated memory embedding, and is omitted when M_t is empty. Here, u_i is the representation of skill $s_i \in \mathcal{S}_t$, computed from its description as a compact and stable semantic signal rather than the full skill content. The embedding model $e(\cdot)$ is shared and fixed, while f_θ^{ctx} and f_θ^{skill} are trainable neural networks in the controller for learnable skill selection.

Compatibility with an evolving skill bank. Instead of producing a fixed-dimensional action head tied to a fixed number of skills, the controller concatenates the state representation with each candidate skill representation and applies a shared scorer to all such state-skill pairs in parallel:

$$z_{t,i} = f_\theta^{\text{score}}([h_t; u_i]), \qquad p_\theta(i \mid h_t) = \text{softmax}(z_t)_i, \tag{2}$$

where f_θ^{score} is a trainable neural network, and $z_t \in \mathbb{R}^{|\mathcal{S}_t|}$ adapts as the skill bank evolves.

Top-K skill selection. Given the categorical distribution $p_\theta(i \mid h_t)$ over the current skill bank $\mathcal{S}_t$, the controller selects an ordered Top-K set of skills $A_t = (a_{t,1}, \ldots, a_{t,K})$ (e.g., via Gumbel-Top-K [9]), and only passes the selected skills to the executor, keeping the skill context concise and relevant.

3.3.2 Executor: Skill-Conditioned Memory Extraction

Given the selected skills A_t, the fixed executor constructs memory updates by conditioning an LLM on (i) the current text span x_t, (ii) the retrieved memory items M_t, and (iii) the selected skills A_t. This mirrors skill-conditioned inference in agent systems, where a small set of relevant skills is provided to guide behavior for the current context. The executor produces structured memory updates, which are parsed and applied to the trace's memory bank. By composing several skills for the same text span and extracting memory in one LLM call, MemSkill reduces repeated per-turn processing and scales better to long interaction histories. Appendix D details the executor prompt.

3.3.3 Controller Optimization

We train the controller with reinforcement learning, using downstream task performance as feedback for its skill selections. For each training trace, the controller makes a sequence of Top-K selections while the executor incrementally builds the trace-specific memory bank. After construction, we evaluate the resulting memory bank on the trace's memory-dependent training queries and use the resulting task performance as the reward (e.g., F1 or success rate).

A key technical detail is that the controller's action is an ordered Top-K *list* selected without replacement, rather than a single discrete action. We therefore compute the joint log-probability $\log \pi_\theta(A_t \mid h_t)$ under this selection process and use it in standard policy-gradient objectives [16] via importance weighting and clipping. Concretely, for $A_t = (a_{t,1}, \ldots, a_{t,K})$, the joint probability is

$$\pi_\theta(A_t \mid h_t) = \prod_{j=1}^{K} \frac{p_\theta(a_{t,j} \mid h_t)}{1 - \sum_{\ell<j} p_\theta(a_{t,\ell} \mid h_t)}, \tag{3}$$

which reduces to the single-action case when $K = 1$. Appendix B.4 gives implementation details.

3.4 Skill Evolution through Designer Feedback

Beyond learning to select from a fixed set of skills, MemSkill evolves the skill bank using an LLM-based designer (fixed) that operates periodically during training.

Hard-case buffer. During controller training, we maintain a sliding-window buffer of challenging cases observed recently. Each case is query-centric, recording the query along with its ground-truth and metadata (e.g., retrieved memories and model prediction), as well as summary statistics such as task performance and the number of failures observed so far. The buffer uses two expiration rules: cases are removed if they become too old (exceeding a maximum training step gap) or if the buffer reaches its capacity limit, which tracks recent failure patterns without growing unbounded.

Selecting representative hard cases. To focus designer updates on impactful failures, we cluster cases (e.g., KMeans) into groups that naturally reflect different query or error types. Within each cluster, we prioritize representative cases using a difficulty score that increases when task performance is low and when the same case fails repeatedly. This produces a compact set of high-value cases for skill evolution while preserving diversity across error types.

Two-stage skill evolution. The designer updates the skill bank in two stages. First, it employs an LLM to analyze the selected hard cases and identify what memory behaviors are missing or mis-specified. Second, it uses the resulting analysis to propose concrete edits to existing skills and to introduce new skills. We keep the designer description concise and provide prompts in Appendix D.

Notably, we maintain snapshots of the best-performing skill bank and roll back if an update degrades performance, with early stopping when repeated designer updates fail to improve the training signal. After each evolution step, we also briefly increase exploration by biasing selection toward newly introduced skills, encouraging the controller to try them and facilitating efficient learning of their utility. Due to page limit, more details about the designer can be found in Appendix B.2.

3.5 Closed-Loop Optimization

MemSkill alternates between (i) learning to select and apply skills to build memory banks and (ii) evolving the skill bank from hard cases mined during training. Each cycle begins with controller training on the current skill bank, where the executor constructs memories and accumulates challenging cases. The designer then updates the skill bank using representative hard cases, optionally rolling back to a prior snapshot if performance regresses. The next cycle resumes controller training on the updated skill bank, with additional exploration to encourage early use of new skills. Over cycles, this closed loop gradually improves how skills are selected, applied, and refined for memory construction.

4 Experiments

4.1 Experiment Setup

Datasets and Baselines. We evaluate MemSkill on four benchmarks: LoCoMo [12], LongMemEval [23], HotpotQA [28], and ALFWorld [17], where HotpotQA is used in Section 4.3 to study skill transfer under distribution shift. The remaining three benchmarks cover two representative settings. (i) *Conversational Benchmarks* include LoCoMo and LongMemEval, which evaluate memory construction from long, dialogue-style interaction histories. For these datasets, we report F1-score (F1) and an LLM-based judge score (L-J). (ii) *Embodied Interactive Tasks* are evaluated on ALFWorld with two standard subsets, ALF-Seen and ALF-Unseen, and we report success rate (SR) and the number of environment interaction steps (#Stps). Appendix B.1 provides dataset splits.

We compare MemSkill against several strong baselines: (1) **No-Memory**, which answers directly without an external memory; (2) **CoN** [30]; (3) **ReadAgent** [11]; (4) **MemoryBank** [38]; (5) **A-MEM** [25]; (6) **Mem0** [2]; (7) **LangMem** [10]; and (8) **MemoryOS** [8]. Overall, this setup spans diverse benchmarks and baselines, enabling a broad and consistent comparison across diverse settings.

Implementation Details. We instantiate f_θ^{ctx}, $f_\theta^{\mathrm{skill}}$, and $f_\theta^{\mathrm{score}}$ as independent lightweight multilayer perceptrons (MLPs), and use LLaMA-3.3-70B-Instruct [4] and Qwen3-Next-80B-A3B-Instruct [27] as the base LLMs, accessed through an API service. Unless otherwise specified, we train MemSkill on LLaMA and use Qwen only for transfer experiments. LongMemEval is also evaluated in a transfer setting, where we directly apply the skills learned on LoCoMo without further training.

During training, we optimize the controller with PPO [16]. MemSkill constructs memory at the span level; on conversational benchmarks, each dialogue session is a processing unit, and the controller selects $K{=}3$ skills per unit. We instantiate $e(\cdot)$ with Qwen3-Embedding-0.6B [27], also used as the memory retriever, and retrieve up to 20 memory items for MemSkill and all baselines for consistency. For the designer, skill evolution is triggered every 100 training steps, with at most 3 skill edits per evolution round. For ALFWorld, we cap environment interactions at 50 steps.

At evaluation time, we keep the same span-level formulation and set the span/chunk size to 512 by default, while keeping the overall procedure unchanged. Unless otherwise specified, we use $K{=}7$ skills for LoCoMo and LongMemEval at evaluation time, and $K{=}5$ for ALFWorld. Additional implementation details and prompt templates are provided in Appendix B and Appendix D.

Table 1: **Main comparison results on LoCoMo, LongMemEval, and ALFWorld.**

Model	Methods	Conversational Benchmarks					Embodied Interactive Tasks				
		LoCoMo		▲LongMemEval		Avg.	ALF-Seen†		ALF-Unseen†		Avg.
		F1	L-J	F1	L-J	L-J	SR	#Stps↓	SR	#Stps↓	SR
LLaMA3.3 70B-Instruct	No-Memory	-	-	-	-	-	62.14	26.10	73.88	21.36	68.01
	CoN	30.86	41.72	30.78	56.44	49.08	75.00	19.15	80.60	17.38	77.80
	ReadAgent	28.63	38.25	24.48	42.62	40.44	62.86	26.14	71.64	22.88	67.25
	MemoryBank	36.80	44.43	30.56	41.96	43.20	60.71	28.23	66.42	24.64	63.57
	A-MEM	39.39	49.71	25.83	38.04	43.88	62.86	27.53	70.15	23.79	66.51
	Mem0	25.48	34.58	30.25	46.81	40.70	74.29	19.77	81.34	17.15	77.82
	LangMem	30.91	35.82	18.36	24.35	30.09	72.86	21.25	79.85	18.30	76.36
	MemoryOS	41.39	48.64	17.59	39.83	44.24	57.86	27.94	65.67	24.46	61.77
	MemSkill	**44.21**	**53.82**	**31.12**	**60.89**	**57.36**	**77.14**	**18.91**	**83.58**	**16.63**	**80.36**
▲Qwen3-Next 80B-A3B-Instruct	No-Memory	-	-	-	-	-	63.57	24.61	60.45	26.57	62.01
	CoN	38.46	50.96	**29.19**	44.06	47.51	77.14	17.59	70.90	20.74	74.02
	ReadAgent	25.89	34.26	24.13	42.25	38.26	73.57	20.21	65.67	23.28	69.62
	MemoryBank	29.56	44.15	8.45	26.37	35.26	63.57	23.68	52.24	29.01	57.91
	A-MEM	36.43	50.30	13.84	36.59	43.45	55.71	27.42	54.48	28.60	55.10
	Mem0	23.29	33.68	27.36	46.20	39.94	71.43	19.89	64.93	23.32	68.18
	LangMem	28.17	32.94	18.35	23.86	28.40	73.57	19.76	64.18	23.58	68.88
	MemoryOS	39.86	47.37	15.97	39.25	43.31	62.14	25.64	50.75	30.35	56.45
	MemSkill	**42.08**	**54.14**	25.29	**60.40**	**57.27**	**85.71**	**13.84**	**76.87**	**18.16**	**81.29**

Bold indicates the best score within each base model block; ▲ indicates transfer evaluation only.
† indicates evaluation with in-context demonstrations. Appendix A reports more baselines and datasets.

4.2 Comparison Experiments

Effectiveness across conversational and embodied settings. Table 1 summarizes the main comparison results on LoCoMo, LongMemEval, and ALFWorld. Across these datasets, MemSkill achieves the strongest overall performance among all compared methods. On conversational benchmarks, MemSkill attains the best LLM-judge scores on both LoCoMo and LongMemEval within each base-model block, indicating higher-quality constructed memories. In comparison, prior methods such as MemoryBank, A-MEM, and MemoryOS use fixed, manually specified memory procedures for extraction and revision, whereas MemSkill learns and evolves its skills from interaction, enabling better adaptation across contexts. On ALFWorld, MemSkill achieves the highest success rates on both seen and unseen splits, indicating that skill-guided memory construction can benefit interactive decision making, whereas other baselines are less reliable at leveraging memory to support long-horizon action execution. Overall, the results show that MemSkill is effective across diverse settings.

Generalization across base models. A key advantage of MemSkill is strong generalization across base models. We train MemSkill only with LLaMA and directly transfer the learned skills to Qwen without retraining. Despite this strict transfer setting, MemSkill remains competitive and outperforms strong baselines on both conversational and embodied evaluations, showing that the evolved skills capture reusable memory behaviors that can be instantiated by different underlying LLMs.

Cross-dataset transfer. MemSkill also generalizes across datasets within the same broad setting. In particular, LongMemEval is evaluated purely by transferring the skill bank learned on LoCoMo, yet MemSkill achieves the best results among all methods, suggesting that the learned skills are not overfit to a single benchmark. Section 4.3 studies transfer under more pronounced distribution shifts.

4.3 Skill Generalization Under Distribution Shift

Beyond transfer within dialogue-style memory benchmarks, we evaluate whether learned skills generalize under a distribution shift in interaction format and evidence structure. Concretely, we directly apply the skill bank trained on LoCoMo to HotpotQA, where inputs are long-form, document-style narratives rather than multi-turn dialogues. Following the protocol in [29], we test three context-length settings with increasing difficulty, corresponding to different numbers of concatenated documents (i.e., 50/100/200). All results in this section use LLaMA as the base model and report the

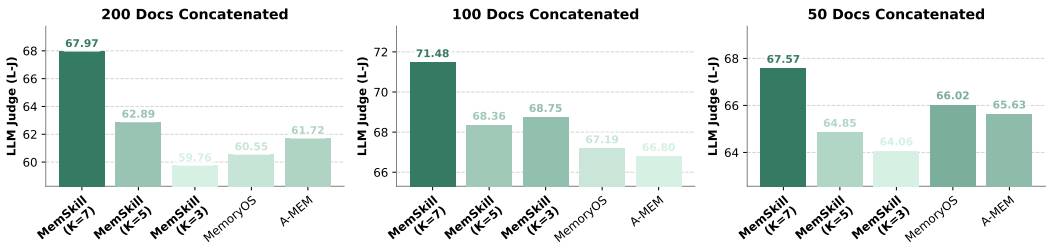

Figure 3: **Skill generalization under distribution shift on HotpotQA.** We transfer the LoCoMo-trained skill bank to HotpotQA and evaluate three context-length settings (50/100/200 concatenated documents) following [29]. Bars show LLM-judge (L-J) under LLaMA with different Top-K skill counts, compared to MemoryOS and A-MEM.

LLM-judge score (L-J). For baselines, we include MemoryOS and A-MEM, the most competitive methods on conversational benchmarks in Table 1, and omit weaker alternatives for clarity.

Figure 3 shows that MemSkill transfers strongly to HotpotQA across all three context sizes. MemSkill consistently outperforms strong baselines such as MemoryOS and A-MEM, with gains becoming more pronounced in the challenging long-context setting. These results suggest that the learned memory skills are not tied to dialogue-specific surface forms, but capture reusable extraction and revision behaviors that remain effective as input structure and retrieval demands change.

The same plots also reveal mild sensitivity to the number of selected skills K. Increasing K generally improves performance, with $K{=}7$ achieving the best results across all three settings, while smaller K can under-utilize the skill bank under longer contexts. Overall, the trend indicates that MemSkill benefits from composing multiple skills when the context becomes longer and noisier, while still maintaining strong transfer without any HotpotQA-specific training.

4.4 Ablation Study

We perform ablations to disentangle the contributions of (i) learning to select skills and (ii) evolving the skill bank. Table 2 reports LLM Judge (L-J) results on LoCoMo under both base models (LLaMA and Qwen). As shown, **w/o controller (w/o Ctrl)** replaces the learned controller with random skill selection while keeping the rest of the pipeline unchanged. **w/o designer (w/o Des)** disables the designer and fixes the skill bank to the four initial primitives. **Refine-only (Ref.-only)** allows the designer to refine existing skills but prohibits adding new ones.

Table 2: **LoCoMo ablation** (L-J).

Variant	LLaMA	Qwen
MemSkill	**53.82**	**54.14**
w/o Ctrl	48.43	42.84
w/o Des	46.50	36.15
Ref.-only	47.45	48.88

Across both base models, removing either component consistently degrades performance, confirming that MemSkill benefits from both targeted skill selection and skill evolution. In particular, random skill selection leads to a clear drop from the default setting, highlighting the importance of learning to choose relevant skills rather than providing arbitrary ones. Disabling the designer yields an even larger degradation, especially under Qwen, suggesting that evolving the skill bank is important for learning reusable memory behaviors that generalize beyond a fixed, manually specified operation set. Finally, refinement-only consistently outperforms static skills on both LLaMA and Qwen, with a particularly large gain under Qwen, yet remains below the default setting, indicating that introducing new skills yields additional benefits beyond refining the initial primitives.

4.5 Discussion

Case study. To make MemSkill more interpretable, we inspect the final evolved skill bank and report representative skills from LoCoMo and ALFWorld. As shown in Figure 4, the learned skills show clear domain specialization. LoCoMo skills emphasize temporal context and activity details, suggesting that dialogue memory benefits from lightweight event structure. In contrast, ALFWorld skills focus on action constraints and object locations, showing that embodied tasks require actionable

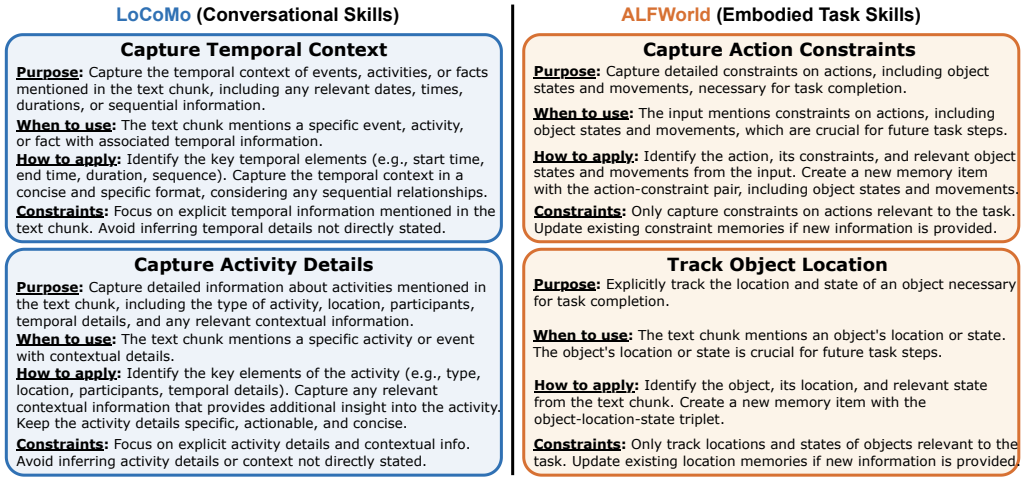

Figure 4: **Case study.** We show representative evolved skills learned on LoCoMo and ALFWorld.

world-state memories for multi-step execution. Overall, the evolved skill bank reflects recurring information needs from the data, rather than a fixed notion of what to remember.

Together, these skills show that MemSkill distills and refines reusable memory behaviors from interaction data, reducing reliance on hand-crafted memory designs. Appendix C gives more examples.

Cost analysis. We conduct a runtime cost analysis on LoCoMo using LLaMA, additionally including **LightMem** [3] as a baseline. We report L-J, input/output tokens, and LLM calls, accounting for all inference-time LLM calls from memory extraction and query answering, excluding LLM-judge calls used only for evaluation. Since MemSkill learns and evolves the skill bank before deployment, this preparation cost is amortized over repeated use, with further evolution triggered only occasionally. We therefore focus on runtime cost as the practical efficiency measure. At inference time, MemSkill constructs memory at the span level rather than turn by turn, substantially reducing LLM calls. To show this trade-off, we vary the span size (SS) and report the quality-cost frontier in Table 3.

MemSkill achieves a stronger quality-cost trade-off than prior baselines. With moderate span sizes, it obtains higher L-J scores while using fewer input/output tokens and LLM calls. In particular, Span Size=512 offers the best overall balance, achieving the highest quality with much lower runtime cost than MemoryOS, A-MEM, and LightMem. This suggests span-level construction reduces redundant LLM calls without sacrificing memory quality. Non-monotonic performance reflects a granularity trade-off: smaller spans may fragment events, while larger spans may overlook details. Span size thus balances quality and efficiency.

Table 3: **Cost analysis on LoCoMo.**

Setting	L-J	In (K)	Out (K)	Calls
MemoryOS	48.64	1013	165	1288
A-MEM	49.71	2850	362	1548
LightMem	51.95	789	209	685
Ours (SS=128)	53.14	622	57	376
Ours (SS=256)	51.61	390	19	270
Ours (SS=512)	53.82	249	18	215
Ours (SS=1024)	48.11	188	11	187
Ours (SS=2048)	50.46	178	10	185

5 Conclusion

We present **MemSkill**, an agent memory method that reframes memory operations as an evolving skill bank. MemSkill learns to select relevant skills for each context span and conditions an LLM executor on them to construct skill-guided memories. Beyond learning to use a fixed skill set, MemSkill introduces a designer that improves the skill bank by refining existing skills and proposing new ones from challenging cases, forming a closed-loop training procedure. Experiments on LoCoMo, LongMemEval, HotpotQA, and ALFWorld show consistent improvements over strong baselines, while qualitative analyses illustrate how evolving skills enable more adaptive memory management. We hope MemSkill encourages future work on self-improving agent memory systems that learn not only to use memory, but also to continually improve how memory is constructed and maintained.

Acknowledgments and Disclosure of Funding

This research is supported by the National Research Foundation, Singapore under its National Cybersecurity R&D Programme (CRPO-GC5-NTU-011). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore.

References

[1] Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution. *arXiv preprint arXiv:2510.23595*, 2025.

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

[3] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Huajun Chen, and Ningyu Zhang. Lightmem: Lightweight and efficient memory-augmented generation. In Carl Vondrick, Bharath Hariharan, Colin Raffel, Lerrel Pinto, Diyi Yang, and Aleksandra Faust, editors, *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026.

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

[5] Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

[6] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of AI agents. *CoRR*, abs/2512.13564, 2025.

[7] Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*, 2025.

[8] Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. Memory os of ai agent. *arXiv preprint arXiv:2506.06326*, 2025.

[9] Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In *International conference on machine learning*, pages 3499–3508. PMLR, 2019.

[10] LangChain. Langmem. `https://github.com/langchain-ai/langmem`, 2025. GitHub repository.

[11] Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

[12] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. *arXiv preprint arXiv:2402.17753*, 2024.

[13] Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. *CoRR*, abs/2506.13131, 2025.

[14] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In Carl Vondrick, Bharath Hariharan, Colin Raffel, Lerrel Pinto, Diyi Yang, and Aleksandra Faust, editors, *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026.

[15] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: Towards llms as operating systems. 2023.

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

[17] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

[18] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, 2024.

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

[20] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: understanding self-evolution in LLM agents via multi-turn reinforcement learning. *CoRR*, abs/2504.20073, 2025.

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

[22] Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking LLM agent test-time learning with self-evolving memory. *CoRR*, abs/2511.20857, 2025.

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

[24] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. Evolver: Self-evolving LLM agents through an experience-driven lifecycle. *CoRR*, abs/2510.16079, 2025.

[25] 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.

[26] Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens, editors,

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2026, San Diego, California, United States, July 2-7, 2026, pages 12805–12825. Association for Computational Linguistics, 2026.

[27] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

[29] Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context LLM with multi-conv rl-based memory agent. In Carl Vondrick, Bharath Hariharan, Colin Raffel, Lerrel Pinto, Diyi Yang, and Aleksandra Faust, editors, *The Fourteenth International Conference on Learning Representations, ICLR 2026, Rio de Janeiro, Brazil, April 23-27, 2026*. proceedings.iclr.cc, 2026.

[30] Wenhao Yu, Hongming Zhang, Xiaoman Pan, Peixin Cao, Kaixin Ma, Jian Li, Hongwei Wang, and Dong Yu. Chain-of-note: Enhancing robustness in retrieval-augmented language models. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 14672–14685, 2024.

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

[32] Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, Zhaoyang Liu, Bolin Ding, and Jingren Zhou. Agentevolver: Towards efficient self-evolving agent system. *CoRR*, abs/2511.10395, 2025.

[33] Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. *arXiv preprint arXiv:2509.24704*, 2025.

[34] Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems. *arXiv preprint arXiv:2512.18746*, 2025.

[35] 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.

[36] Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *arXiv preprint arXiv:2505.03335*, 2025.

[37] Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. Skillweaver: Web agents can self-improve by discovering and honing skills. *CoRR*, abs/2504.07079, 2025.

[38] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

Appendix Contents

A More Experimental Results **13**

A.1 More Comparison Experiments 13

A.2 More Results on Appworld 15
A.3 Experimental Results on Small Models 16
A.4 Training Stability 16
A.5 Additional Analysis on Skill Selection 17
A.6 Comparison with Simplified Alternatives 17

B More Implementation Details **17**
B.1 Evaluation Details 17
B.2 More Details of the Designer 18
B.3 Details on ALFWorld Training 19
B.4 Details on Training Objectives 20

C Case Study **21**
C.1 Initial Primitive Skills 21
C.2 Evolved Skills on LoCoMo 23
C.3 Evolved Skills on ALFWorld 28

D Prompts **30**

E Use of Large Language Models **36**

F Limitations and Societal Impact **36**

A More Experimental Results

A.1 More Comparison Experiments

We further compare MemSkill with additional baselines on ALFWorld, including LightMem [3], AWM [21], and Expel [35]. As shown in Table 4, MemSkill consistently achieves the best performance across both base models and evaluation splits. With LLaMA, MemSkill obtains an average success rate of 80.36, outperforming the strongest baseline LightMem by 5.53 points. The gain is especially clear on ALF-Unseen, where MemSkill improves the success rate from 75.37 to 83.58 while reducing the average number of steps from 20.69 to 16.63.

The advantage remains under transfer evaluation with Qwen, where MemSkill is not trained using this base model or dataset. MemSkill achieves an average success rate of 81.29, surpassing the strongest baseline Expel by 7.61 points. It also requires fewer interaction steps on both ALF-Seen and ALF-Unseen. These results suggest that the learned memory skills can capture reusable experience patterns for embodied decision-making, leading to both higher task success and more efficient execution across seen and unseen environments.

Table 5 further reports ALFWorld results *without* in-context demonstrations. This setting is complementary to the main-table evaluation with demonstrations, but is more controlled for studying memory itself. Since in-context demonstrations can also act as an external form of memory, including them may confound the gains brought by each method's constructed memory. Therefore, this setting decouples demonstrations from memory construction and isolates the contribution of the learned memory mechanism.

Under this stricter setting, MemSkill still consistently outperforms all baselines across both base models and both evaluation splits. With LLaMA, MemSkill achieves an average success rate of 57.71, outperforming the strongest baseline LightMem by 11.72 points. The improvement is larger on ALF-Unseen, where MemSkill improves the success rate from 46.27 to 59.70, suggesting stronger

Table 4: **More comparison results on ALFWorld.**

Model	Methods	Embodied Interactive Tasks				
		ALF-Seen†		ALF-Unseen†		Avg.
		SR	#Stps↓	SR	#Stps↓	SR
LLaMA	LightMem	74.29	21.69	75.37	20.69	74.83
	AWM	66.43	23.25	68.66	22.25	67.55
	Expel	67.14	23.26	66.42	22.40	66.78
	MemSkill	**77.14**	**18.91**	**83.58**	**16.63**	**80.36**
▲Qwen	LightMem	70.71	20.48	58.21	25.62	64.46
	AWM	74.29	19.03	61.19	24.67	67.74
	Expel	75.71	18.81	71.64	20.98	73.68
	MemSkill	**85.71**	**13.84**	**76.87**	**18.16**	**81.29**

Bold indicates the best score within each base model block.
▲ indicates no training using this base model or dataset.
† indicates evaluation with in-context demonstrations.

Table 5: **More comparison results on ALFWorld *without* in-context demonstrations.**

Model	Methods	Embodied Interactive Tasks				
		ALF-Seen		ALF-Unseen		Avg.
		SR	#Stps↓	SR	#Stps↓	SR
LLaMA3.3 70B-Instruct	No-Memory	17.14	43.74	20.15	42.99	18.65
	CoN	40.71	33.44	30.60	37.66	35.66
	ReadAgent	32.86	37.09	38.06	34.78	35.46
	MemoryBank	25.00	39.96	32.84	36.54	28.92
	A-MEM	24.29	40.51	28.36	38.83	26.33
	Mem0	32.86	36.47	32.09	37.32	32.48
	LangMem	37.86	34.39	35.07	35.70	36.47
	MemoryOS	15.71	43.74	14.18	44.54	14.95
	LightMem	45.71	31.76	46.27	30.66	45.99
	AWM	35.71	35.87	39.55	34.93	37.63
	Expel	42.14	34.61	43.28	33.25	42.71
	MemSkill	**55.71**	**26.87**	**59.70**	**25.88**	**57.71**
▲Qwen3-Next 80B-A3B-Instruct	No-Memory	18.57	42.48	26.12	39.35	22.35
	CoN	57.86	25.81	53.73	28.40	55.80
	ReadAgent	53.57	27.88	54.48	27.41	54.03
	MemoryBank	37.86	35.15	38.06	34.99	37.96
	A-MEM	25.00	40.28	29.10	39.04	27.05
	Mem0	38.57	33.64	41.04	33.16	39.81
	LangMem	37.14	34.42	31.34	37.17	34.24
	MemoryOS	19.29	42.43	18.66	42.95	18.98
	LightMem	30.71	36.81	26.87	38.96	28.79
	AWM	47.86	31.24	47.01	31.69	47.44
	Expel	54.29	28.90	55.22	29.57	54.76
	MemSkill	**65.71**	**22.49**	**61.94**	**24.59**	**63.83**

Bold indicates the best score within each base model block.
▲ indicates no training using this base model or dataset.

generalization to unseen environments. MemSkill also requires fewer interaction steps than all baselines, reducing the average steps on ALF-Seen and ALF-Unseen to 26.87 and 25.88, respectively.

The same trend holds in the transfer setting with Qwen, where MemSkill is not trained using this base model or dataset. MemSkill achieves an average success rate of 63.83, improving over the strongest baseline by 8.03 points. It also obtains the lowest number of steps on both ALF-Seen and ALF-Unseen. These results show that the gains of MemSkill do not rely on in-context demonstrations, and that the learned memory skills provide reusable task experience that improves both success rate and execution efficiency.

Table 6: **More comparison results on AppWorld *with* and *without* in-context demonstrations.**

Model	Methods	Interactive Tool-Use Benchmark									
		Test-N		Test-C		Avg.	Test-N†		Test-C†		Avg.
		PR	#Steps↓	PR	#Steps↓	PR	PR	#Steps↓	PR	#Steps↓	PR
LLaMA3.3 70B-Instruct	No-Memory	22.80	36.71	19.10	37.92	20.95	54.03	14.21	38.78	19.12	46.41
	ReadAgent	23.43	34.82	20.49	31.36	21.96	53.76	14.54	38.67	19.24	46.22
	MemoryBank	23.28	35.66	19.96	33.34	21.62	53.66	14.89	38.46	18.84	46.06
	A-MEM	22.13	38.46	17.60	38.46	19.87	52.85	14.31	38.16	18.78	45.51
	Mem0	27.02	27.04	21.62	30.64	24.32	54.32	14.14	39.73	18.55	47.03
	MemoryOS	23.24	38.26	19.17	36.17	21.21	53.14	13.62	38.52	17.83	45.83
	LightMem	25.42	30.73	21.32	30.33	23.37	51.25	14.10	39.12	18.50	45.19
	AWM	29.00	**17.73**	21.83	**20.48**	25.42	54.68	14.26	39.54	18.64	47.11
	Expel	25.29	29.96	21.20	30.63	23.25	52.88	13.56	39.49	17.91	46.19
	MemSkill	**31.12**	29.38	**22.29**	32.17	**26.71**	**54.84**	**13.25**	**39.79**	**17.59**	**47.32**
▲Qwen3-Next 80B-A3B-Instruct	No-Memory	22.41	38.39	19.99	36.78	21.20	47.68	16.81	38.42	19.70	43.05
	ReadAgent	26.29	34.77	20.36	35.07	23.33	46.83	17.59	37.65	21.12	42.24
	MemoryBank	24.41	36.94	20.28	36.19	22.35	46.50	16.98	37.18	22.03	41.84
	A-MEM	23.03	37.17	18.39	37.43	20.71	45.44	17.49	38.14	21.40	41.79
	Mem0	40.42	**18.48**	22.16	22.97	31.29	50.19	15.11	39.64	20.33	44.92
	MemoryOS	25.00	35.98	21.46	35.20	23.23	50.98	16.11	38.03	21.32	44.51
	LightMem	36.90	23.71	23.44	27.05	30.17	48.91	17.23	40.85	21.14	44.88
	AWM	27.02	23.90	22.05	22.63	24.54	53.23	16.77	41.26	22.95	47.25
	Expel	25.00	22.20	20.40	**21.00**	22.70	49.88	16.86	39.17	20.06	44.53
	MemSkill	**43.05**	26.11	**25.41**	31.39	**34.23**	**54.53**	**15.05**	**42.30**	**19.23**	**48.42**

Bold indicates the best score within each base model block.
▲ indicates no training using this base model or dataset.
† indicates evaluation with in-context demonstrations.

A.2 More Results on Appworld

To further evaluate whether memory skills generalize beyond conversational and embodied household settings, we extend our study to AppWorld [18], a more challenging interactive tool-use benchmark. We report results on both Test-Normal (Test-N) and Test-Challenge (Test-C), using Pass Rate (PR) and the average number of execution steps as evaluation metrics. Our default setting follows the stronger evaluation protocol with in-context demonstrations. In addition, we also report a controlled variant without demonstrations, which helps isolate the contribution of each method's constructed memory from the task guidance provided by demonstrations.

As shown in Table 6, MemSkill achieves the best average PR across both base models under the default demonstration-based setting. With LLaMA, the No-Memory baseline already performs strongly, indicating that in-context demonstrations provide substantial task-specific guidance and make this setting partially saturated. As a result, the performance gaps among different memory methods become relatively small. Nevertheless, MemSkill still achieves the best overall pass rate and uses the fewest execution steps across both splits, suggesting that the learned memory skills can still improve efficiency even when demonstrations already provide strong external guidance.

The benefit of MemSkill is more evident with Qwen, where the demonstration-based setting leaves more room for memory-based improvement. In this case, MemSkill consistently outperforms prior memory methods and also improves over the No-Memory baseline. It further achieves the lowest execution steps on both Test-N and Test-C, indicating that MemSkill can effectively complement in-context demonstrations and support more efficient tool-use behavior.

The controlled setting without in-context demonstrations further confirms the independent contribution of learned memory skills. In this setting, MemSkill achieves the best average PR under both base models, with consistent gains on both Test-N and Test-C. This shows that MemSkill is not merely benefiting from demonstrations, but can provide useful experience abstraction when the agent must rely more directly on constructed memory.

Overall, these results show that MemSkill generalizes to interactive tool-use environments beyond conversational and embodied household tasks. Under the default demonstration-based setting,

MemSkill remains competitive even when demonstrations provide strong task guidance, and brings clearer gains when the base model leaves more room for memory-based improvement. Under the controlled no-demonstration setting, MemSkill consistently improves pass rate over prior memory methods, further supporting the effectiveness of skill-conditioned memory construction for complex multi-step tool use.

A.3 Experimental Results on Small Models

We further evaluate MemSkill with Llama-3.1-8B-Instruct to examine whether the learned memory skills remain effective when the base model has more limited capacity. As shown in Table 7, MemSkill consistently achieves the best performance across both LoCoMo and LongMemEval. Compared with prior memory methods, MemSkill improves both F1 and L-J on the two conversational benchmarks, indicating that skill-conditioned memory construction can provide useful support even for smaller backbone models.

The improvement is also maintained on LongMemEval, where no training is performed using this base model or dataset. This suggests that the learned memory skills are not tightly coupled to a specific backbone, and can transfer to smaller models under long-context conversational settings. Overall, these results show that MemSkill does not rely solely on the capability of a large base model. Instead, its learned memory skills provide complementary gains that remain effective under a more resource-efficient model setting.

Table 7: **Experimental results on LoCoMo and LongMemEval using Llama-3.1-8B-Instruct.**

Model	Methods	Conversational Benchmarks				
		LoCoMo		▲LongMemEval		Avg.
		F1	L-J	F1	L-J	L-J
LLaMA3.1 8B-Instruct	ReadAgent	17.53	21.49	9.45	18.80	20.15
	MemoryBank	18.84	22.36	13.27	24.59	23.48
	A-MEM	19.95	25.62	16.12	25.71	25.67
	Mem0	14.26	17.12	14.59	26.38	21.75
	LangMem	13.36	15.71	6.52	17.16	16.44
	MemoryOS	17.68	23.08	7.88	18.24	20.66
	LightMem	19.74	26.85	17.68	29.40	28.13
	MemSkill	**20.59**	**27.23**	**18.13**	**30.20**	**28.72**

Bold indicates the best score within each base model block.
▲ indicates no training using this base model or dataset.

A.4 Training Stability

MemSkill incorporates several mechanisms to improve the stability of skill learning and evolution. First, we maintain a snapshot of the skill bank after each evolution round. If the updated skill bank underperforms the best-performing snapshot observed so far, we roll back to the best snapshot and continue subsequent evolution from it. This prevents occasional harmful skill updates from permanently degrading the memory system.

Second, hard cases are selected according to difficulty scores, so the designer is guided by recurring failure patterns rather than noisy random examples. This encourages each evolution round to focus on systematic weaknesses of the current skill bank. Third, skill evolution is performed at controlled intervals, with a capped number of skill modifications in each round. This avoids abrupt changes to the skill bank and makes the learning process more gradual.

Together, these mechanisms make the evolution process less sensitive to noisy feedback and reduce the risk of unstable skill drift. In practice, we observe that the learned skill bank improves progressively over evolution rounds, suggesting that the snapshot rollback, hard-case selection, and controlled update strategy provide a stable basis for self-evolving memory skills. More details can be found in Appendix B.2

A.5 Additional Analysis on Skill Selection

We further evaluate the learned controller against two alternative skill selection strategies on LoCoMo. **Frozen Similarity** selects Top-K skills using cosine similarity between frozen state and skill embeddings. **LLM Selector** prompts the base LLM to select Top-K skills given the current span, retrieved memories, and full skill index. Both baselines use the same skill bank and executor as MemSkill.

Table 8: **Comparison of skill selection strategies on LoCoMo using LLaMA.**

Method	F1	L-J
Frozen Similarity	40.97	50.80
LLM Selector	39.16	48.73
MemSkill	**44.21**	**53.82**

As shown in Table 8, MemSkill consistently outperforms both alternatives, demonstrating that reward-optimized skill selection is more effective than static semantic similarity or direct LLM prompting.

A.6 Comparison with Simplified Alternatives

We further compare MemSkill with two simplified alternatives on LoCoMo using LLaMA.

Gradient-Free Self-Correction. We replace the learned controller with a prompted LLM selector throughout training and inference. The Designer continues to evolve the skill bank from downstream failure cases, but no policy optimization is performed.

Static Well-Engineered Skill Set. Starting from MemSkill's best-performing skill bank, we further optimize its instructions using GPT-5.6-Sol-Pro, guided by existing memory systems and benchmark feedback, followed by manual verification. The resulting skill bank is then frozen and evaluated with a prompted LLM selector, without PPO training or further evolution.

Table 9: **Comparison with simplified alternatives on LoCoMo using LLaMA.**

Method	F1	L-J
Gradient-Free Self-Correction	36.35	46.34
Static Well-Engineered Skills	35.28	44.90
MemSkill	**44.21**	**53.82**

As shown in Table 9, MemSkill outperforms both alternatives, demonstrating the benefits of combining learned skill selection with feedback-driven skill evolution over purely prompted or static skill management.

B More Implementation Details

B.1 Evaluation Details

LLM judge and infrastructure. We use `openai/gpt-oss-120b` as the LLM judge (judge prompts can be found in Appendix D). All API-based models are accessed through the NV NIM API[1] and Together API[2]. Training is conducted on NVIDIA A6000 GPUs.

Training cost. On LoCoMo, a typical training run takes approximately 8–12 hours to converge in our setup, using a single NVIDIA A6000 GPU and consuming on the order of hundreds of millions of input tokens and tens of millions of output tokens. Training is primarily constrained by remote LLM inference and API latency, as PPO only optimizes lightweight MLPs. Skill evolution typically involves 5–6 rounds before convergence, with each Designer update taking a few minutes. These estimates may vary with API conditions and training configurations.

[1] `https://docs.nvidia.com/nim/`
[2] `https://docs.together.ai/`

LoCoMo [12]. LoCoMo contains 10 long interaction samples, each paired with roughly 200 training queries on average. We split the dataset by sample into train, validation, and test sets with a 6/2/2 ratio. We further remove *adversarial* queries, since their supporting evidence is absent from the provided context and may introduce noisy supervision during training.

LongMemEval [23]. We use the LongMemEval-S split, where each example contains an ultra-long conversation of roughly 100K tokens. We remove abstention questions, since they do not require retrieving or constructing useful memory from the conversation history. We split the remaining data into train, validation, and test sets. Although the training split is not used for learning in this transfer setting, the validation split is used to tune dataset-specific configurations. We then conduct transfer evaluation on a stratified test sample of about one-fifth of the dataset, approximately 100 samples, ensuring coverage of different question types for a comprehensive assessment.

ALFWorld [17]. We first collect expert trajectories from the training split and use them as the corpus for memory or experience construction. We then evaluate on the official ALF-Seen and ALF-Unseen splits. More training configuration details can be found in Appendix B.3.

HotpotQA [28]. We use HotpotQA to study transfer under distribution shift, following the evaluation protocol of [29]. Specifically, we evaluate on three context-length settings with increasing difficulty, corresponding to 50, 100, and 200 concatenated documents, denoted as `eval_50`, `eval_100`, and `eval_200`. Unless otherwise specified, all results in this part use LLaMA as the base model and report the LLM-judge score (L-J).

Span-level evaluation. During evaluation, we construct memory at the span level with a default span size of 512 tokens, rather than updating memory turn by turn. This substantially reduces the number of LLM calls and improves evaluation efficiency.

B.2 More Details of the Designer

Hard-case buffer and representative case mining. The designer maintains a sliding *hard-case buffer* that tracks recently challenging evaluation cases without growing unbounded. Each case stores the query, the retrieved memories used to answer it, the model prediction, the reference answer, the resulting task reward (e.g., F1), and a failure counter that records how many times the case has been answered incorrectly. To prioritize cases that are both low-reward and repeatedly failed, we assign each case a difficulty score

$$d(q) \;=\; \big(1 - r(q)\big) \cdot c(q), \tag{4}$$

where $r(q) \in [0, 1]$ is the task reward for query q and $c(q)$ is its cumulative failure count within the buffer window. Higher $d(q)$ indicates more critical cases that should be examined first.

To encourage coverage over *diverse* failure types, we further cluster hard cases by semantic similarity of their queries and mine representative cases from each cluster. For example, in LoCoMo, some queries focus on temporal cues (e.g., *when* an event happened) while others emphasize locations (e.g., *where* something occurred). Clustering helps separate these semantic types so the designer feedback is not dominated by a single frequent error mode, improving diversity and completeness of the mined supervision.

Exploration incentive for newly introduced skills. After each evolution round, the designer may introduce new skills that the controller has not yet learned to utilize. To facilitate adoption, we apply a short post-update exploration phase by biasing the controller toward new skills directly at the logit level. Let $\mathcal{S}_{\text{new}} \subseteq \mathcal{S}_t$ denote the newly added skills, and let $p_\theta(i \mid h_t) = \mathrm{softmax}(z_t)_i$ be the controller distribution at step t. We encourage the total probability mass assigned to new skills to reach a target threshold τ_q:

$$\sum_{i \in \mathcal{S}_{\text{new}}} p_\theta(i \mid h_t) \;\geq\; \tau_q, \qquad \tau_q \in [0, 1]. \tag{5}$$

When the constraint in Eq. (5) is violated, we add a uniform logit gain δ_q to all new skills,

$$z'_{t,i} \;=\; \begin{cases} z_{t,i} + \delta_q, & i \in \mathcal{S}_{\text{new}}, \\ z_{t,i}, & \text{otherwise}, \end{cases} \qquad p'_\theta(\cdot \mid h_t) = \mathrm{softmax}(z'_t), \tag{6}$$

where δ_q is chosen as the minimal value that makes $\sum_{i \in \mathcal{S}_{\text{new}}} p'_\theta(i \mid h_t) \geq \tau_q$. By operating on logits, this mechanism preserves the controller architecture and provides a smooth probability-level encouragement toward new skills.

We apply this incentive for the first T_{explore}=50 training steps after each evolution round. To avoid persistent bias, the target threshold decays linearly within this window:

$$\tau_q \;=\; \tau_0 \cdot \Big(1 - \frac{q}{T_{\text{explore}}}\Big), \qquad q = 0, 1, \ldots, T_{\text{explore}} - 1, \tag{7}$$

with default τ_0=0.3. This schedule provides strong initial exploration and then gradually fades, yielding a smooth transition back to the controller's learned selection behavior.

Early stopping and rollback based on stabilized rewards. MemSkill performs skill evolution periodically, where each evolution cycle consists of a fixed number of controller-training steps (e.g., 100 steps) on the current skill bank. Because the reward signal can be volatile immediately after a skill-bank update, we assess whether a cycle improves performance using a *stabilized* reward estimate: we compute the average task reward over the *last quarter* of training steps within the cycle, and treat this value as the cycle's score.

Let L denote the number of controller-training steps per cycle and $\{r_t\}_{t=1}^{L}$ the step-level rewards within the cycle. We define the cycle score as

$$\bar{r}_{\text{tail}} \;=\; \frac{1}{L/4} \sum_{t=3L/4+1}^{L} r_t. \tag{8}$$

We compare $\bar{r}_{\text{tail}}$ against the best score observed so far. If the current cycle does not improve this criterion, then before performing the next skill evolution step, we roll back the skill bank to the previously best-performing snapshot and restart evolution from that snapshot. This rollback prevents compounding degradations from suboptimal designer updates.

Finally, if the stabilized reward fails to improve for several consecutive evolution cycles (we use a fixed patience), we early stop training and return the best skill bank snapshot encountered during training.

B.3 Details on ALFWorld Training

ALFWorld differs from the other benchmarks in that it is an interactive environment rather than a static text corpus. To instantiate MemSkill in this setting, we first convert ALFWorld into an offline training protocol by collecting expert trajectories on the training split. Each trajectory records the agent's interaction sequence (observations, actions, and outcomes) and serves as an interaction trace for memory construction.

Task-type grouping. ALFWorld tasks naturally fall into a small number of recurring goal templates. Following common practice, we group trajectories by task type (i.e., goal template), such as PICK & PLACE (put an object into/on a target receptacle), CLEAN & PLACE (clean an object and then place it), HEAT & PLACE (heat an object and then place it), and COOL & PLACE (cool an object and then place it).[3]

Experience corpus vs. evaluation cases. To fit ALFWorld into our training framework, we construct per-type train-time data splits from the offline expert trajectories. For each task type, we randomly sample a subset of trajectories as the *experience corpus* used for memory construction, and sample another *non-overlapping* subset of trajectories from the same type as *evaluation cases*. During training, MemSkill builds a trajectory-specific memory bank from the experience corpus (span by span, via controller and executor), and then evaluates the constructed memory on the evaluation cases to obtain task reward and to log failure cases.

Motivation. Using non-overlapping trajectories from the *same* task type for experience construction and evaluation provides a controlled generalization signal: trajectories within a type share goal structure and recurrent interaction patterns, making memories and skills more transferable across different instances of the same template. This setup encourages MemSkill to learn reusable memory skills that capture type-level regularities (e.g., relevant object states and action prerequisites) rather than overfitting to a single trajectory, while still ensuring that evaluation traces are held out from the traces used to build memory.

[3]We use the task template provided by the environment to define task types.

B.4 Details on Training Objectives

This part details the reinforcement learning objective used to optimize the controller in MemSkill when each decision selects an ordered Top-K *set* of skills without replacement.

Episode, states, and Top-K actions. Training iterates over interaction traces (episodes). For a trace, MemSkill processes spans sequentially. At step t, the controller observes the raw state $s_t \triangleq (x_t, M_t)$, represented by the learned state embedding h_t defined in Section 3.3.1. Let $\mathcal{S}_t = \{1, \ldots, N_t\}$ denote the current skill bank, whose size N_t may change as the designer evolves skills. The controller computes logits $z_t \in \mathbb{R}^{N_t}$ by applying the shared scorer to all state-skill pairs, and induces

$$p_\theta(i \mid h_t) = \mathrm{softmax}(z_t)_i. \tag{9}$$

Instead of sampling a single skill, the controller selects an *ordered* Top-K set $A_t = (a_{t,1}, \ldots, a_{t,K})$ *without replacement*, implemented via Gumbel-Top-K sampling [9] (i.e., adding i.i.d. Gumbel noise to logits and taking the top-K indices).

Joint probability of Top-K without-replacement selection. For PPO-style policy optimization, we need the joint probability of sampling the ordered set A_t under the without-replacement process. This probability can be written as

$$\pi_\theta(A_t \mid h_t) = \prod_{j=1}^{K} \frac{p_\theta(a_{t,j} \mid h_t)}{1 - \sum_{\ell<j} p_\theta(a_{t,\ell} \mid h_t)}, \tag{10}$$

with the corresponding joint log-probability

$$\log \pi_\theta(A_t \mid h_t) = \sum_{j=1}^{K} \Big(\log p_\theta(a_{t,j} \mid h_t) - \log \big(1 - \sum_{\ell<j} p_\theta(a_{t,\ell} \mid h_t)\big) \Big). \tag{11}$$

When $K = 1$, Eq. 10 reduces to the standard single-action case.

Rewards from memory-dependent evaluation. For each trace, after processing all spans and constructing the trace-specific memory bank, we evaluate the memory bank on the trace's memory-dependent training queries and obtain a scalar task score (e.g., F1 or success rate). We treat this score as the episode-level reward:

$$R \triangleq \mathrm{Eval}(\text{memory bank}; \text{training queries}) \in \mathbb{R}. \tag{12}$$

This reward is then assigned to the sequence of controller decisions within the trace. Concretely, we use standard return computation with discount factor γ:

$$G_t = \sum_{\tau=t}^{T} \gamma^{\tau-t} r_\tau, \tag{13}$$

where r_τ is the per-step reward. In our default setting, reward is provided only after memory construction completes, i.e., $r_T = R$ and $r_\tau = 0$ for $\tau < T$, so $G_t = \gamma^{T-t} R$. We learn a value function $V_\phi(h_t)$ and compute advantages $\hat{A}_t$ using generalized advantage estimation (GAE).

PPO objective with Top-K actions. We optimize the controller using proximal policy optimization (PPO) [16], replacing the standard single-action log-probability with the Top-K joint log-probability in Eq. 11. Let θ_{old} denote the parameters of the behavior policy used to collect rollouts. For simplicity, we use h_t to denote the state representation computed under the policy being evaluated. Define the importance ratio:

$$r_t(\theta) = \frac{\pi_\theta(A_t \mid h_t)}{\pi_{\theta_{\text{old}}}(A_t \mid h_t)} = \exp\Big(\log \pi_\theta(A_t \mid h_t) - \log \pi_{\theta_{\text{old}}}(A_t \mid h_t) \Big). \tag{14}$$

The clipped surrogate policy objective is

$$\mathcal{L}_{\text{policy}}(\theta) = \mathbb{E}_t \Big[\min \big(r_t(\theta)\, \hat{A}_t,\ \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\, \hat{A}_t \big) \Big]. \tag{15}$$

We additionally optimize a value function and include an entropy bonus for exploration:

$$\mathcal{L}_{\text{value}}(\phi) = \mathbb{E}_t\Big[\big(V_\phi(h_t) - G_t\big)^2\Big], \tag{16}$$

$$\mathcal{H}(\theta) = \mathbb{E}_t\big[H(p_\theta(\cdot \mid h_t))\big], \tag{17}$$

where $H(\cdot)$ is the entropy of the categorical distribution over all skills. The overall objective (to maximize) is

$$\max_{\theta,\phi} \; \mathcal{L}_{\text{policy}}(\theta) - c_v\, \mathcal{L}_{\text{value}}(\phi) + c_H\, \mathcal{H}(\theta). \tag{18}$$

In implementation, we minimize the negative of Eq. 18.

Gumbel-Top-K exploration. To sample Top-K skills without replacement during rollout collection, we use Gumbel-Top-K sampling: at each step we draw i.i.d. Gumbel noise $\{g_i\}_{i=1}^{N_t}$, form perturbed logits $\tilde{z}_{t,i} = z_{t,i} + g_i$, and take the indices of the K largest $\tilde{z}_{t,i}$ to obtain A_t. This provides stochastic exploration over skill subsets while remaining compatible with PPO through the joint probability in Eq. 10. For training stability, entropy regularization is computed from the base categorical distribution $p_\theta(\cdot \mid h_t)$ over all skills (Eq. 17), which encourages exploration of the evolving skill bank even though the executed action is a Top-K set.

C Case Study

C.1 Initial Primitive Skills

Initial Primitive Skill - INSERT

Skill: Insert New Memory

Description: Memory management skill for capturing new, durable facts from the current text chunk that are not already in memory.

Purpose: Capture new, durable facts from the current text chunk that are missing in memory.

When to use:
- The text chunk introduces new facts, events, plans, or context worth storing.
- The information is stable and likely useful later.

How to apply:
- Compare against retrieved memories to avoid duplicates.
- Split distinct facts into separate items.
- Keep each item concise and specific.

Constraints:
- Skip trivial, fleeting, or speculative content.
- Do not update or delete existing memories.

Action type: INSERT only.

Initial Primitive Skill - UPDATE

Skill: Update Existing Memory

Description: Memory management skill for revising an existing memory item when the text chunk provides corrections or new details.

Purpose: Revise a retrieved memory with new or corrected information from the text chunk.

When to use:
- The text chunk clarifies, corrects, or extends a retrieved memory.

How to apply:
- Select the best matching memory item.
- Merge new details into a single updated item.
- Preserve accurate details that still hold.

Constraints:
- Do not create new memories.
- Do not delete items.

Action type: UPDATE only.

Initial Primitive Skill - DELETE

Skill: Delete Invalid Memory

Description: Memory management skill for removing memory items that are incorrect, outdated, or superseded.

Purpose: Remove a retrieved memory that is wrong, outdated, or superseded by the text chunk.

When to use:
- The text chunk clearly contradicts a memory.
- A plan or fact is explicitly canceled or replaced.

How to apply:
- Only delete when evidence is explicit.

Constraints:
- If uncertain, prefer no action over deletion.

Action type: DELETE only.

Initial Primitive Skill - SKIP

Skill: No Operation

Description: Memory management skill for confirming that no memory changes are required.

Purpose: Confirm no memory changes are needed for the text chunk.

When to use:
- The text chunk contains no new, corrective, or actionable information.

Constraints:
- Emit NOOP only if none of the selected skills produce actions.

Action type: NOOP only.

C.2 Evolved Skills on LoCoMo

Evolved Skill on LoCoMo - CAPTURE_ACTIVITY_DETAILS

Skill: Capture Activity Details

Purpose: Capture detailed information about activities mentioned in the text chunk, including the type of activity, location, participants, temporal details, and any relevant contextual information.

When to use:
- The text chunk mentions a specific activity or event with contextual details.

How to apply:
- Identify the key elements of the activity (e.g., type, location, participants, temporal details).
- Capture any relevant contextual information that provides additional insight into the activity.
- Keep the activity details specific, actionable, and concise.

Constraints:
- Focus on explicit activity details and contextual information mentioned in the text chunk.
- Avoid inferring activity details or context not directly stated.

Action type: INSERT only.

Evolved Skill on LoCoMo - CAPTURE_ENTITY_NUANCES

Skill: Capture Entity Nuances

Purpose: Capture nuanced details about entities mentioned in the text chunk, such as nicknames, aliases, or comparative statements.

When to use:
- The text chunk mentions an entity with nuanced details (e.g., nickname, alias, comparison).

How to apply:
- Identify the entity and its associated nuances.
- Capture these nuances in a way that distinguishes them from the entity's primary information.

Constraints:
- Focus on explicit nuances mentioned in the text chunk.
- Avoid inferring nuances not directly stated.

Action type: INSERT only.

Evolved Skill on LoCoMo - CAPTURE_TEMPORAL_CONTEXT

Skill: Capture Temporal Context

Purpose: Capture the temporal context of events, activities, or facts mentioned in the text chunk, including any relevant dates, times, durations, or sequential information.

When to use:
- The text chunk mentions a specific event, activity, or fact with associated temporal information.

How to apply:
- Identify the key temporal elements (e.g., start time, end time, duration, sequence).
- Capture the temporal context in a concise and specific format, considering any sequential relationships.

Constraints:
- Focus on explicit temporal information mentioned in the text chunk.
- Avoid inferring temporal details not directly stated.

Action type: INSERT only.

Evolved Skill on LoCoMo - DELETE

Skill: Delete Invalid Memory

Purpose: Remove a retrieved memory that is wrong, outdated, or superseded by the text chunk.

When to use:
- The text chunk clearly contradicts a memory.
- A plan or fact is explicitly canceled or replaced.

How to apply:
- Only delete when evidence is explicit.

Constraints:
- If uncertain, prefer no action over deletion.

Action type: DELETE only.

Evolved Skill on LoCoMo - HANDLE_ENTITY_RELATIONSHIPS

Skill: Handle Entity Relationships

Purpose: Capture and manage complex relationships between entities mentioned in the text chunk, including nuanced details.

When to use:
- The text chunk mentions interactions, associations, or relationships between entities with specific details.

How to apply:
- Identify the entities involved and their roles in the relationship.
- Capture the nature of the relationship and any nuanced details (e.g., nicknames, comparative statements).
- Update existing memories to reflect the new relationship information.

Constraints:
- Focus on explicit relationships mentioned in the text chunk.
- Avoid inferring relationships not directly stated.

Action type: INSERT only.

Evolved Skill on LoCoMo - INSERT

Skill: Insert New Memory

Purpose: Capture new, durable facts from the current text chunk that are missing in memory, including specific temporal details such as dates or time frames and detailed activity information.

When to use:
- The text chunk introduces new facts, events, plans, or context worth storing.
- The information is stable and likely useful later.

How to apply:
- Compare against retrieved memories to avoid duplicates.
- Split distinct facts into separate items.
- Keep each item concise and specific, including relevant temporal information and activity details.

Constraints:
- Skip trivial, fleeting, or speculative content.
- Do not update or delete existing memories.

Action type: INSERT only.

Evolved Skill on LoCoMo - NOOP

Skill: No Operation

Purpose: Confirm no memory changes are needed for the text chunk.

When to use:
- The text chunk contains no new, corrective, or actionable information.

Constraints:
- Emit NOOP only if none of the selected skills produce actions.

Action type: NOOP only.

Evolved Skill on LoCoMo - REFINE_TEMPORAL_DETAILS_WITH_CONTEXT

Skill: Refine Temporal Details with Context

Purpose: Update the temporal context of existing memories with new information from the text chunk, considering the context in which the information is provided.

When to use:
- The text chunk provides new or corrected temporal information relevant to an existing memory, and the context suggests a need for refinement.

How to apply:
- Identify the relevant existing memory and its current temporal context.
- Update the temporal details to reflect the new information, ensuring consistency with the provided context.

Constraints:
- Focus on explicit temporal information mentioned in the text chunk and supported by the context.
- Avoid inferring temporal details not directly stated or implied by the context.

Action type: UPDATE only.

Evolved Skill on LoCoMo - UPDATE

Skill: Update Existing Memory

Purpose: Revise a retrieved memory with new or corrected information from the text chunk, including entity-specific details.

When to use:
- The text chunk clarifies, corrects, or extends a retrieved memory.
- The text chunk provides new information about a specific entity or its activities.

How to apply:
- Select the best matching memory item.
- Merge new details into a single updated item.
- Preserve accurate details that still hold, and ensure entity-specific information is accurately captured and updated.

Constraints:
- Do not create new memories.
- Do not delete items.

Action type: UPDATE only.

C.3 Evolved Skills on ALFWorld

Evolved Skill on ALFWorld - CAPTURE_ACTION_CONSTRAINTS

Skill: Capture Action Constraints

Purpose: Capture detailed constraints on actions, including object states and movements, necessary for task completion.

When to use:
- The text chunk mentions constraints on actions, including object states and movements.
- The constraints are crucial for future task steps.

How to apply:
- Identify the action, its constraints, and relevant object states and movements from the text chunk.
- Create a new memory item with the action-constraint pair, including object states and movements.

Constraints:
- Only capture constraints on actions relevant to the task.
- Update existing constraint memories if new information is provided.

Action type: INSERT only.

Evolved Skill on ALFWorld - DELETE

Skill: Delete Invalid Memory

Purpose: Remove a retrieved memory that is wrong, outdated, or superseded by the text chunk.

When to use:
- The text chunk clearly contradicts a memory.
- A plan or fact is explicitly canceled or replaced.

How to apply:
- Only delete when evidence is explicit.

Constraints:
- If uncertain, prefer no action over deletion.

Action type: DELETE only.

Evolved Skill on ALFWorld - INSERT

Skill: Insert New Memory

Purpose: Capture new, durable facts, including procedural knowledge and action sequences, from the current text chunk that are missing in memory.

When to use:
- The text chunk introduces new facts, events, plans, or context worth storing.
- The information is stable and likely useful later.

How to apply:
- Compare against retrieved memories to avoid duplicates.
- Split distinct facts into separate items, including action sequences.
- Keep each item concise and specific.

Constraints:
- Skip trivial, fleeting, or speculative content.
- Do not update or delete existing memories.

Action type: INSERT only.

Evolved Skill on ALFWorld - NOOP

Skill: No Operation

Purpose: Confirm no memory changes are needed for the text chunk.

When to use:
- The text chunk contains no new, corrective, or actionable information.

Constraints:
- Emit NOOP only if none of the selected skills produce actions.

Action type: NOOP only.

Evolved Skill on ALFWorld - TRACK_OBJECT_LOCATION

Skill: Track Object Location

Purpose: Explicitly track the location and state of an object necessary for task completion.

When to use:
- The text chunk mentions an object's location or state.
- The object's location or state is crucial for future task steps.

How to apply:
- Identify the object, its location, and relevant state from the text chunk.
- Create a new memory item with the object-location-state triplet.

Constraints:
- Only track locations and states of objects relevant to the task.
- Update existing location memories if new information is provided.

Action type: INSERT only.

Evolved Skill on ALFWorld - TRACK_OBJECT_MOVEMENTS

Skill: Track Object Movements

Purpose: Track movements of objects necessary for task completion.

When to use:
- The text chunk mentions an object's movement.
- The object's movement is crucial for future task steps.

How to apply:
- Identify the object and its movement from the text chunk.
- Create a new memory item with the object-movement pair.

Constraints:
- Only track movements of objects relevant to the task.
- Update existing movement memories if new information is provided.

Action type: INSERT only.

Evolved Skill on ALFWorld - UPDATE

Skill: Update Existing Memory

Purpose: Revise a retrieved memory with new or corrected information from the text chunk.

When to use:
- The text chunk clarifies, corrects, or extends a retrieved memory.

How to apply:
- Select the best matching memory item.
- Merge new details into a single updated item.
- Preserve accurate details that still hold.

Constraints:
- Do not create new memories.
- Do not delete items.

Action type: UPDATE only.

D Prompts

LoCoMo Answer Prompt

Based on the above context, write an answer in the form of a short phrase for the following question. Answer with exact words from the context whenever possible.

Question: {} Short answer:

LongMemEval Answer Prompt

I will give you several history chats between you and a user. Please answer the question based on the relevant chat history.
History Chats: {}
Current Date: {}
Question: {}
Short Answer:

HotpotQA Answer Prompt

Based on the following context, answer the question. The question may require reasoning across multiple pieces of information.
{context}
Question: {question}

Instructions:
- Read the context carefully and identify relevant information
- If the answer can be found in the context, provide a short, precise answer
- Output your answer within <answer></answer>tags

<answer>your answer here</answer>

ALFWorld Env Interaction Prompt

You are controlling a text-based ALFWorld environment. Your job: choose the NEXT action as ONE text command. Output ONLY the command string, with no extra text. You MUST choose an action from the admissible actions list and copy it EXACTLY.

Goal: {goal}
Retrieved procedural tips (optional, short & actionable): {retrieved_memory}
Interaction history so far (most recent info matters most): {history}
Admissible actions (choose exactly ONE and copy it verbatim): {actions}

Now output exactly one line: the chosen action (must match one item above).

Executor Prompt

You are a memory management executor. Apply the selected skills to the input text chunk and retrieved memories, then output memory actions.

Input Text Chunk: {session_text}
Retrieved Memories (0-based index): {mem_text}
Selected Skills: {skills_text}

Guidelines:
- Apply any skill as needed; a skill may be used multiple times.
- Read the input text chunk carefully line by line and apply any skill as needed.
- Only use action types supported by the selected skills.
- MEMORY_INDEX is 0-based and must reference the retrieved memories list.
- Output only action blocks in the format below.
- Do not include explanations or REASONING lines.

Output format (repeat as needed). Use ONE block per action and separate blocks with a blank line:

INSERT block:
ACTION: INSERT
MEMORY_ITEM: [concise but complete summary with essential details]

UPDATE block:
ACTION: UPDATE
MEMORY_INDEX: [0-based index]
UPDATED_MEMORY: [concise but complete merged summary with essential updates]

DELETE block:
ACTION: DELETE
MEMORY_INDEX: [0-based index]

Designer Analysis Prompt

You are an expert analyst for a memory-augmented QA system. Analyze the failure cases below to identify why the system failed and how the memory management skills should change.

How This System Works
1. **Memory Storage**: The system applies memory management skills to decide what information to store from the text chunk.
2. **Memory Retrieval**: At question time, it retrieves the most relevant memories by semantic similarity.
3. **Answer Generation**: An LLM answers using the retrieved memories.

Failures can occur at any stage:
- **Storage failure**: Important information was never stored (skill missing or misapplied)
- **Retrieval failure**: Relevant memory exists but was not retrieved (embedding mismatch)
- **Memory quality failure**: Memory exists but is too vague or incomplete to answer

Current Memory Management Skills
{operation_bank_description}

Operation Evolution Feedback
{evolution_feedback}

Failure Cases ({num_failure_cases} cases)
{failure_cases_details}

Analysis Instructions
This is round 1 of a reflection loop. Produce a strong initial analysis that can be critiqued and improved.
1. For each case, check whether the retrieved memories contain the answer or the needed evidence.
2. If missing, decide whether it was never stored (storage failure) or stored but too weak (memory quality failure).
3. If the answer is present but not retrieved, label it retrieval failure and avoid changing skills unless the pattern repeats.
4. Group cases into patterns tied to information types, entities, temporal details, or constraints.
5. For each pattern, propose a concrete skill change: add a new skill or refine an existing one to capture missing details.
6. Provide up to {max_changes} recommendations total (use fewer if only one change is justified).

Output Format
Provide your analysis as JSON:
{ “failure_patterns": [{ “pattern_name": “[descriptive name for this failure pattern]", “affected_cases": [list of case numbers, e.g., 1, 3, 5], “root_cause": “[storage_failure|retrieval_failure|memory_quality_failure]", “explanation": “[why this pattern of failures is occurring]", “potential_fix": “[what kind of operation change could address this]" }], “recommendations": [{ “action": “[add_new_operation|refine_existing_operation|no_change]", “target_operation": “[operation name to refine, or null if adding new]", “rationale": “[clear explanation of why this is the best improvement]", “priority": “[high|medium|low]" }], “summary": “[1-2 sentence summary of main findings]" }

Focus on actionable insights. What specific change to the skill bank would prevent these failures?
Output ONLY the JSON, no other text.

Designer Refinement Prompt (1/2)

Based on the failure analysis, propose a specific improvement to the memory operation bank.

Failure Analysis (from Stage 1)
{analysis_feedback}

Current Operation Bank
{operation_bank_full}

{evolution_feedback}

Your Task
Propose up to {max_changes} improvements based on the analysis:

Option A - Add New Operation: Create a new operation if the analysis shows a capability gap (e.g., certain information types are not being captured).
Option B - Refine Existing Operation: Improve an existing operation's instruction template if the analysis shows it's not working well (e.g., memories are too vague, missing key details).
Option C - No Change: If the failures are due to retrieval issues (not operation issues), or if the current operations are already well-designed.

CRITICAL Requirements
1. instruction_template MUST be a skill-style guide and MUST NOT include context placeholders (the executor injects the text chunk and retrieved memories)
2. instruction_template MUST clearly state purpose, when to use, and constraints
3. instruction_template MUST specify the allowed action type (INSERT or UPDATE only)
4. For new operations, 'update_type' must be either "insert" or "update" (delete and noop operations are not evolved at this time)
5. Only propose operations with update_type "insert" or "update"
6. Avoid labels like "ENHANCED", "ADVANCED", or other marketing adjectives in descriptions or templates; keep phrasing neutral and task-specific
7. Do NOT embed output blocks; the executor handles output formatting and can apply the skill multiple times
8. The number of changes in the list MUST be less than max_changes
9. Do NOT modify the same operation more than once in a single response, and do NOT refine an operation you add in the same response

Example of a Well-Designed Insert Operation
{ "name": "extract_personal_preferences", "description": "Memory management skill for capturing personal preferences and habits mentioned in the text chunk.", "update_type": "insert", "instruction_template": "Skill: Insert Preferences. Purpose: Capture personal preferences, habits, or opinions stated in the text chunk. When to use: The text chunk mentions likes, dislikes, routines, or goals tied to a person. How to apply: Attribute the preference to the correct person. Keep the preference specific and actionable. Constraints: Avoid one-off or ambiguous statements. Action type: INSERT only." }

Output Format
Respond with ONE of these JSON structures:
One or more changes (up to max_changes): { "action": "apply_changes", "summary": "[overall rationale for the set of changes]", "changes": [{ "action": "add_new", "new_operation": { "name": "[snake_case_name]", "description": "[what it does and when to use it]", "instruction_template": "[skill-style instruction template]", "update_type": "[insert|update]", "reasoning": "[how this addresses the identified failures]" } }, { "action": "refine_existing", "refined_operation": { "name": "[existing_operation_name]", "changes": { "description": "[improved description]", "instruction_template": "[improved template]" }, "reasoning": "[how these changes address the identified failures]" } }] }

Designer Refinement Prompt (2/2)

No changes needed:
{ “action": “no_change", “reasoning": “[why the current operations are sufficient]" }

Instruction Template Structure
When writing instruction templates, follow this structure:

Skill: [Short skill name]
Purpose: [What this skill does]
When to use:
- [Trigger 1]
- [Trigger 2]
How to apply:
- [Step 1]
- [Step 2]
Constraints:
- [What to avoid]
Action type: [INSERT only | UPDATE only]

Output ONLY the JSON, no other text.

LLM Judge Prompt

You are an expert judge evaluating the quality of an answer for a QA task. Your goal is to determine whether the model's answer correctly and sufficiently answers the given question.

Read the following information carefully:

[Question]
{question}

[Ground Truth Answers]
{ground_truth}

[Model Answer]
{model_answer}

Your evaluation criteria:
1. Correctness:
- Is the model answer factually consistent with ANY of the correct answers?
- Does it avoid contradictions or introducing false information?

2. Relevance:
- Does the answer address the question directly without unnecessary content?

3. Completeness:
- Does the answer include all essential information needed to fully answer the question?
- Partial answers are allowed but should receive lower scores.

Scoring Rules:
- Score = 1.0 if the answer is fully correct.
- Score = 0.5 if the answer is partially correct but incomplete or slightly inaccurate.
- Score = 0.0 if the answer is incorrect, irrelevant, or contradicts the ground truth.

Output Format (STRICT):
Return your output as a JSON dictionary with two fields:
{ "explanation": "<brief explanation of your reasoning>", "score": <0.0 | 0.5 | 1.0>}

Be concise and objective. Do not include anything outside the JSON.

E Use of Large Language Models

LLMs are used as part of our method and evaluation pipeline. Specifically, MemSkill uses an LLM executor for skill-conditioned memory generation, an LLM designer for skill evolution, and an LLM judge for evaluation on tasks where automatic metrics are insufficient. The corresponding prompts are provided in Appendix D.

LLM-based tools may also be used for grammar polishing and code assistance. The authors take full responsibility for all scientific claims, experimental results, figures, and references in this paper.

F Limitations and Societal Impact

Limitations. This work focuses on benchmarked research settings for skill-conditioned memory construction. Practical deployments of agent memory systems may require additional mechanisms for privacy protection, user consent, access control, and data retention. These considerations are orthogonal to the core algorithmic contribution of MemSkill and are left for future deployment-oriented studies.

Societal impact. MemSkill contributes toward more adaptive and reusable memory mechanisms for long-horizon LLM agents. Instead of repeatedly designing task-specific memory pipelines, the proposed skill-based formulation allows memory behaviors to be learned, reused, and evolved from interaction traces, which may lower the engineering barrier for building memory-augmented agents. This can benefit research and practical applications that require long-term context management, such as personalized assistants, educational agents, research support tools, and embodied task-solving systems. Since agent memory may involve user-provided or task-specific information, practical deployments should follow standard data governance practices, including user consent, privacy protection, data retention control, and mechanisms for inspecting or deleting stored memories. Our work focuses on benchmarked research settings and does not target surveillance, deception, or high-stakes decision making.

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: The abstract and introduction summarize the proposed skill-conditioned memory construction formulation, the main components of MemSkill, and the empirical settings used to support the claims.

   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: We discuss the limitations of MemSkill in Appendix F, including benchmarked research settings and practical considerations such as privacy, user consent, access control, and data retention.

   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: [N/A]

Justification: The paper does not include formal theoretical results. We provide the RL training objective and optimization formulation in Appendix B.4.

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 provide the implementation details, model configurations, prompts, training objective, dataset settings, evaluation protocol, and hyperparameters needed to reproduce the main experiments in Section 4 and B.

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 supplemental material?

   Answer: [No]

   Justification: We do not provide codes at submission time to preserve anonymity, but the paper and appendix include detailed implementation details, prompts, dataset settings, hyperparameters, and evaluation protocols needed to reproduce the main results (Section 4 and B). We will fully release the code upon publication.

   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: The paper specifies the datasets and splits, model backbones, training setup, optimizer and hyperparameters, retrieval settings, prompts, and evaluation protocols in Section 4 and B.

   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: [No]

   Justification: We fix the seed for all our experiments, which ensures the reliability of our experiments.

   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 describe the compute resources used for the experiments in Section 4 and Appendix B.

   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, uses standard public benchmarks and evaluation protocols, preserves submission anonymity, and does not involve human-subject experiments, sensitive personal data collection, or high-stakes decision making.

   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: We discuss broader impacts in Appendix F.

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 poses no such risks.

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 use several existing datasets in our work and have ensured that all are properly credited.

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: [N/A]

    Justification: The paper does not release new assets.

    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: The paper does not involve crowdsourcing nor 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 nor research with human subjects.

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: We describe the use of LLMs in Appendix E, including their roles as the executor for skill-conditioned memory generation, the designer for skill evolution, and the judge for evaluation.

    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.