Remember Me, Refine Me: A Dynamic Procedural Memory Framework for Experience-Driven Agent Evolution

Anonymous ACL submission

Abstract

Procedural memory enables large language model (LLM) agents to internalize "how-to" knowledge and thus reduce redundant trial-and-error. However, existing frameworks predominantly suffer from a "passive accumulation" paradigm, treating memory as a static append-only archive. To bridge the gap between static storage and dynamic reasoning, we propose **ReMe** (*Remember Me, Refine Me*), a comprehensive framework for experience-driven agent evolution. ReMe manages the memory lifecycle via three mechanisms: 1) *multi-faceted distillation*, which extracts fine-grained experiences by recognizing success patterns, analyzing failure triggers and generating comparative insights; 2) *context-adaptive reuse*, which tailors historical insights to new contexts through scenario-aware indexing; and 3) *utility-based refinement*, which automatically adds validated memories and prunes outdated ones to maintain a compact, high-quality experience pool. Experiments on BFCL-V3 and AppWorld demonstrate that ReMe establishes a new state-of-the-art in agent memory system. Crucially, we observe a significant memory-scaling effect: Qwen3-8B equipped with ReMe outperforms larger, memoryless Qwen3-14B, indicating that self-evolving memory provides a computation-efficient path for lifelong learning.

1 Introduction

The transition from static language models to autonomous agents marks a pivotal shift in artificial intelligence, enabling systems to handle complex, dynamic tasks through iterative reasoning and tool use (Tao et al., 2024; Gao et al., 2025; Fang et al., 2025). To facilitate continuous improvement without model retraining, **procedural memory**, which internalizes "how-to" knowledge from past interactions, has emerged as a critical substrate for agent evolution (Zhang et al., 2025b; Xu et al., 2025). By accumulating high-quality problem-solving experiences, agents can leverage prior successes and lessons to navigate novel scenarios, theoretically reducing redundant trial-and-error and avoiding local optima (Wang and Chen, 2025; Chen et al., 2025). Figure 1 contrasts how an agent completes one stock trading task with and without experiences.

To bridge the gap between static storage and dynamic reasoning, an ideal procedural memory system must function not merely as a database, but as an evolving cognitive substrate satisfying three core criteria: 1) *High-quality Extraction*: The system should distill generalized, reusable knowledge from noisy execution trajectories, rather than raw, problem-specific observations. 2) *Task-grounded Utilization*: Retrieved memories should be dynamically adapted to the specific requirements of the current task, maximizing their utility in novel scenarios. 3) *Progressive Optimization*: The memory pool should maintain its vitality through continuous updates, autonomously reinforcing effective entries while removing outdated ones to prevent degradation over time.

However, current frameworks often fall short of these criteria, largely constrained by a "passive accumulation" paradigm. Prevailing approaches typically treat memory as inert, static storage, built on either raw trajectories as experiences (Zheng et al., 2024; Hu et al., 2024) or summarized workflows corresponding to entire trajectories (Tang et al., 2025; Liu et al., 2025b). This introduces several fundamental limitations. First, coarse-grained trajectory-level experiences may introduce irrelevant information that can prevent the agent from grasping the core logic. Second, fetched experiences are applied without adaptation, leading to failures in slightly shifted scenarios. Crucially, lack of timely update strategies causes the experience pool to degrade into a mixture of valid insights and toxic noise (Xiong et al., 2025).

To address these challenges, we propose **ReMe** (*Remember Me, Refine Me*), a dynamic procedural memory framework that shifts the paradigm

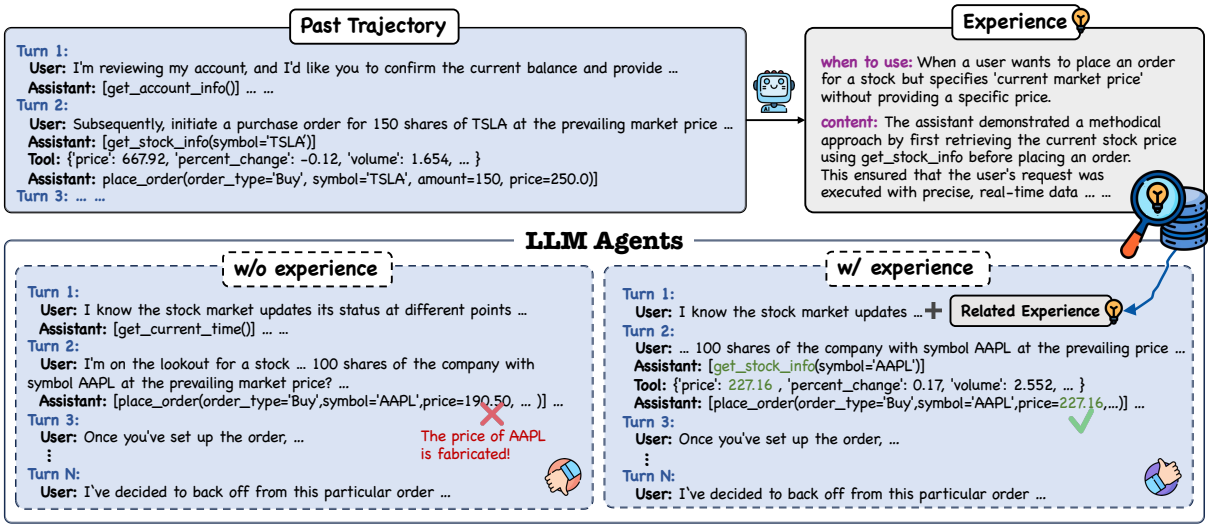

Figure 1: Example of how agents complete one stock trading task with and without past experience.

from passive storage to feedback-driven evolution. We introduce coordinated innovations across the memory lifecycle to meet the criteria of an ideal system. First, ReMe employs a multi-faceted distillation strategy for high-quality extraction. Through success pattern recognition, failure analysis and comparative insight generation, the system distills key steps from past execution trajectories into structured, reusable experiences. Second, we design a comprehensive reuse pipeline for task-grounded utilization. ReMe employs usage scenario indexing strategy for retrieval, supplemented by reranking and adaptive rewriting, aligning historical insights with the specific constraints of new tasks. Finally, ReMe implements a utility-based refinement mechanism for progressive optimization. The memory pool grows as new successful trajectories contribute reliable experiences and failure attempts trigger self-reflection to explore viable solutions for potential insights. Concurrently, our framework tracks the utility of each experience during reuse, periodically pruning low-utility entries to maintain a compact and highly effective memory state.

Through extensive experiments on BFCL-V3 and AppWorld benchmarks, ReMe achieves state-of-the-art performance, demonstrating its effectiveness for memory-augmented agents. Most notably, results reveal that memory quality can substitute for model scale: ReMe enables Qwen3-8B to outperform larger Qwen3-14B (without memory), achieving average gains of 8.83% in Avg@4 and 7.29% in Pass@4. These findings suggest that a self-evolving memory mechanism paves the way for resource-efficient lifelong learning in LLM agents.

In summary, our contributions are as follows:

- We propose **ReMe**, a comprehensive framework for agent evolution that integrates multi-faceted experience distillation, context-adaptive reuse, and utility-based refinement. This achieves the closed loop of procedural memory, resolving the "passive accumulation" dilemma by enabling agents to autonomously distill, adapt, and maintain high-quality reasoning patterns.
- We release **`reme.library`**, a fine-grained procedural memory dataset constructed from diverse agentic tasks, with structured success patterns and failure lessons to serve as a valuable community resource for studying procedure memory and optimizing memory-augmented agents.
- Extensive experiments show that ReMe significantly enhances agent performance across diverse benchmarks. Crucially, we demonstrate a **memory-scaling effect**, where smaller models equipped with ReMe surpass larger baselines, validating our framework as a computationally efficient pathway for lifelong agent learning.

2 Related Works

Memory-enhanced LLM Agents. LLM-based agents excel at handling complex tasks and interactions, fueling their integration into diverse fields, such as finance (Ding et al., 2024), education (Wang et al., 2024) and personalized assistant applications (Abbasian et al., 2023). Contemporary LLM agents employ memory systems that store explored information and reuse these experiences, to enhance their reasoning capabilities

and training efficiency (Mei et al., 2025). In general, memory-enhanced agents often leverage two forms of memory: parametric memory and non-parametric memory (Zhang et al., 2024). Parametric memory refers to encoding long-term knowledge within model weights, while non-parametric memory utilizes external resources like knowledge bases and databases to enrich task contexts without modifying model parameters. WKM (Qiao et al., 2024) incorporates a parametric world-knowledge model to facilitate agent planning. AWM (Wang et al., 2025) enables agents to automatically induce and use task workflows from past experiences, improving their performance on web navigation tasks. MARK (Ganguli et al., 2025) constructs user preference memory to deliver personalized responses in conversational AI systems.

Experience Learning Strategies. Recent studies show LLMs can improve their decision-making abilities through gathering experiences and recalling relevant knowledge (Zhao et al., 2024; Tan et al., 2025). The core of experience learning involves extracting usable information to selectively update the experience pool and retrieving effective experiences to help generate responses. Early approaches, such as Synapse (Zheng et al., 2024) and HiAgent (Hu et al., 2024), store complete trajectories as experiences for retrieval. However, collecting raw and long interaction histories is hard to manage, and the lack of abstraction limits task generalization. Current works (Wang et al., 2025; Chen et al., 2025) focus on summarizing structured knowledge from prior trajectories and implementing a context-aware retrieval system to reuse experiences for task guidance. For instance, Agent KB (Tang et al., 2025) captures generalizable experience units and introduces a teacher-student dual-phase retrieval mechanism that enables complex agentic problem solving. CER (Liu et al., 2025b) distills fine-grained skills and environment dynamics, allowing agents to augment themselves with relevant knowledge in new tasks. These methods neglect strategic experience removal mechanism, since harmful experiences inevitably exist even with human validation and initial helpful ones can also degrade over time (Xiong et al., 2025).

3 Methodology

3.1 Overview of ReMe

Our framework, ReMe, as illustrated in Figure 2, operates through three interconnected phases: experience acquisition, reuse, and refinement. In the ***experience acquisition*** phase, a summarizer analyzes agent generated trajectories (both successful and failed) and distills actionable knowledge into a structured experience pool. During ***experience reuse***, given a novel task, a retriever recalls relevant experiences from the experience pool. These experiences then augment the agent's context, enhancing their reasoning and task-solving performance. Finally, the ***experience refinement*** phase continuously optimizes the experience pool by incorporating new solid experiences and discarding outdated ones, ensuring long-term relevance and adaptability to shifting task demands.

3.2 Experience Acquisition

We begin by defining agentic experiences $\mathcal{E}$ as structured, generalizable representations of agent execution insights. Each individual experience $E \in \mathcal{E}$ is denoted as $E = \langle \omega, e, \kappa, c, \tau \rangle$, where ω states the scenario when to use the experience, e represents the core experience content, $\kappa = \{\kappa_1, \kappa_2, ..., \kappa_m\}$ is a set of relevant keywords for categorization, $c \in [0, 1]$ quantifies the confidence score, and τ enumerates the tools utilized.

To construct the initial experience pool, the execution agent $\text{LLM}_{execute}$ interacts with the environment over time and across the training tasks, incrementally accumulating informative trajectories. For each task query q, we sample trajectories N times aiming to capture diverse execution paths and thereby increase the likelihood of obtaining valuable success/failure pairs for comparisons during ***experience acquisition***.

After collecting a set of exploration trajectories, a summarizer LLM_{summ} is instructed to transform them into structured, reusable experiences through three complementary analyses: First, the summarizer engages in success pattern recognition, identifying effective strategies and distilling the underlying principles from succeeded trajectories. Concurrently, LLM_{summ} conducts failure analysis, scrutinizing unsuccessful attempts to derive valuable lessons. These preventive insights discuss common pitfalls, ineffective approaches, and critical errors that can be used to avoid repeating them in future tasks. Additionally, LLM_{summ} performs comparative analysis by jointly examining successful and failed trajectories, identifying critical differences that distinguish effective from ineffective attempts.

Following the summarization, a validation step leveraging LLM-as-a-Judge (Zheng et al., 2023) is

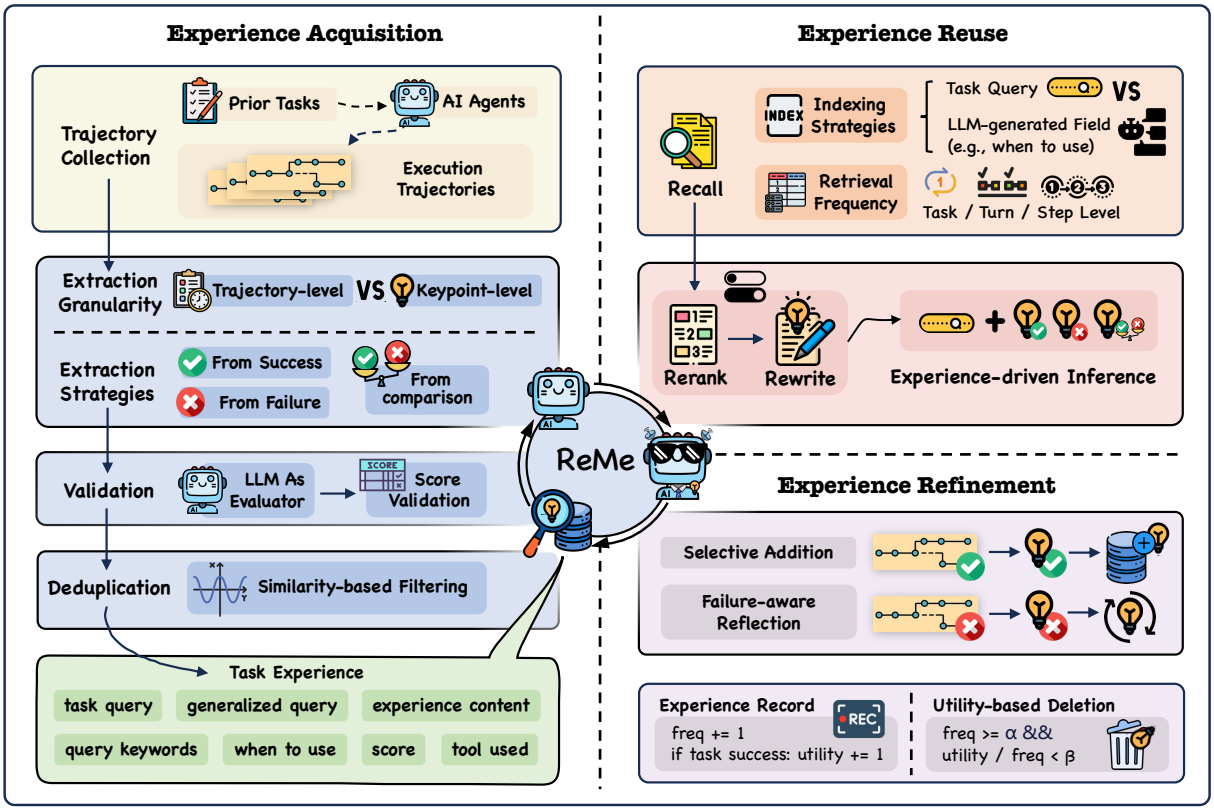

Figure 2: The ReMe framework comprises three alternating phases: build an experience pool from the agent's past trajectories, recall and adapt relevant experiences for new tasks, then refine the pool by selectively adding new insights and removing outdated ones after each task execution.

further applied to assess whether the extracted experiences are actionable, accurate, and valuable for future agent executions. The designed prompt template is presented in Appendix Table 10. Moreover, to keep the experience pool compact, validated experiences undergo a similarity-based deduplication process which removes redundant experiences. This helps maintain the efficiency of the subsequent experience reuse phase and preserve the diversity of retrieved experiences.

All retained experiences are indexed by the embedding vector of usage scenario ω and then stored in a vector database, which we refer to as the experience pool. The multi-faceted experience pool establishes a foundation for efficient retrieval and application of relevant knowledge in future problem-solving scenarios, promoting the agent evolution from trial-and-error to strategic reasoning.

3.3 Experience Reuse

Equipped with the experience pool, we can retrieve top-K relevant experiences based on task similarity, which serve as a candidate set of in-context learning demonstrations to guide $\text{LLM}_{execute}$. To be specific, the retriever utilizes advanced embedding models (e.g., *text-embedding-v4* [1]) to encode the current task query and computes cosine similarity scores to rank prior experiences. More retrieval details can be found in Appendix B.4.2. Upon fetching the top-K experiences, we optionally employ a context-aware reranker LLM_{rerank} to further refine the selection. This involves a nuanced evaluation of experience relevance in light of the current task's specific context, constraints, and objectives, thus ensuring the most pertinent experiences are brought to the forefront.

To better adapt the experiences to new task requirements, we introduce the rewriting module to reorganize the original context (containing multiple experiences) into a cohesive, task-specific guidance that is more directly applicable. Since past experiences may not always perfectly align with new situations, this intelligent adaptation mechanism not only increases the immediate utility of the retrieved experiences but also empowers the agent to make more flexible and context-aware decisions.

[1] https://bailian.console.aliyun.com/?tab=model#/model-market/detail/text-embedding-v4

The ***experience reuse*** phase extends beyond mere experience retrieval, acting as a cognitive bridge that dynamically connects past knowledge with present challenges. By combining retrieval, reranking and rewriting, it not only leverages prior wisdom but also encourages novel thinking when past experiences fall short, thereby achieving a balance between exploitation and exploration.

3.4 Experience Refinement

However, a static experience pool cannot adapt to shifts in task distributions or improvements in model capability, making retrieved experiences increasingly irrelevant. To address this, we introduce a ***experience refinement*** mechanism that dynamically updates the experience pool via selective addition and utility-based deletion.

First, we carefully compare two distinct strategies for adding new experiences to the pool: 1) full addition, which incorporates experiences summarized from all new trajectories regardless of outcome; 2) selective addition, where only trajectories that lead to success are distilled into experiences and stored. The empirical evidence indicates that full addition often underperforms selective addition, which may be attributed to the quality of failure-based experiences. During initial experience pool construction, multiple failed trajectories can be collectively analyzed to extract meaningful insights. However, in real-time task execution, a single failed trajectory often provides insufficient context for accurate failure analysis, potentially leading to misguided experiences. In contrast, successful trajectories consistently yield more reliable and actionable insights, thereby making selective addition effective.

Additionally, we recognize the potential value of learning from failures and introduce a failure-aware reflection mechanism that encourages agents to explore alternative strategies when encountering new task failures. Specifically, LLM_{summ} analyzes this unsuccessful attempt, extracts key insights about potential areas for improvement, and then $\text{LLM}_{execute}$ starts a new trial based on these lessons. When such trial succeeds, the corresponding lessons are incorporated into memory; otherwise, they are discarded without cluttering the experience pool. To avoid falling into an endless loop caused by inherent model limitations, we limit the maximum number of self-reflections to 3.

Second, to prevent the accumulation of outdated or ineffective experiences, we employ a utility-based deletion strategy that removes any experience whose average utility across all its past recalls falls below a predefined threshold β. Specifically, `ReMe` continuously records the status of existing experiences, including the total retrievals f and the historical utility u which increments by 1 each time its recall contributes to a successful task completion. An experience $E \in \mathcal{E}$ is considered to be removed when it is frequently retrieved yet fails to improve new task performance:

$$\phi_{remove}(E) = \begin{cases} \mathbb{1}\left[\frac{u(E)}{f(E)} \leq \beta\right], & \text{if } f(E) \geq \alpha, \\ 0\ , & \text{otherwise.} \end{cases} \tag{1}$$

Note that we only consider an experience for removal after it has been retrieved at least α times.

By integrating these components, `ReMe` facilitates a self-evolving experience pool that retains high-quality experiences capable of shaping long-term agent behavior while adapting to new task demands.

4 Experiments

4.1 Experimental Settings

Datasets. We conduct experiments on two tool-augmented benchmarks: BFCL-V3 (Patil et al., 2025), AppWorld (Trivedi et al., 2024). For BFCL-V3, we randomly select 50 tasks from the base multi-turn category to construct the initial experience pool since the default dataset does not provide training split. The remaining 150 tasks serve as the evaluation set. For AppWorld, we use 90 training tasks for the initial experience acquisition stage and evaluate agents on 168 test-normal tasks. Detailed information of the datasets are in Appendix B.1.

Metrics. We report both Avg@4 and Pass@4 results: the average task success rate across four independent trials, and the probability that at least one out of four independent task trials is successful. Unless otherwise specified, all results are averaged over three independent runs and reported as mean with standard deviation.

Baselines. To evaluate the effectiveness of `ReMe`, we compare it against three baselines: (1) No Memory, and two popular baseline memory systems (2) A-Mem (Xu et al., 2025), an agentic memory system that enables LLM agents to dynamically organize their memories for future action guidance, and (3) LangMem (LangChain, 2025), LangChain's long-term memory module that provides tooling to

Model	Methods	BFCL-V3		AppWorld		Avg	
		Avg@4	Pass@4	Avg@4	Pass@4	Avg@4	Pass@4
Qwen3-8B	No Memory	$40.33^{\pm 0.94}$	$59.55^{\pm 0.83}$	$14.97^{\pm 0.24}$	$32.85^{\pm 2.11}$	27.65	46.20
	A-Mem	$41.22^{\pm 0.61}$	$62.00^{\pm 2.37}$	$12.95^{\pm 0.37}$	$29.76^{\pm 2.80}$	27.09	45.88
	LangMem	$44.11^{\pm 0.28}$	$65.55^{\pm 1.13}$	$11.46^{\pm 0.53}$	$26.79^{\pm 0.84}$	27.79	46.17
	`ReMe (fixed)`	$44.50^{\pm 0.85}$	$65.77^{\pm 0.63}$	$17.06^{\pm 0.25}$	$36.31^{\pm 1.29}$	30.78	51.04
	`ReMe (dynamic)`	$\mathbf{45.17}^{\pm 0.36}$	$\mathbf{68.00}^{\pm 0.55}$	$\mathbf{24.70}^{\pm 1.04}$	$\mathbf{42.06}^{\pm 0.74}$	**34.94**	**55.03**
Qwen3-14B	No Memory	$48.66^{\pm 1.51}$	$68.22^{\pm 0.63}$	$22.57^{\pm 0.19}$	$41.07^{\pm 0.84}$	35.62	54.65
	A-Mem	$47.44^{\pm 0.44}$	$69.77^{\pm 0.63}$	$18.95^{\pm 0.31}$	$37.70^{\pm 0.57}$	33.20	53.74
	LangMem	$49.17^{\pm 0.33}$	$71.33^{\pm 1.33}$	$21.88^{\pm 1.37}$	$41.67^{\pm 1.68}$	35.53	56.50
	`ReMe (fixed)`	$51.89^{\pm 0.34}$	$72.44^{\pm 0.63}$	$25.35^{\pm 0.91}$	$46.82^{\pm 0.74}$	38.62	59.63
	`ReMe (dynamic)`	$\mathbf{55.00}^{\pm 0.72}$	$\mathbf{74.44}^{\pm 0.83}$	$\mathbf{34.32}^{\pm 0.81}$	$\mathbf{52.98}^{\pm 1.29}$	**44.66**	**63.71**
Qwen3-32B	No Memory	$54.55^{\pm 0.63}$	$72.44^{\pm 0.83}$	$27.23^{\pm 0.92}$	$50.59^{\pm 1.68}$	40.89	61.52
	A-Mem	$54.50^{\pm 1.09}$	$72.66^{\pm 0.54}$	$28.13^{\pm 0.75}$	$51.19^{\pm 0.97}$	41.32	61.93
	LangMem	$52.27^{\pm 1.13}$	$72.22^{\pm 1.91}$	$24.55^{\pm 0.57}$	$47.02^{\pm 1.56}$	38.41	59.62
	`ReMe (fixed)`	$56.05^{\pm 1.26}$	$74.89^{\pm 0.63}$	$31.50^{\pm 0.67}$	$58.13^{\pm 1.40}$	43.78	66.51
	`ReMe (dynamic)`	$\mathbf{56.17}^{\pm 0.24}$	$\mathbf{76.44}^{\pm 1.13}$	$\mathbf{42.02}^{\pm 0.51}$	$\mathbf{63.49}^{\pm 0.28}$	**49.10**	**69.97**

Table 1: Performance comparison (%) between `ReMe` and the baselines on BFCL-V3, AppWorld benchmarks. **Bold** indicate the best results of each model. All results are computed as the average over three independent runs, with the superscript showing the standard deviation.

extract important information from previous conversations and optimize agent behavior through prompt refinement. For fair comparison, all methods perform experience retrieval only once at the beginning of each task. Additionally, the memory addition operation for these systems is triggered only upon the collection of successful trajectories. Further implementation details of the baseline methods are provided in Appendix B.2.

Implementation Details. We use the Qwen3 series instruct models (Team, 2025b) as $\text{LLM}_{execute}$ and set $\text{LLM}_{summ} = \text{LLM}_{execute}$ for experience-driven self-evolution. For experience indexing, we employ Qwen3-Embedding (Zhang et al., 2025a) with its default embedding dimension of 1024. In the experience acquisition phase, we set $N = 8$ and temperature $= 0.9$ for trajectory sampling. The prompts used in this phase and more details can be found in Appendix B.4.1. In the experience reuse phase, we use a top-K value of 5, retrieving the five most relevant experiences for each task. The configuration difference between `ReMe (fixed)` and `ReMe (dynamic)` lies in whether the experience pool is dynamically updated during agent execution. In the experience refinement phase, utility-based deletion is controlled by the retrieval threshold $\alpha = 5$ and the utility threshold $\beta = 0.5$. Additionally, the maximum number of iterations is limited to 30, after which the agent terminates regardless of task success or failure. To ensure fair comparison, we maintain these settings consistently across all experiments unless otherwise specified for ablation studies.

4.2 Main Results

Table 1 presents the main results of `ReMe` across Qwen3 family models on BFCL-V3 and AppWorld benchmarks. Overall, `ReMe` achieves the highest average task success rate across three model sizes, consistently outperforming No Memory baseline and competitive baseline memory systems. Specifically, Qwen3-8B with `ReMe` surpasses the No Memory baseline by an improvement of 7.29% Pass@4 and 8.83% Avg@4 on average. The gains observed in Pass@4 indicate that retrieved experiences are effective at broadening the exploration space, increasing the likelihood of finding at least one successful solution among multiple attempts. Besides, the performance stability of our `ReMe` is particularly evident when compared to the baseline methods. For instance, while LangMem performs well on BFCL-V3, its performance drops significantly on AppWorld, especially for smaller models. Instead, `ReMe (dynamic)` shows remarkable consistency across both BFCL-V3 and AppWorld benchmarks.

Notably, smaller models equipped with `ReMe` can be comparable to, or even surpass, larger models without memory. For example, the average Pass@4

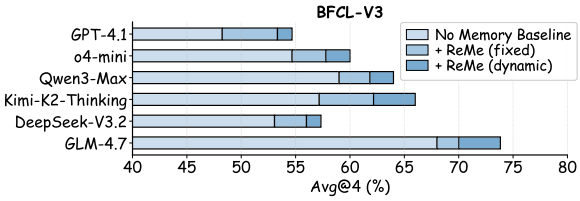

Figure 3: Performance improvements (%) for multiple LLMs enhanced with ReMe.

Granularity	Qwen3-8B		Qwen3-14B	
	Avg@4(%)	Pass@4(%)	Avg@4(%)	Pass@4(%)
Trajectory-level	$43.00_{+2.67}$	$60.00_{+0.45}$	$49.66_{+1.00}$	$69.33_{+1.11}$
Keypoint-level	$44.50_{+4.17}$	$65.77_{+6.22}$	$51.89_{+4.23}$	$72.44_{+4.22}$

Table 2: Ablation study on extraction granularity levels in the experience acquisition stage. The experimental setting is ReMe (fixed), with subscript showing the performance gap compared with No Memory baseline.

score for Qwen3-8B + ReMe (dynamic) exceeds that of vanilla Qwen3-14B (55.03% vs. 54.65%). Similarly, Qwen3-14B + ReMe (dynamic) exceeds the overall performance of Qwen3-32B without memory (Avg@4: 44.66% vs. 40.89%; Pass@4: 63.71% vs. 61.52%). This underscores that an effective memory mechanism can significantly narrow the performance gap across model scales.

Moreover, ReMe (dynamic) consistently outperforms ReMe (fixed) across all model sizes and benchmarks. This underscores the importance of adaptive experience refinement during task execution. In addition, ReMe tends to reduce the standard deviation in performance across runs, particularly for larger models. This suggests that ReMe not only improves overall performance but also enhances the robustness and reliability of model outputs.

To demonstrate the generalizability of ReMe across different backbones, we evaluate on six additional LLMs as shown in Figure 3. ReMe delivers consistent gains in Avg@4 across all models, with the dynamic variant yielding further improvements.

4.3 Ablation Studies

Granularity Ablations. We compare two granularity levels for experience acquisition: trajectory-level and keypoint-level. In Appendix C, we present two experience examples illustrating the structural and content differences between these granularity settings. As shown in Table 2, although the incorporation of trajectory-level experiences exhibits minor progress over No Memory baseline, the performance gains brought by keypoint-level experiences are substantially higher. This

Full Addition	Selective Addition	Reflection	Deletion	BFCL-V3	
				Avg@4	Pass@4
✓	–	–	–	40.83%	62.00%
–	✓	–	–	44.33%	64.66%
–	✓	✓	–	45.00%	64.66%
–	✓	✓	✓	**45.17%**	**68.00%**

Table 3: Ablation on key components. We compare the full addition and selective addition and assess the impact of failure-aware reflection and utility-based deletion. A checkmark (✓) indicates the component is used.

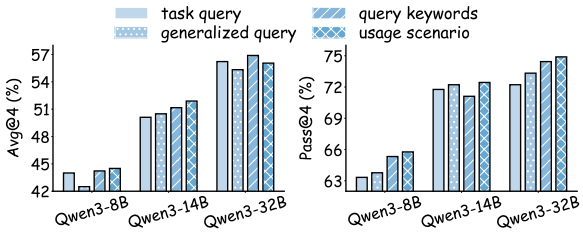

Figure 4: Ablation on retrieval keys. The experiments are evaluated on BFCL-V3 in ReMe (fixed) setting.

underscores that summarizing experiences at a fine-grained level enables more effective knowledge transfer, leading to superior agent performance across different tasks and model scales.

Component Ablations. Taking Qwen3-8B as an example, Table 3 presents an ablation study on key components of our ReMe framework. Firstly, replacing full addition with selective addition leads to substantial performance improvements, with gains of 3.50% Avg@4 and 2.66% Pass@4 on BFCL-V3. This highlights the importance of experience quality over quantity in experience-driven agent evolution. Moreover, the introduction of the failure-aware reflection module enhances the average task success rate, demonstrating the value of learning from unsuccessful attempts. Notably, incorporating the utility-based deletion yields further improvements, indicating that regularly discarding outdated experiences is critical for agents to adapt to non-stationary environments.

Retrieval Key Ablations. Regarding the indexing strategy, we explore four different retrieval keys to assess their impact on the performance of ReMe. From Figure 4, it can be seen that using the raw task description or their extracted keywords to index experiences underperforms the LLM-generated fields (generalized query and usage scenario). The usage scenario indexing strategy, which likely captures both the task context and potential application

$\text{LLM}_{\text{execute}}$	LLM_{summ}	BFCL-V3	
		Avg@4 (%)	Pass@4 (%)
Qwen3-8B	Qwen3-8B	44.50	65.77
	Qwen3-14B	46.33 $\triangle = 1.83 \uparrow$	66.00 $\triangle = 0.23 \uparrow$
	Qwen3-32B	47.83 $\triangle = 3.33 \uparrow$	68.00 $\triangle = 2.23 \uparrow$

Table 4: Performance of different LLM_{summ} capabilities with fixed $\text{LLM}_{execute}$ in ReMe (fixed) setting.

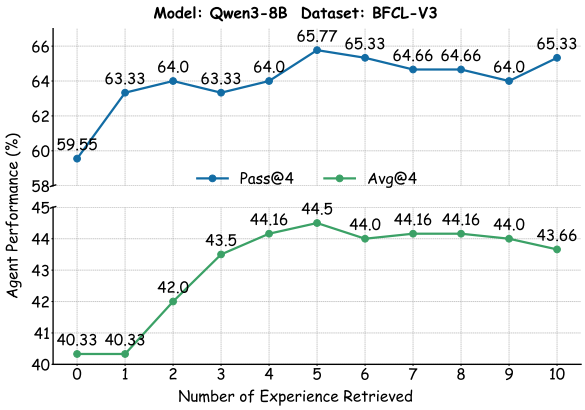

Figure 5: Effect of retrieved experience number on agent performance (%) in ReMe (fixed) setting.

areas, proves to be the most effective in retrieving relevant experiences from the database. For comprehensive results, please refer to Appendix D.2.

4.4 More Analysis

Agent Gains More with Stronger LLM_{summ}. Our main experiments demonstrate an agent can learn effectively through experience-driven self-evolution, i.e., LLM_{summ}=$\text{LLM}_{execute}$. To investigate whether the agent gains more as LLM_{summ} capability increases, we scale the summarization model from Qwen3-8B to Qwen3-32B with the fixed $\text{LLM}_{execute}$ = Qwen3-8B. It can be observed from Table 4 that stronger summarization capability yields clear performance improvements in both Avg@4 and Pass@4 metrics (Avg@4: +1.83% $\rightarrow$ +3.33%; Pass@4: +0.23% $\rightarrow$ +2.23%). These findings emphasize the critical role of high-quality experience summarization in overall agent performance, highlighting the potential for further gains through advanced summarization techniques.

Effect of Retrieved Experience Number. To evaluate the relationship between retrieved experience number and performance, we vary the value K from 0 to 10. Figure 5 illustrates increasing the number of in-context experiences achieves steady performance gains that rise and then saturate. Beyond the saturation point, retrieving more may degrade performance, primarily due to the higher chance of incorporating noisy experiences. This is why we select $K = 5$ in the main experiments.

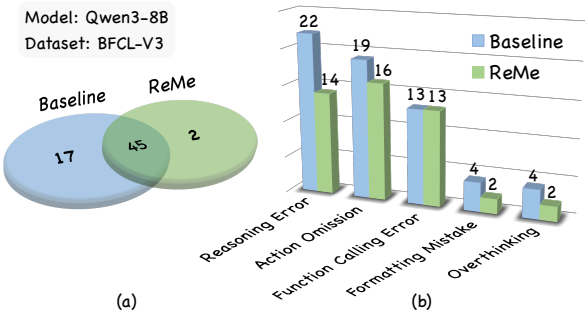

Figure 6: Statistics of failed tasks with and without ReMe. (a) Left: shows overlapping and unique failure cases; (b) Right: displays the number of task failures across different error categories.

Error Analysis. We conduct an analysis of the error patterns with and without ReMe for Qwen3-8B on BFCL-V3. The Venn diagram (Figure 6a) reveals a reduction in the total number of failure cases from 62 (No Memory Baseline) to 47 (ReMe). Notably, ReMe corrects 17 baseline-specific errors while introduces only 2 new ones. Further, we manually review and categorize each failure case to examine the impact of ReMe on different error types (Figure 6b). A substantial decrease in `Reasoning Error` (22 $\rightarrow$ 14) indicates ReMe effectively leverages past experiences to strengthen its multi-step reasoning capabilities, reducing propagation of earlier mistakes. ReMe also yields a moderate but meaningful reduction in `Action Omission` errors, which helps the agent recognize missing steps in multi-turn tasks, especially those requiring sequential tool interactions or state tracking.

5 Conclusion

We introduce ReMe, a dynamic procedural memory framework that evolves agent reasoning from blind trial-and-error to strategic experience reuse. By distilling structured knowledge from prior trajectories at a fine-grained level, ReMe enables agents to leverage critical insights, thus avoiding potential experience interference in coarse-grained approaches. Equipped with effective experience refinement, ReMe maintains a high-quality experience pool for agent evolution. Extensive experiments validate that ReMe significantly outperforms several baselines, with ablation studies highlighting the value of each core component in ReMe.

Limitations

This paper focuses on procedural memory management for agent self-evolution. Despite its promising performance, there are several limitations that could be addressed in future work. First, ReMe currently employs a fixed retrieval strategy, where experiences are retrieved once at the beginning of each task. Implementing a more flexible, context-aware retrieval mechanism could potentially improve system performance, since dynamic experience incorporation promotes adaptive knowledge utilization. Secondly, although the existing experience validation process effectively filters out low-quality experiences, relying primarily on an LLM-as-judge approach may overlook nuanced aspects of experience quality and relevance. In the future, we can explore more sophisticated validation techniques for more precise experience evaluation. Furthermore, a larger-scale summarizer brings greater performance gains in agent reasoning, as shown in Section 4.4, which can be attributed to its stronger summarization capability. This indicates that designing advanced summarization strategies with small models can further boost agent self-evolution.

References

Mahyar Abbasian, Iman Azimi, Amir M Rahmani, and Ramesh Jain. 2023. Conversational health agents: A personalized llm-powered agent framework. *arXiv preprint arXiv:2310.02374*.

Silin Chen, Shaoxin Lin, Xiaodong Gu, Yuling Shi, Heng Lian, Longfei Yun, Dong Chen, Weiguo Sun, Lin Cao, and Qianxiang Wang. 2025. Swe-exp: Experience-driven software issue resolution. *arXiv preprint arXiv:2507.23361*.

Han Ding, Yinheng Li, Junhao Wang, and Hang Chen. 2024. Large language model agent in financial trading: A survey. *arXiv preprint arXiv:2408.06361*.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, and 1 others. 2025. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*.

Anish Ganguli, Prabal Deb, and Debleena Banerjee. 2025. Mark: Memory augmented refinement of knowledge. *arXiv preprint arXiv:2505.05177*.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. 2024. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.09559*.

LangChain. 2025. Langmem: Modular memory for agentic systems. https://github.com/langchain-ai/langmem. Accessed: 2025-10-13.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025a. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

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

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, and 1 others. 2025. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*.

MoonshotAI. 2025. Introducing kimi k2 thinking.

OpenAI. 2025a. Introducing gpt-4.1 in the api.

OpenAI. 2025b. Introducing openai o3 and o4-mini.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2024. Agent planning with world knowledge model. *Advances in Neural Information Processing Systems*, 37:114843–114871.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, Anand Rajan Iyer, Tianlong Chen, Huan Liu, Chen-Yu Lee, and Tomas Pfister. 2025. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8416–8439.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, and 1 others. 2025. Agent kb: Leveraging cross-domain experience for agentic problem solving. *arXiv preprint arXiv:2507.06229*.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. 2024. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*.

Qwen Team. 2025a. Qwen3-max: Just scale it.

Qwen Team. 2025b. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. 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.

Shen Wang, Tianlong Xu, Hang Li, Chaoli Zhang, Joleen Liang, Jiliang Tang, Philip S Yu, and Qingsong Wen. 2024. Large language models for education: A survey and outlook. *arXiv preprint arXiv:2403.18105*.

Yu Wang and Xi Chen. 2025. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025. Agent workflow memory. In *Forty-second International Conference on Machine Learning*.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. 2025. How memory management impacts llm agents: An empirical study of experience-following behavior. *arXiv preprint arXiv:2505.16067*.

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

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

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024. A survey on the memory mechanism of large language model based agents. *ACM Transactions on Information Systems*.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025b. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*.

ZhipuAI. 2025. Glm-4.7: Advancing the coding capability.

A Use of AI Assistants

During the writing process of this paper, we utilized AI assistants for language refinement, including tasks such as grammar checking, sentence restructuring, and phrasing. All AI-suggested changes were thoroughly reviewed and approved by the authors to ensure the final content of this paper represents the authors' original ideas.

B Experimental Details

In this section, we detail our experimental setup, including the benchmark datasets, evaluation metrics, and baseline methods compared. We also describe the implementation details of `ReMe` crucial for reproducing our results.

B.1 Datasets and Evaluation Metrics

BFCL-V3 Berkeley Function Calling Leaderboard V3 (BFCL-V3) (Patil et al., 2025) is a benchmark which assesses the function calling and tool-using capabilities of LLMs, particularly in multi-turn and multi-step scenarios. It provides over 1,800 test tasks that require models to generate precise API calls, handle various programming languages (Python, Java, JavaScript), and manage complex interactions like parallel function calls. The evaluation employs both Abstract Syntax Tree (AST) matching to check syntactic correctness and executable testing to verify functional outcomes. In our experiments, a task is deemed successful when the agent makes the necessary function calls correctly and yields the expected outputs.

task query: Access and retrieve the details of my most recent order, as I've misplaced the ID but need the latest transaction.
query keywords: ["order retrieval", "ambiguous requests", "efficiency", "user experience"]
generalized query: Retrieve recent order details when order ID is unavailable.
when to use: When users need order details without explicit order IDs.
experience content: The higher-scoring approach automatically retrieved and displayed the most recent order details (ID 12446) after fetching the history, while the lower-scoring response only listed order IDs without immediate detail retrieval. This demonstrates efficiency in handling ambiguous user requests by combining history lookup with direct detail fetching.

Figure 7: Different indexing examples for the same BFCL-V3 task experience.

AppWorld AppWorld (Trivedi et al., 2024) is a benchmark designed to evaluate function calling and interactive coding agents. It simulates a world of 9 day-to-day applications (e.g., email, Spotify, Venmo) through 457 APIs and is populated with the digital activities of approximately 100 simulated users. A key feature of AppWorld is its robust evaluation framework, which uses state-based unit tests to assess task completion and provides two metrics to measure performance: 1) Task Goal Completion (TGC) measures percentage of tasks for which the agent passes all evaluation tests; 2) Scenario Goal Completion (SGC) is the percentage of scenarios where the agent passes all the unit tests for all tasks from that scenario. In our experiments, we report Task Goal Completion metric, which naturally reflects task success rate.

B.2 Baseline Details

LangMem LangMem (LangChain, 2025) is Langchain's long-term memory module that extracts and stores key information from conversations for future retrieval. It provides both functional primitives compatible with any storage system and native integration with LangGraph's storage layer, enabling agents to continuously improve. In our experiments, we adopt LangMem's implementation of episodic memory[2], which helps the agent learn from experience.

A-Mem A-Mem (Xu et al., 2025) is a system designed to provide LLM agents with agentic memory, allowing them to autonomously manage their own long-term knowledge. It constructs a memory-centric knowledge graph for agents, actively deciding what information to store, recall, and update based on their goals and interaction. In our experiments, we reproduce A-Mem using its open-source code, with slight prompt modifications to extract procedural memories.

[2]https://langchain-ai.github.io/langmem/guides/extract_episodic_memories/

when to use: When a user wants to place an order for a stock but without providing a specific price.
experience content: The assistant demonstrated a methodical approach by first retrieving the current stock price using get_stock_info and then using that price in the place_order function. This two-step process ensures compliance with the required parameters of the place_order function while aligning with the user's intent for a market-price-based order.

Figure 8: Experience example on BFCL-V3.

when to use: When interacting with APIs that require precise authentication parameters and data extraction.
experience content: The higher-scoring approach prioritized API specification validation before execution (e.g., confirming phone login requires phone number as username), implemented robust error handling for authentication failures, and used precise data extraction techniques (search_notes with tags/query filters). The lower-scoring approach made repeated authentication errors, included explanatory text in code blocks causing syntax failures, and used inefficient string parsing that retained metadata instead of clean titles.

Figure 9: Experience example on AppWorld.

B.3 Model Configuration

We select Qwen3 series for backbone models in our main experiments. By using Qwen3-Instruct models of different sizes, including 8B, 14B, and 32B, we can explore whether memory-enhanced agent performance improves as model size increases. For high-quality experience acquisition, Qwen3 thinking mode is activated for BFCL-V3 tasks and disabled for AppWorld tasks.

To validate the generalizability of our system, we also evaluate other LLMs with and without ReMe on BFCL-V3 tasks, including GPT-4.1-2025-04-14 (OpenAI, 2025a), o4-mini-2025-04-16 (OpenAI, 2025b), Qwen3-Max-Preview (Team, 2025a), Kimi-K2-Thinking (MoonshotAI, 2025), DeepSeek-V3.2 (Liu et al., 2025a), and GLM-4.7 (ZhipuAI, 2025), where Qwen3-Max-Preview, DeepSeek-V3.2, and Kimi-K2-Thinking are in the thinking mode, and the others are in the non-thinking mode.

Trajectory-level Experience

when to use: When a user needs to assess the current market status and make informed trading decisions, such as buying or canceling an order.
experience content:
1. Retrieve the current time using `get_current_time`.
2. Use the retrieved time to update and obtain the market status via `update_market_status`.
3. If the market is open and the user decides to trade, use `place_order` to execute the trade.
4. If the user requests cancellation, call `cancel_order` with the appropriate order ID.
5. Provide updates on account details through `get_account_info` if requested by the user.

Keypoint-level Experience

when to use: When a user wants to place an order for a stock but without providing a specific price.
experience content: The assistant demonstrated a methodical approach by first retrieving the current stock price using get_stock_info and then using that price in the place_order function. This two-step process ensures compliance with the required parameters of the place_order function while aligning with the user's intent for a market-price-based order.

Figure 10: Comparison of trajectory-level and keypoint-level experience granularity.

Model	No Memory	ReMe (fixed)	ReMe (dynamic)
GPT-4.1	48.25%	53.33% $_{+5.08\%}$	54.67% $_{+6.42\%}$
o4-mini	54.67%	57.78% $_{+3.11\%}$	60.00% $_{+5.33\%}$
Qwen3-Max	59.00%	61.83% $_{+2.83\%}$	64.00% $_{+5.00\%}$
Kimi-K2-Thinking	57.17%	62.17% $_{+5.00\%}$	66.00% $_{+8.83\%}$
DeepSeek-V3.2	53.06%	56.00% $_{+2.94\%}$	57.33% $_{+4.27\%}$
GLM-4.7	68.00%	70.00% $_{+2.00\%}$	73.83% $_{+5.83\%}$

Table 5: Performance comparison of more LLMs with and without ReMe on BFCL-V3 (Avg@4 %).

B.4 Implementation Details

B.4.1 For Experience Acquisition

First, we sample trajectories N= 8 times for each task query to obtain a diverse set of potential solutions including both high-reward and low-reward results. Next, within each group corresponding to the same task, all trajectories are sorted by their rewards and only the lowest-scoring and highest-scoring examples are selected to the following experience acquisition.

- **Success Pattern Recognition**: Successful trajectories are defined as those exceeding a predefined score threshold (empirically set to 1.0). Then, we prompt LLM_{summ} to identify the key point that contributes to the task success.
- **Failure Analysis**: Conversely, failed trajectories trigger failure analysis by prompting LLM_{summ} to determine the earliest key step that leads to suboptimal outcomes.
- **Comparative Insight Generation**: When the reward gap exists between the chosen two trajectories, we prompt LLM_{summ} to articulate which specific decision or action distinguishes higher-scoring from lower-scoring attempts.

Three example prompts for experience acquisition are shown in Table 7, 8 and 9. To filter out the generated invalid experiences, we employ the LLM-as-a-Judge prompt in Table 10 for validation.

B.4.2 For Experience Retrieval

When a new task is received, $\text{LLM}_{execute}$ retrieves relevant experiences $\mathcal{E}_r$ by matching the current task's query q_{new} against the usage scenario field w of stored experiences:

$$\mathcal{E}_r = \arg\text{top}_k \left[sim_{cos}(E_i, q_{new}) \right]. \quad (2)$$

Here, sim_{cos} stands for the computation of cosine similarity between embeddings. In our experiments, past experiences are indexed using vector representations of the usage scenario field $\phi(w)$, obtained from Qwen3-Embedding model $\phi(\cdot)$. Our selected vector database is Elasticsearch.

$$sim_{cos}(E, q_{new}) = \frac{\phi(w) \cdot \phi(q_{new})}{\|\phi(w)\| \, \|\phi(q_{new})\|} \quad (3)$$

In Section 4.3, we also explore more indexing strategies for experience storage. The example in Figure 7 illustrates the differences among these retrieval keys.

C Experience Examples

ReMe focuses on extracting keypoint-level experiences from historical trajectories, with examples for BFCL-V3 and AppWorld illustrated in Figure 8 and 9, respectively. To further investigate the impact of experience granularity, we compare trajectory-level and keypoint-level acquisition, as described in Section 4.3. In Figure 10, we contrast the structural and content characteristics of the two granularity levels, showing how trajectory-level captures exhaustive procedural details, while keypoint-level emphasizes critical actions and omits less relevant steps.

D Detailed Experimental Results

D.1 Generalizability across Different Models

To examine the generalizability of ReMe, we conduct experiments with different backbone models, including GPT-4.1, o4-mini, Qwen3-Max, Kimi-K2, DeepSeek-V3.2, and GLM-4.7. The

Model	Retrieval Key	Avg@4	Pass@4
Qwen3-8B	*task query*	44.00%	63.33%
	generalized query	42.50%	63.77%
	query keywords	44.22%	65.33%
	usage scenario	**44.50%**	**65.77%**
Qwen3-14B	*task query*	50.11%	71.77%
	generalized query	50.49%	72.22%
	query keywords	51.16%	71.11%
	usage scenario	**51.89%**	**72.44%**
Qwen3-32B	*task query*	56.22%	72.22%
	generalized query	55.33%	73.33%
	query keywords	**56.89%**	74.44%
	usage scenario	56.05%	**74.89%**

Table 6: Ablation study of retrieve keys.

stacked analysis in Figure 3 and detailed results in Table 5 show that `ReMe` consistently outperforms No Memory baseline across various backbones, confirming the model-agnostic advantage of our novel procedural memory system.

D.2 Retrieval Key Analysis

Table 6 compares four retrieval key strategies (task query, generalized query, query keywords, and usage scenario) across three model scales (Qwen3–8B, Qwen3–14B, and Qwen3–32B) on the BFCL-V3 benchmark under the `ReMe(fixed)` setting. Consistent with the trends observed in Figure 4, simple indexing methods such as raw task query and query keywords generally yield lower performance. In contrast, LLM-generated retrieval keys, particularly the usage scenario field, exhibit consistently strong results across all models, achieving the highest or near-highest Avg@4 and Pass@4 scores.

E Case Study

To gain deeper insights into how experience reuse influences agent reasoning, we compare two agent trajectories on the same BFCL-V3 task, one guided by retrieved experiences and one without. As illustrated in Figure 1, without past experience, the agent encounters a failure when purchasing Apple shares since it fabricates the current market price instead of fetching real-time data. With `ReMe`, past experience guides the agent to correctly obtain real-time pricing before placing an order, successfully completing the stock trading task. This case demonstrates how experience-driven reasoning prevents agents from repeating earlier mistakes and improves robustness across similar scenarios.

Example Prompt for Success Pattern Recognition

You are an expert AI analyst reviewing successful step sequences from an AI agent execution.

Your task is to extract reusable, actionable step-level experiences that can guide future agent executions.
Focus on identifying specific patterns, techniques, and decision points that contributed to success.

ANALYSIS FRAMEWORK:
• STEP PATTERN ANALYSIS: Identify the specific sequence of actions that led to success
• DECISION POINTS: Highlight critical decisions made during these steps
• TECHNIQUE EFFECTIVENESS: Analyze why specific approaches worked well
• REUSABILITY: Extract patterns that can be applied to similar scenarios

EXTRACTION PRINCIPLES:
• Focus on TRANSFERABLE TECHNIQUES and decision frameworks
• Frame insights as actionable guidelines and best practices

Original Query
{query}

Step Sequence Analysis
{step_sequence}

Context Information
{context}

Outcome
This step sequence was part of a successful trajectory.

OUTPUT FORMAT:
Generate 1-3 step-level success insights as JSON objects:
```json
[
  {{
    “when_to_use” : “Specific conditions when this success insight should be applied”,
    “task_query” : “Identify the specific task query from the original trajectory that this success experience is most closely related to. Extract the exact query text.”,
    “generalized_query” : “Abstract the specific task query to create a more generalized task representation.”,
    “experience” : “Detailed description of the successful step pattern and why it works”,
    “tags” : [“relevant", “keywords", “from", “the", “task", “query"],
    “confidence” : 0.8,
    “tools_used” : [“list", “of", “tools"]
  }}
]
```

Table 7: Example prompt for success pattern recognition.

Example Prompt for Failure Analysis

You are an expert AI analyst reviewing failed step sequences from an AI agent execution.

Your task is to extract learning experiences from failures to prevent similar mistakes in future executions.
Focus on identifying error patterns, missed opportunities, and alternative approaches.

ANALYSIS FRAMEWORK:
• FAILURE POINT IDENTIFICATION: Pinpoint where and why the steps went wrong
• ERROR PATTERN ANALYSIS: Identify recurring mistakes or problematic approaches
• ALTERNATIVE APPROACHES: Suggest what could have been done differently
• PREVENTION STRATEGIES: Extract actionable insights to avoid similar failures

EXTRACTION PRINCIPLES:
• Extract GENERAL PRINCIPLES as well as SPECIFIC INSTRUCTIONS
• Focus on PATTERNS and RULES as well as particular instances

Original Query
{query}

Step Sequence Analysis
{step_sequence}

Context Information
{context}

Outcome
This step sequence was part of a failed trajectory.

OUTPUT FORMAT:
Generate 1-3 step-level failure prevention insights as JSON objects:
```json
[
  {{
    “when_to_use” : “Specific situations where this lesson should be remembered”,
    “task_query” : “Identify the specific task query from the original trajectory that this lesson is most closely related to. Extract the exact query text.”,
    “generalized_query” : “Abstract the specific task query to create a more generalized task representation.”,
    “experience” : “Universal principle or rule extracted from the failure pattern”,
    “tags” : [“relevant", “keywords", “from", “the", “task", “query"],
    “confidence” : 0.8,
    “tools_used” : [“list", “of", “tools"]
  }}
]
```

Table 8: Example prompt for failure analysis.

Example Prompt for Comparative Insights Generation

You are an expert AI analyst comparing higher-scoring and lower-scoring step sequences to extract performance insights.

Your task is to identify the key differences between higher and lower performing approaches at the step level.
Focus on what made the higher-scoring approach more effective, even when both approaches may have had partial success.

SOFT COMPARATIVE ANALYSIS FRAMEWORK:
• PERFORMANCE FACTORS: Identify what specifically contributed to the higher score
• APPROACH DIFFERENCES: Compare methodologies and execution strategies
• EFFICIENCY ANALYSIS: Analyze why one approach was more efficient or effective
• OPTIMIZATION INSIGHTS: Extract lessons for improving performance

EXTRACTION PRINCIPLES:
• Focus on INCREMENTAL IMPROVEMENTS and performance optimization
• Extract QUALITY INDICATORS that differentiate better vs good approaches
• Identify REFINEMENT STRATEGIES that lead to higher scores
• Frame insights as PERFORMANCE ENHANCEMENT guidelines

Higher-Scoring Step Sequence (Score: {higher_score})
{higher_steps}

Lower-Scoring Step Sequence (Score: {lower_score})
{lower_steps}

OUTPUT FORMAT:
Generate 1-2 performance improvement insights as JSON objects:
```json
[
  {{
    “when_to_use” : “Specific scenarios where this performance insight applies”,
    “task_query” : “Identify the specific task query from the original trajectory that this performance insight is most closely related to. Extract the exact query text.”,
    “generalized_query” : “Abstract the specific task query to create a more generalized task representation.”,
    “experience” : “Detailed analysis of what made the higher-scoring approach more effective”,
    “tags” : [“relevant", “keywords", “from", “the", “task", “query"],
    “confidence” : 0.8,
    “tools_used” : [“list", “of", “tools"]
  }}
]
```

Table 9: Example prompt for comparative insights generation.

Example Prompt for Experience Validation

You are an expert AI analyst tasked with validating the quality and usefulness of extracted step-level experiences.

Your task is to assess whether the extracted experience is actionable, accurate, and valuable for future agent executions.

VALIDATION CRITERIA:
• ACTIONABILITY: Is the experience specific enough to guide future actions?
• ACCURACY: Does the experience correctly reflect the patterns observed?
• RELEVANCE: Is the experience applicable to similar future scenarios?
• CLARITY: Is the experience clearly articulated and understandable?
• UNIQUENESS: Does the experience provide novel insights or common knowledge?

Experience to Validate
Condition: condition
Experience Content: experience_content

OUTPUT FORMAT:
Provide validation assessment:
```json
{{
  "is_valid" : true/false,
  "score" : 0.8,
  "feedback" : "Detailed explanation of validation decision",
  "recommendations" : "Suggestions for improvement if applicable"
}}
```
Score should be between 0.0 (poor quality) and 1.0 (excellent quality).
Mark as invalid if score is below 0.3 or if there are fundamental issues with the experience.

Table 10: Example prompt for experience validation.