RefCon: Iterative Refinement and Contrastive Memory Extraction for Context-Evolving Agent

Anonymous ACL submission

Abstract

Long-horizon agent interactions generate useful but noisy experience, and retraining models to absorb it is expensive. Context-evolving agents therefore need memory extraction methods that improve with more test-time compute without relying on gold labels. We study memory-aware test-time scaling for context-evolving agents and propose RefCon, which combines sequential self-refinement with parallel self-contrast to extract higher-quality memories without gold labels. Evaluated on AppWorld and BFCL-V3 within multiple context-evolving agent frameworks, RefCon delivers strong and consistent gains, including relative improvements of 21.6% on ACE and 16.6% on ReMe over no-scaling baselines, while a diversity-focused variant (DivCon) achieves a 35.5% relative gain on ReasoningBank. In the without-ground-truth setting, RefCon consistently outperforms existing context-evolving agent baselines, showing that its extraction pipeline remains effective when labels are unavailable. We further analyze the accuracy-token trade-off and scaling behavior, showing that RefCon maintains favorable efficiency while continuing to improve as more trajectories are used, unlike diversity-only scaling which saturates earlier. These results demonstrate that integrating refinement and contrast is a practical, compute-efficient path to more robust memory extraction and continual agent improvement.

1 Introduction

A few years ago, large language models (LLMs) began to be deployed as agents capable of solving complex tasks through iterative reasoning and tool use. Applications such as web browsing (Wei et al., 2025), interactive tool calling (Patil et al., 2025), and deep research (Phan et al., 2025) require agents to generate long trajectories of actions and observations. However, these trajectories are typically discarded after task completion, despite often containing reusable insights that could benefit future tasks (Shinn et al., 2023; Zhao et al., 2024). While one possible solution is to incorporate such knowledge through model retraining, updating large language models remains computationally expensive due to their growing scale (Hoffmann et al., 2022). This motivates lightweight mechanisms that allow agents to accumulate and reuse experience directly from past interactions without costly retraining.

More recently, self-evolving agent approaches integrate knowledge updates directly into the inference loop. In this paradigm, agents continuously write, refine, and retrieve memories to compose an up-to-date context for each new query, enabling continual learning without expensive model retraining (Park et al., 2023; Wang et al., 2024). These context-evolving agents typically involve three components: memory extraction, memory retrieval, and memory management (Zhang et al., 2025b; Cao et al., 2025; Yan et al., 2025; Zhang et al., 2025a). Memory extraction distills raw trajectories into reusable heuristics; memory retrieval selects relevant experiences for the current task; and memory management maintains a compact memory store by filtering redundancy and irrelevant content. Recent work further explores memory-aware test-time scaling (MaTTS) to improve memory extraction by generating multiple trajectories and selecting higher-quality experiences (Ouyang et al., 2025). This direction builds on broader findings that test-time scaling can improve agent performance (Snell et al., 2025; Zhu et al., 2025).

Despite recent progress, several challenges remain. Some methods rely on gold labels or ground-truth signals to extract memories (Cao et al., 2025; Zhang et al., 2025b; Cai et al., 2025b), which are often unavailable in real-world settings. Others replace these signals with LLM-as-a-Judge (Zheng et al., 2023), but such evaluations can be unreliable for long-horizon interactions where trajectories ap-

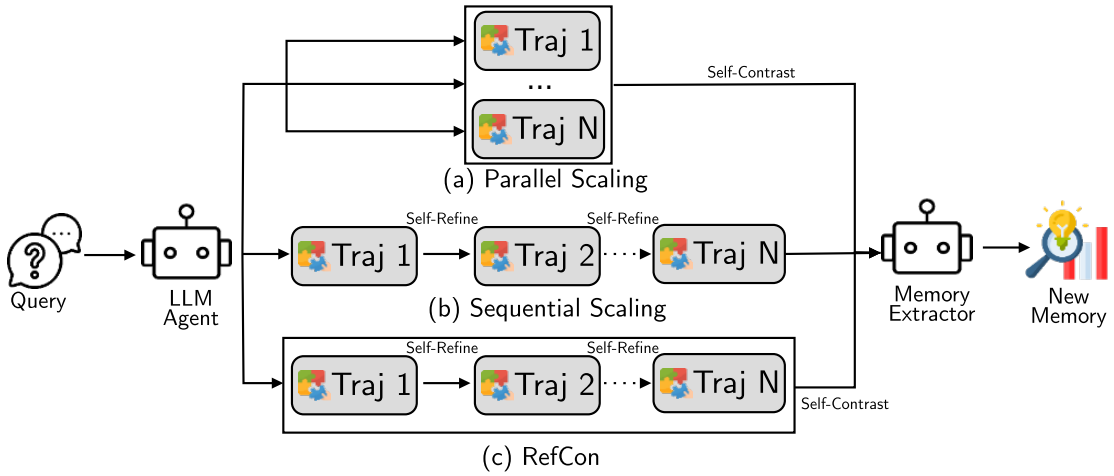

Figure 1: Memory extraction with (a) parallel scaling using self-contrast, and (b) sequential scaling with self-refine, compared to our proposed method (c) RefCon, combining both strength of parallel and sequential scaling.

pear plausible yet fail to achieve the intended goal (Ouyang et al., 2025; Yang et al., 2025; Liu et al., 2025). ReasoningBank (Ouyang et al., 2025) introduces self-contrast reasoning over multiple trajectories, leveraging parallel scaling to identify successful reasoning patterns without relying on gold labels, as shown in Figure 1 (a) and also introduces sequential scaling by self-refining previous trajectory as shown in Figure 1 (b). However, while prior work explores both parallel scaling (Stein et al., 2025; Wu et al., 2025; Wang et al., 2025a) and sequential scaling (Yan et al., 2025; Tang et al., 2025), they are typically studied in isolation. In practice, the two strategies are complementary: parallel scaling improves trajectory diversity, while sequential scaling enhances trajectory quality. Existing work has yet to fully combine these strengths within a unified framework.

To address these limitations, we propose RefCon, an extension of memory-aware test-time scaling (MaTTS) for memory extraction that combines the benefits of sequential and parallel scaling. RefCon first applies sequential scaling to generate N trajectories using self-refine (Madaan et al., 2023), as illustrated in Figure 1. These trajectories are then jointly analyzed through parallel scaling, where the memory extractor performs self-contrast reasoning (Chen et al., 2020) to identify improvements across trajectories and distill reusable insights. This design encourages both trajectory diversity and quality, enabling more reliable memory extraction. We further introduce a variant, DivCon, which promotes diversity by instructing the agent to explore alternative strategies instead of refining previous trajectories. For memory retrieval, we adopt a retrieve–rerank–rewrite pipeline that improves relevance by reranking retrieved memories and rewriting them into a concise form. Overall, our contributions are twofold:

- We propose **RefCon**, a memory-aware test-time scaling framework that unifies parallel and sequential scaling to improve memory extraction quality. We will release it as a plug-in for agentic systems to support context-evolving methods, enabling easier adoption and future research.

- Extensive experiments demonstrate that RefCon consistently outperforms existing baselines in the without-ground-truth setting. Compared with the no-scaling setting, RefCon improves performance by 21.6% on ACE and 16.6% on ReMe, while the variant DivCon achieves a 35.5% gain on ReasoningBank.

2 Related Works

Memory Extraction. Modern memory extraction has transitioned from passive logging to active insight distillation. ACE (Zhang et al., 2025b) decouples reflection from curation to synthesize small memory updates which will be integrated by curator. ReasoningBank (Ouyang et al., 2025) highlights the importance of integrating cautionary insights from failed trajectories. Frameworks like MUSE (Yang et al., 2025) and FLEX (Cai et al., 2025b) employ hierarchical distillation to separate high-level heuristics (principles, strategies) from

execution details (tools, patterns). Beyond distilling general insights, EGuR (Stein et al., 2025) also learns task-specific workflows tailored to each problem, while SAGE (Wang et al., 2025a) and SMITH (Liu et al., 2025) generate reusable code snippet for the next query. Furthermore, ReasoningBank (Ouyang et al., 2025) introduces the fundamental of memory-aware test-time scaling (MaTTS) which utilizes self-contrast reasoning (Chen et al., 2020) in parallel scaling and self-refine (Madaan et al., 2023) in sequential scaling. Since then, many follow-up works have incorporated parallel (Stein et al., 2025; Wu et al., 2025; Cao et al., 2025; Liu et al., 2025; Cai et al., 2025a; Wang et al., 2025a) and sequential (Cao et al., 2025; Yang et al., 2025; Yan et al., 2025; Tang et al., 2025) scaling adapted to fit their own design.

Memory Retrieval. Earlier approaches either inserted all memories into the prompt (Suzgun et al., 2025; Zhang et al., 2025b) or relied on basic embedding-similarity retrieval (Ouyang et al., 2025). Since then, retrieval architectures have evolved into dynamic and hierarchical process. ReMe (Cao et al., 2025) utilizes pipeline of retrieve, rerank, and rewrite to ensure the generated memory is relevant with current query. GAM (Yan et al., 2025), employ "Researcher" agents to perform iterative deep dives into historical data generating context by performing deep research. Instead of static lookups, G-Memory (Zhang et al., 2025a) utilizes graph traversals, while AgentKB (Tang et al., 2025) separates workflow patterns from execution details. EvolveR (Wu et al., 2025) treats memory retrieval as learned tool-call via GRPO. This shift allows models to interpolate strategies for novel problems or retrieve guidance that bridge isolated inference events, ensuring relevant formulas and patterns are available exactly when needed.

Memory Management. To prevent unlimited growing memory, robust management mechanisms now treat memory as a self-optimizing component through pruning and evolution. ACE (Zhang et al., 2025b) and FLEX (Cai et al., 2025b) share the same idea on delegating memory management responsibility to a single agent capable of insert, update, and delete the memory database. ReMe (Cao et al., 2025) and EvolveR (Wu et al., 2025) utilizes scoring based on ratio the number of time a memory is helpful when retrieved, and remove memories with score below threshold. Despite its importance, explicit memory management is often omitted in current systems.

3 Methodology

3.1 Problem Formulation

In this work, we focus on the test-time learning (TTL) paradigm (Wu et al., 2024; Wang et al., 2025b), where a model continues to improve its performance during inference without access to ground-truth supervision. Specifically, we consider a setting in which an agent must adapt to a continuous stream of task queries $Q = \{q_1, q_2, \ldots, q_N\}$ encountered sequentially in an online environment. Unlike traditional supervised learning, this setting precludes access to future queries or ground-truth labels. Instead, the agent must **self-evolve** by critically analyzing its past trajectories and leveraging self-verification mechanisms. The primary objective is to enable the system to extract and retain useful insights from prior interactions, thereby avoiding redundant rediscovery of successful strategies or the repetition of past failures.

Specifically, we instantiate an LLM-based agent $\pi_{\mathcal{L}}$ that solves each query through iterative tool use and reasoning in a ReAct loop, while reusing accumulated experience across tasks. To model this cross-task learning process, we define a memory state initialized as $\mathcal{M}_0 = \emptyset$. At task i, the agent conditions on the current memory $\mathcal{M}_i$ to produce a trajectory for the query. After the task is completed, the trajectory is distilled into memory updates, yielding the next state $\mathcal{M}_{i+1}$. Thus, memory evolves at the task level as $\mathcal{M}_0, \mathcal{M}_1, \ldots$, and the agent for task i is parameterized by $\pi_{\mathcal{L}}(\cdot; \mathcal{M}_i, \mathcal{A})$, where $\mathcal{A}$ is the available action/tool space. In practice, $\mathcal{M}_i$ is injected into the prompt after the task instruction so that decisions for the current task are guided by insights from prior tasks.

3.2 RefCon for Memory Extraction

We propose an iterative **ref**inement and **con**trastive (RefCon) framework, which generates trajectories for our memory extraction process as shown in Figure 2 green zone. The memory extraction process within the test-time learning paradigm is formulated as an iterative refinement procedure that generates a sequence of trajectories. For a given task query q_i, the agent first produces an initial trajectory by conditioning its policy $\pi_{\mathcal{L}}$ on the rewritten memory m_i, which is retrieved from the current memory state $\mathcal{M}_i$:

$$\tau_{i,1} = \pi_{\mathcal{L}}(q_i, m_i). \quad (1)$$

Subsequently, the agent iteratively refines its

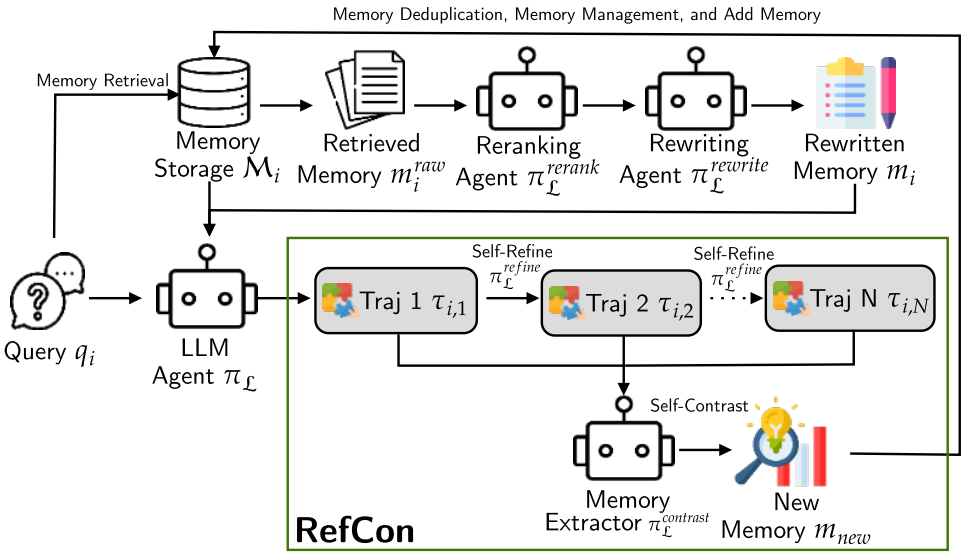

Figure 2: Overview of context-evolving agent workflow utilizing RefCon scaling for memory extraction.

previous trajectory to generate improved problem-solving attempts:

$$\tau_{i,n} = \pi_{\mathcal{L}}^{\text{refine}}(q_i, m_i, \tau_{i,n-1}), \quad n \in \{2, \ldots, N\}, \tag{2}$$

where $\pi_{\mathcal{L}}^{\text{refine}}$ represents the self-refinement policy that evaluates and improves upon the preceding trajectory. The resulting trajectory set $\{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,N}\}$ captures a progression of reasoning attempts, evolving from initial exploration to increasingly refined and self-corrected strategies.

Once the N trajectories are generated, a self-contrast reasoning mechanism is applied to distill high-value insights for memory extraction. Specifically, the agent's memory extractor compares the differences between *successful* and *unsuccessful* steps across the N trajectories, identifying key turning points or reasoning patterns that lead to successful outcomes. Through this contrastive analysis, the agent summarizes these insights into a set of candidate memories m_{new}, formally defined as

$$m_{new} = \pi_{\mathcal{L}}^{\text{contrast}}(\tau_{i,1}, \ldots, \tau_{i,N}), \tag{3}$$

where $\pi_{\mathcal{L}}^{\text{contrast}}(\cdot)$ denotes the contrastive memory extraction operator that distills actionable knowledge from multiple trajectories.

Before finalizing the knowledge state $\mathcal{M}_{i+1}$, the system performs a memory deduplication step using embedding-based similarity (Sim) with a threshold ϵ to remove redundant memories. Specifically, the candidate memories m_{new} are compared against both the existing memory $\mathcal{M}_i$ (inter-memory comparison) and among themselves (intra-memory comparison). Only non-redundant memories are retained, ensuring that memory remains concise and informative. The update rule is defined as:

$$\mathcal{M}_{i+1} = \mathcal{M}_i \cup \{m \in m_{\text{new}} \mid \text{Sim}(m, \mathcal{M}_i \cup m_{\text{new}} \setminus \{m\}) < \epsilon\}. \tag{4}$$

The resulting memory $\mathcal{M}_{i+1}$ now contains memories from all previous $i+1$ task. Each memory entry contains two key components: (i) *when to use*, a short explanation of when to use the memory, also used as an index; (ii) *content*, describing the insight from previous trajectory to guide agent in similar situation. These memories also contains other metadata such as frequency and utility.

DivCon. We also propose an alternative trajectory generation strategy named **DivCon**. Unlike self-refine in RefCon, DivCon uses self-diversity to enhance memory extraction through self-contrasting highly diverse trajectories. The difference with RefCon relies on Eq. 2, where DivCon prompts the agent to explore other solutions:

$$\tau_{i,n} = \pi_{\mathcal{L}}^{\text{diversity}}(q_i, m_i, \tau_{i,n-1}). \tag{5}$$

3.3 Memory Retrieval and Management

The retrieval process serves as the initial phase of the test-time learning cycle, augmenting the agent's policy $\pi_{\mathcal{L}}$ with the most relevant historical insights before solving a new task. We adopt a *retrieve–rerank–rewrite* strategy. Given a query q_i, the system first retrieves candidate memories from

the current memory state $\mathcal{M}_i$ using embedding similarity. A reranking agent then evaluates these candidates and selects the most relevant insights according to the policy's internal ranking logic. Finally, a rewriting agent consolidates the reranked memories to produce a concise and query-aligned memory:

$$m_i = \pi_{\mathcal{L}}^{\text{rewrite}}(\pi_{\mathcal{L}}^{\text{rerank}}(\text{Sim}(\mathcal{M}_i, q_i), q_i), q_i). \quad (6)$$

The resulting memory m_i summarizes the most relevant prior experience and is used to guide the agent when solving the current task. To maintain a high-quality knowledge base, we implement a utility-based deletion strategy to prune ineffective memories. This ensures that the agent's behavior is shaped only by high-utility memories. We define the utility of an experience $E \in \mathcal{M}$ based on two metrics: the total number of retrievals $f(E)$ and the historical utility $u(E)$, which increments by 1 whenever its recall contributes to a successful task completion. An experience is flagged for removal if its average utility falls below a predefined threshold β, provided it has been retrieved at least α times.Using the established notation, the deletion function ϕ_{remove} is formulated as:

$$\phi_{remove}(E) = \begin{cases} 1\left[\frac{u(E)}{f(E)} \leq \beta\right], & \text{if } f(E) \geq \alpha, \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

4 Experiments

4.1 Setup

Datasets. We use two datasets that emphasize multi-step and multi-turn agent interaction: AppWorld and BFCL-V3. AppWorld (Trivedi et al., 2024) contains app-like tasks that require tool use across multiple steps, while BFCL-V3 (Patil et al., 2025) focuses on tool-calling and decision sequences. We use the AppWorld dev split and sample 50 tasks from the BFCL-V3 multi-turn travel subset, which is the most challenging part. We report Avg@3 and Pass@3 on BFCL-V3, and Avg@3 and Pass@3 for each Task Goal Completion (TGC) and Scenario Goal Completion (SGC) on AppWorld. We additionally report the standard deviation across three runs for Avg@3.

Models. We use GLM-4.6 (Z.ai, 2024) as the backbone LLM for agent reasoning, tool use, and memory extraction, and OpenAI text-embedding-3-small (OpenAI, 2024) for retrieval indexing and similarity search. GLM-4.6 provides strong instruction following and multi-step reasoning, while text-embedding-3-small offers high-quality semantic representations for effective memory retrieval. We also experiment using Qwen3.5 9B (Team, 2026) for generalization in smaller model.

Baseline Methods. We use ReAct framework as base and utilize ACE, ReasoningBank, and ReMe as baseline context-evolving agents. ACE (Zhang et al., 2025b) extracts and curates memories through a reflector–curator agent pipeline but directly uses all stored memories without a dedicated retrieval mechanism. ReasoningBank (Ouyang et al., 2025) introduces the MaTTS paradigm, which performs memory extraction from both successful and failed trajectories and retrieves memories using embedding-based similarity. ReMe (Cao et al., 2025) further improves retrieval by incorporating a retrieve–rerank–rewrite pipeline to filter and refine retrieved memories. These methods also differ in their test-time scaling strategies, including parallel and sequential trajectory refinement. More details are in Appendix B.

Hyperparameters. We generally employ a scaling factor of 3 (except 2 for DivCon). To promote trajectory diversity, we use parallel rollouts with varying temperatures (Holtzman et al., 2020; Wang et al., 2023). We follow the original implementations for results in Table 1, but for Table 2, we disable ground-truth usage and integrate MaTTS for memory extraction. For RefCon and ReMe, we utilize a top-k retrieval and reranking strategy with specific usage and utility pruning thresholds. Detailed parameter settings, including exact temperature values, retrieval counts, and pruning formulas (ϵ, α, β), are provided in Appendix D. Prompt templates are detailed in Appendix E.

4.2 Main Comparison

Table 1 shows that while ground-truth methods remain strong due to they can extract memories that closely match the target signals. Notably, RefCon achieves the best overall performance even in the absence of ground-truth labels, surpassing some ground-truth baselines (e.g., ReMe Parallel) and nearly matching the top scores on AppWorld. Relative to ReAct, RefCon improves AppWorld Avg@3 by 16.51% (TGC) and 50.06% (SGC), and and BFCL-V3 Pass@3 by 32.36%. Compared to the existing baselines in without-ground-truth setting, RefCon achieves superior performance. Against ReasoningBank (sequential), RefCon im-

Table 1: Performance of context-evolving agent methods on AppWorld and BFCL-V3 Using GLM-4.6.

Methods	AppWorld				BFCL-V3	
	Avg@3		Pass@3		Avg@3	Pass@3
	TGC	SGC	TGC	SGC		
Baseline						
ReAct	$63.77^{\pm 4.60}$	$35.10^{\pm 10.80}$	87.72	68.42	$52.67^{\pm 4.11}$	62.00
With Gold Labels/Ground-Truth						
ACE (with Ground-Truth)	$\mathbf{78.00^{\pm 5.34}}$	$50.9^{\pm 8.96}$	**96.49**	**89.47**	$66.67^{\pm 4.71}$	**82.00**
ReMe (Sequential Scaling)	$75.43^{\pm 10.81}$	$\mathbf{57.93^{\pm 14.90}}$	**96.49**	**89.47**	$\mathbf{67.33^{\pm 0.94}}$	**82.00**
ReMe (Parallel Scaling)	$62.00^{\pm 0.85}$	$36.80^{\pm 0.00}$	84.21	68.42	$58.67^{\pm 1.89}$	68.00
Without Gold Labels/Ground-Truth						
ACE (without Ground-Truth)	$59.63^{\pm 2.9}$	$42.10^{\pm 4.33}$	77.19	73.68	$58.00^{\pm 5.89}$	80.00
ReasoningBank	$56.13^{\pm 1.43}$	$42.10^{\pm 4.33}$	80.70	68.42	$60.00^{\pm 7.12}$	**82.00**
ReasoningBank (Sequential Scaling)	$69.03^{\pm 1.65}$	$43.87^{\pm 2.50}$	84.21	73.68	$64.00^{\pm 1.63}$	74.00
ReasoningBank (Parallel Scaling)	$67.27^{\pm 2.20}$	$45.63^{\pm 10.81}$	85.96	**78.95**	$\mathbf{65.33^{\pm 4.11}}$	72.00
DivCon	$71.93^{\pm 5.76}$	$47.37^{\pm 4.29}$	**92.98**	**78.95**	$58.67^{\pm 3.40}$	76.00
RefCon	$\mathbf{74.30^{\pm 0.94}}$	$\mathbf{52.67^{\pm 7.45}}$	**92.98**	**78.95**	$63.33^{\pm 0.94}$	**82.00**

proves AppWorld Avg@3 by up to 20.06% and BFCL-V3 Pass@3 by 13.89%. However, it trails ReasoningBank (Parallel) in BFCL-V3 Avg@3 by 3.06%, likely because BFCL-V3's narrow distribution favors contrastive diversity over refinement. Our other proposed method, DivCon consistently ranks second in the without ground-truth setting, except on BFCL-V3 benchmark. Overall, RefCon provides the most consistent gains without labels, while DivCon offers a competitive, diversity-driven alternative. Findings for the smaller Qwen3.5 9B model generally align with those of GLM-4.6, with some minor difference. We present the results for these experiments in Table 3 in Appenddix A.

Several anomalies are worth noting. RefCon, DivCon, and ReasoningBank (Sequential Scaling) underperform on BFCL-V3 because self-refinement is less reliable in this benchmark: the refinement step often fails to correct earlier mistakes and can even degrade a trajectory, whereas ReMe's sequential scaling retries only failed trajectories. This suggests RefCon's gains depend on effective self-refinement. On AppWorld, ACE and ReasoningBank without scaling fall below the ReAct baseline, indicating that MaTTS is critical; without labels, extracting memories from a single trajectory can introduce noisy or misleading memories that reinforce incorrect behavior. Finally, ReMe's parallel scaling is weak even with ground-truth because its pipeline does not fully leverage self-contrast: it extracts memories from best and worst trajectories separately and only contrasts them when their scores differ, so the contrastive signal is often absent. Additional analysis on how RefCon improve performance is provided in Appendix C.

4.3 RefCon's Combined Scaling Advantage

In this subsection, we only adopt the memory format and memory retrieval strategy from each method. However, we modify the memory extraction part to adopt parallel and sequential scaling as introduced by (Ouyang et al., 2025) and also RefCon and DivCon as introduced by us. We don't use any gold-label or ground-truth to ensure fair comparison in this section. We compare the effect of MaTTS methods for each context-evolving agent methods as shown in Table 2 and provide detailed results for each run in Appendix A.

Across agents, RefCon is the strongest variant on ACE (21.6% relative improvement over no scaling) and ReMe (16.6%), while DivCon is the top performer on ReasoningBank (35.5%). The runner-up alternates between RefCon and DivCon, suggesting their strengths are complementary: RefCon benefits more when iterative refinement yields progressively better trajectories, whereas DivCon shines when diversity across trajectories exposes contrasting reasoning patterns. This pattern highlights that MaTTS is not tied to a single memory system but generalizes across context-evolving agents. In particular, ACE originally lacks any scaling mechanism, yet adding MaTTS variants substantially improves performance without changing its reflector/curator pipeline; the only change is supplying multiple trajectories to the reflector and curator. This implies the gains come from richer trajectory evidence rather than architectural changes, reinforc-

Table 2: Comparison of RefCon and DivCon against MaTTS methods for context-evolving agents on AppWorld.

Methods	**AppWorld**			
	Avg@3		Pass@3	
	TGC	SGC	TGC	SGC
ACE				
ACE (Without Scaling)	$59.63^{\pm 2.9}$	$42.10^{\pm 4.33}$	77.19	73.68
ACE (Parallel Scaling)	$63.73^{\pm 9.55}$	$43.87^{\pm 13.83}$	84.21	**78.95**
ACE (Sequential Scaling)	$64.30^{\pm 1.55}$	$42.10^{\pm 7.45}$	84.21	**78.95**
ACE (DivCon)	$69.00^{\pm 2.16}$	$42.10^{\pm 4.33}$	91.23	**78.95**
ACE (RefCon)	$\mathbf{72.53^{\pm 5.82}}$	$\mathbf{43.87^{\pm 6.57}}$	**92.98**	**78.95**
ReasoningBank				
ReasoningBank (Without Scaling)	$56.13^{\pm 1.43}$	$42.10^{\pm 4.33}$	80.70	68.42
ReasoningBank (Parallel Scaling)	$67.27^{\pm 2.20}$	$45.63^{\pm 10.81}$	85.96	78.95
ReasoningBank (Sequential Scaling)	$69.03^{\pm 1.65}$	$43.87^{\pm 2.50}$	84.21	73.68
ReasoningBank (DivCon)	$\mathbf{76.03^{\pm 2.21}}$	$\mathbf{52.67^{\pm 0.00}}$	**94.74**	**89.47**
ReasoningBank (RefCon)	$71.93^{\pm 3.80}$	$49.13^{\pm 13.13}$	92.98	84.21
ReMe				
ReMe (Without Scaling)	$63.73^{\pm 3.30}$	$43.87^{\pm 2.50}$	85.96	73.68
ReMe (Parallel Scaling)	$70.76^{\pm 3.57}$	$42.10^{\pm 4.33}$	91.23	84.21
ReMe (Sequential Scaling)	$69.67^{\pm 3.30}$	$47.37^{\pm 4.29}$	91.23	**78.95**
ReMe (DivCon)	$71.93^{\pm 5.76}$	$47.37^{\pm 4.29}$	**92.98**	**78.95**
ReMe (RefCon)	$\mathbf{74.30^{\pm 0.94}}$	$\mathbf{52.67^{\pm 7.45}}$	**92.98**	**78.95**

ing that scaling at test time is a broadly effective lever for memory quality.

DivCon remains competitive with a smaller scaling factor (2), which suggests contrastive memory extraction benefits most when trajectories are genuinely diverse. RefCon leverages a similar contrastive idea, but its diversity comes from incremental self-refinement rather than explicit exploration, whereas temperature-based parallel sampling can be too homogeneous when the model strongly prefers a single solution. DivCon performs particularly well with ReasoningBank because ReasoningBank's memory format is more high level compared to ACE and ReMe. This encourages DivCon to search for other different trajectory while still utilizing the related memory and not contradicting it. We still prioritize RefCon in the main results because DivCon is less stable and exhibits weaker scaling behavior, as will be discussed in the scaling law analysis.

4.4 RefCon Accuracy-Token Tradeoff

As shown in Figure 3, RefCon consistently achieves the highest accuracy across all methods: 72.53% on ACE, 71.93% on ReasoningBank, and 74.30% on ReMe, while maintaining a minimal token overhead of only 0.17M to 1.18M $\leq 3\%$ over sequential scaling to deliver gains up to +8.2%. In Parallel and RefCon settings, ACE's higher total token consumption (4.5-4.74M) relative to ReasoningBank and ReMe (2.5-3.8M) stems from its two-stage reflection and curation process, which effectively doubles extraction overhead per task. This architecture compounds costs by processing 2×3 trajectories worth of new tokens. This is further increased by ACE's practice of injecting all retrieved memories into the ReAct loop, also increasing the cached tokens.

The core insight behind RefCon's efficiency lies in where its token costs are concentrated. Trajectory generation which is the dominant phase by volume is structurally redundant across turns in a ReAct loop, enabling aggressive prompt caching that keeps new token costs low throughout the generation phase. The additional cost RefCon incurs over Sequential is therefore almost entirely localized to the extraction step, which processes three full trajectories of fresh, non-overlapping content and yields predominantly cache misses. This architectural separation is what makes RefCon's trade-off so favorable: the expensive part of the pipeline, trajectory generation via self-refinement, is no more costly than sequential scaling, while the one-time extraction overhead purchases a meaningful jump in memory quality by consolidating the full breadth of all three trajectories. RefCon spends its extra budget where richer input from three refined trajectories improves downstream accuracy.

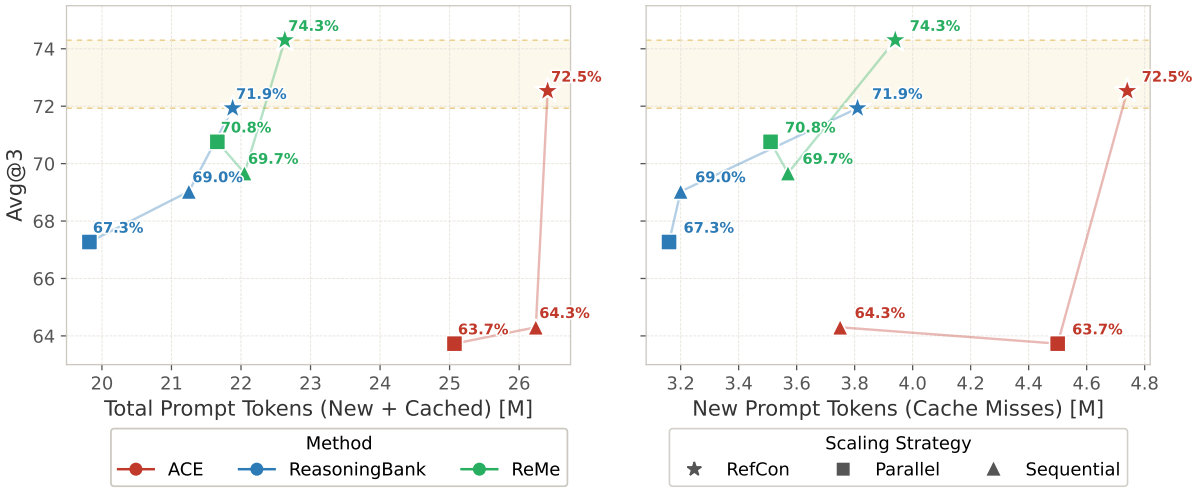

Figure 3: Accuracy (Avg@3) and prompt token tradeoff for each MaTTS methods.

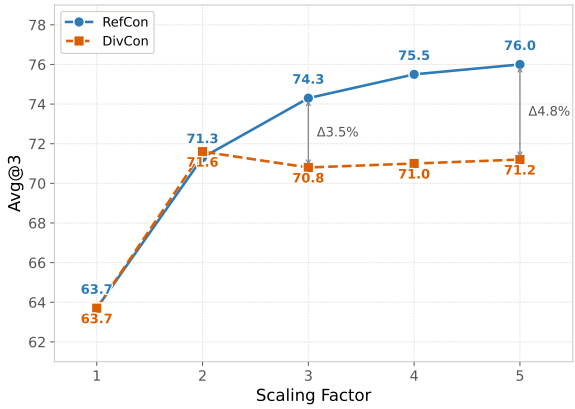

Figure 4: Scaling of RefCon and DivCon.

4.5 Scaling Law

While RefCon and DivCon share a baseline at $k = 1$ (63.7%) and achieve nearly identical performance at $k = 2$ ($\approx 71.5\%$), their scaling trajectories diverge sharply thereafter. As shown in Figure 4, RefCon exhibits steady, albeit decelerating, gains reaching 76.0% at $k = 5$, as self-refinement provides a progressively higher-quality signal for memory consolidation. In contrast, DivCon peaks at $k = 2$ and plateaus around 71.0%, suggesting that additional diversity-seeking trajectories introduce conflicting noise rather than informative signal beyond the initial gain. This indicates that while initial diversity benefits memory extraction, consistent refinement is more effective at converting increased compute into sustained accuracy.

The practical implication is that RefCon is the more scalable strategy under a larger compute budget. While DivCon's early gain at $k = 2$ suggests that a single round of diversity exploration is a useful inductive signal, its inability to leverage further trajectories meaningfully limits its ceiling. RefCon's continued, if slower, improvement from $k = 3$ to $k = 5$ indicates that solution quality rather than solution diversity is the more reliable driver of downstream accuracy as the number of trajectories grows. This makes RefCon particularly well-suited to settings where inference compute can be scaled up, as each additional trajectory reliably improves the memory consolidation input rather than introducing variance that the extractor must filter out.

5 Conclusion

We introduced RefCon, a memory-aware test-time scaling framework that unifies sequential self-refinement with parallel self-contrast to extract higher-quality memories for context-evolving agents without relying on gold labels. Notably, RefCon's label-free extraction surpasses some ground-truth baselines and maintains its effectiveness across different model scales. Across AppWorld and BFCL-V3, RefCon consistently delivers the strongest or near-strongest performance and favorable accuracy-token trade-offs, while our DivCon variant highlights when explicit diversity can help but also reveals stability and scaling limits. These findings emphasize that improving memory quality at test time is a practical, compute-efficient path to continual agent improvement, and they motivate further work on more robust diversity generation, adaptive scaling strategies, and stronger retrieval-extraction coupling for long-horizon tasks.

Limitations

Our study has two main limitations. First, to address potential biases in model scale, we conducted experiments using both GLM-4.6 (large) and Qwen3.5 9B (small). Although incorporating both scales reduces the limitation of a single-model study, the generalizability of our results to models with different architectures or tuning styles is not yet fully guaranteed. We aim to broaden the scope of our evaluations to include more diverse model families in future iterations. Second, our test-time scaling analysis only evaluates up to five trajectories ($k \leq 5$), which means we cannot yet characterize behavior under larger compute budgets, including whether performance continues to improve, plateaus, or degrades beyond this range.

Ethical Considerations

This work does not raise specific ethical concerns. Our contributions focus on developing memory extraction frameworks for effective context-evolving agent. All experiments are conducted on publicly available benchmarks with open-source models, without involving human subjects, sensitive data, or privacy-related information. No potential conflicts of interest are present. While there is a potential for generated memories to occasionally degrade model performance, this occurrence is infrequent and can be mitigated through proper extraction or retrieval thresholds.

References

Yuzheng Cai, Siqi Cai, Yuchen Shi, Zihan Xu, Lichao Chen, Yulei Qin, Xiaoyu Tan, Gang Li, Zongyi Li, Haojia Lin, Yong Mao, Ke Li, and Xing Sun. 2025a. Training-free group relative policy optimization. *Preprint*, arXiv:2510.08191.

Zhicheng Cai, Xinyuan Guo, Yu Pei, Jiangtao Feng, Ya-Qin Zhang, Wei-Ying Ma, Mingxuan Wang, and Hao Zhou. 2025b. Flex: Inheritable intelligence via forward learning from scaling experience. *arXiv preprint*.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. 2025. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *Preprint*, arXiv:2512.10696.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, and 3 others. 2022. An empirical analysis of compute-optimal large language model training. In *Advances in Neural Information Processing Systems*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Jiarun Liu, Shiyue Xu, Yang Li, Shangkun Liu, Yongli Yu, and Peng Cao. 2025. Unifying dynamic tool creation and cross-task experience sharing through cognitive memory architecture. *Preprint*, arXiv:2512.11303.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

OpenAI. 2024. text-embedding-3-small model. OpenAI Developers Documentation. Accessed 2026-02-26.

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. 2025. Reasoningbank: Scaling agent self-evolving with reasoning memory. *Preprint*, arXiv:2509.25140.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *In the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*, UIST '23, New York, NY, USA. Association for Computing Machinery.

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*.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja,

Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, Mantas Mazeika, and 1093 others. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Adam Stein, Matthew Trager, Benjamin Bowman, Michael Kleinman, Aditya Chattopadhyay, Wei Xia, and Stefano Soatto. 2025. Experience-guided adaptation of inference-time reasoning strategies. *Preprint*, arXiv:2511.11519.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2025. Dynamic cheatsheet: Test-time learning with adaptive memory.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, He Zhu, Ge Zhang, Jiaheng Liu, Xingyao Wang, Sirui Hong, Chenglin Wu, and Wangchunshu Zhou. 2025. AGENT KB: A hierarchical memory framework for cross-domain agentic problem solving. In *ICML 2025 Workshop on Collaborative and Federated Agentic Workflows*.

Qwen Team. 2026. Qwen3.5: Accelerating productivity with native multimodal agents.

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, Bangkok, Thailand. Association for Computational Linguistics.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. 2025a. Reinforcement learning for self-improving agent with skill library. *Preprint*, arXiv:2512.17102.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. 2025b. Inducing programmatic skills for agentic tasks. In *Second Conference on Language Modeling*.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *Preprint*, arXiv:2504.12516.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung-yi Lee. 2024. Streambench: Towards benchmarking continuous improvement of language agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 107039–107063. Curran Associates, Inc.

Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. 2025. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *Preprint*, arXiv:2510.16079.

BY Yan, Chaofan Li, Hongjin Qian, Shuqi Lu, and Zheng Liu. 2025. General agentic memory via deep research. *arXiv preprint arXiv:2511.18423*.

Cheng Yang, Xuemeng Yang, Licheng Wen, Daocheng Fu, Jianbiao Mei, Rong Wu, Pinlong Cai, Yufan Shen, Nianchen Deng, Botian Shi, Yu Qiao, and Haifeng Li. 2025. Learning on the job: An experience-driven, self-evolving agent for long-horizon tasks. *Preprint*, arXiv:2510.08002.

Z.ai. 2024. GLM-4.6 model guide. Z.ai Documentation. Accessed 2026-02-26.

Guibin Zhang, Muxin Fu, Kun Wang, Guancheng Wan, Miao Yu, and Shuicheng YAN. 2025a. G-memory: Tracing hierarchical memory for multi-agent systems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2025b. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pages 19632–19642. AAAI Press.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin,

Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. 2025. Scaling test-time compute for llm agents. *Preprint*, arXiv:2506.12928.

Appendix

A RefCon and DivCon Detailed Results

Table 3 presents the performance of context-evolving agents using a smaller language model (Qwen3.5 9B). To provide a focused comparison with RefCon and DivCon, we evaluate a subset of the most competitive methods previously shown in Table 1. We observed a significant performance decline on the AppWorld benchmark, whereas the drop on BFCL-V3 remained marginal. In settings without ground-truth labels, RefCon consistently outperformed other context-evolving agents, even surpassing ACE (with ground-truth) on AppWorld. However, these gains were less pronounced than those achieved with GLM-4.6 in Table 1, suggesting that while RefCon's scaling is effective, smaller models still struggle with the nuances of memory extraction and reuse. Notably, some context-evolving methods even degraded performance on AppWorld. Furthermore, while RefCon led its category on BFCL-V3, it still fell short of the standard ReAct baseline. DivCon also performs worse than RefCon showing that exploring diversity by prompt alone is still hard for small model.

In Table 1 and Table 2, we report Avg@3 for both RefCon and DivCon. For RefCon, the reported Avg@3 comes from the third run (after the second refinement), whereas for DivCon it comes from the first run. We execute RefCon three times, yielding first-, second-, and final-run results for each trial; the same setup is used for DivCon across its first and second runs. Because RefCon applies self-refinement, later runs can improve accuracy (Table 4). By contrast, DivCon emphasizes diversity, which does not always lead to better scores (Table 5). Therefore, we report the final run for RefCon and the first run for DivCon for all experiments.

B Memory Examples

ACE, ReasoningBank, and ReMe use different memory schemas. ACE stores each memory as a *section* and *content* pair (Table 6); the section acts as a stable bucket (e.g., strategy, tool usage, or caution) so related memories can be organized and retrieved consistently. ReasoningBank represents memory with *title*, *description*, and *content* (Table 7); this structure separates a short identifier (title), a concise summary for quick screening (description), and the full actionable detail (content), which is useful when comparing many candidate memories during retrieval. ReMe uses *when_to_use* and *content* (Table 8); this emphasizes direct applicability by explicitly encoding the trigger condition first, followed by the action or rule to execute. Furthermore, ReMe retrieved memories will be rewritten into a coherent step-by-step guide paragraph. All examples are extracted from trajectories generated for the same query: "How many unique songs are there across my Spotify song library, albums library and all playlists?" (*fac291d_1*) The underlying agent frameworks differ, but all examples use the same scaling method, RefCon.

C Example of Successful RefCon Improvement

This section presents a task that RefCon solves but sequential scaling fails on. For analysis, we compare retrieved memories (Table 9) and trajectory snippets (Table 10). The task is *0d8a4ee_2* with the query: "Send the following phone message to my siblings and roommates, who do not have a venmo account, "Get on venmo please!". Although each method retrieves five memories, we display only two that differ and materially influence trajectory generation. The memories retrieved by sequential scaling push the agent to skip verification and jump directly to sending messages. In contrast, RefCon retrieves memories that encourage API exploration and case checking. This suggests that RefCon extracts more useful memories by self-contrasting across multiple trajectories, while sequential scaling extracts weak guidance from a single (possibly failed) trajectory. The effect is visible in the rollouts: sequential scaling brute-forces API field combinations, fails, and then sends messages to `all_contact`. On the other hand, RefCon first inspects the data structure, then refines the code and correctly identifies `contacts_without_venmo`.

Table 3: Performance of context-evolving agent methods on AppWorld and BFCL-V3 Using Qwen3.5 9B.

Methods	AppWorld				BFCL-V3	
	Avg@3		Pass@3		Avg@3	Pass@3
	TGC	SGC	TGC	SGC		
Baseline						
ReAct	$18.71^{\pm 2.19}$	$10.53^{\pm 4.30}$	29.82	15.79	$53.33^{\pm 3.40}$	72.00
With Gold Labels/Ground-Truth						
ACE (with Ground-Truth)	$18.13^{\pm 4.60}$	$8.77^{\pm 2.48}$	29.82	15.79	$56.00^{\pm 4.32}$	80.00
Without Gold Labels/Ground-Truth						
ReasoningBank (Sequential Scaling)	$16.96^{\pm 2.48}$	$8.77^{\pm 4.96}$	22.81	15.79	$45.33^{\pm 0.94}$	68.00
ReasoningBank (Parallel Scaling)	$17.54^{\pm 2.48}$	$12.28^{\pm 4.96}$	22.81	15.79	$43.33^{\pm 5.25}$	62.00
DivCon	$16.96^{\pm 4.38}$	$12.28^{\pm 2.48}$	26.32	21.05	$46.00^{\pm 4.90}$	72.00
RefCon	**$19.88^{\pm 5.42}$**	**$12.28^{\pm 4.96}$**	**31.58**	**21.05**	**$50.67^{\pm 8.38}$**	**80.00**

Table 4: RefCon accuracy across runs for each method.

Methods	AppWorld			
	Avg@3		Pass@3	
	TGC	SGC	TGC	SGC
ACE				
First Run	$68.40^{\pm 5.72}$	**$43.87^{\pm 6.57}$**	85.96	68.42
Second Run (First Refinement)	$70.17^{\pm 6.57}$	$42.1^{\pm 8.57}$	**92.98**	**84.21**
Third Run (Second Refinement)	**$72.53^{\pm 5.82}$**	**$43.87^{\pm 6.57}$**	**92.98**	78.95
ReasoningBank				
First Run	$70.76^{\pm 0.80}$	$47.37^{\pm 8.61}$	89.47	**84.21**
Second Run (First Refinement)	**$71.93^{\pm 2.86}$**	**$50.87^{\pm 10.85}$**	91.23	**84.21**
Third Run (Second Refinement)	**$71.93^{\pm 3.80}$**	$49.13^{\pm 13.13}$	**92.98**	**84.21**
ReMe				
First Run	$69.59^{\pm 2.98}$	$50.87^{\pm 2.48}$	89.47	**78.95**
Second Run (First Refinement)	$71.35^{\pm 4.38}$	$49.13^{\pm 2.48}$	91.23	**78.95**
Third Run (Second Refinement)	**$74.30^{\pm 0.94}$**	**$52.67^{\pm 7.45}$**	**92.98**	**78.95**

D Hyperparameter Details

Across all methods, we prompt the LLM to extract at most five memories per extraction step. For parallel scaling, we rollout three trajectories at temperatures of 0.7, 0.85, and 1.0 to ensure diversity; otherwise, the default temperature is 0.7. Method-specific configuration are as follows:

- **ReasoningBank**: Uses top-1 query retrieval, returning up to five memories for each query.
- **ACE**: The playbook is capped at 50 memories, and ground-truth usage is disabled during scaling.
- **ReMe & RefCon**: We use a top-k retrieval of 10, the only choose top-5 after reranking. We set a deduplication similarity threshold $\epsilon = 0.5$, and pruning thresholds $\alpha = 5n$ (usage) and $\beta = 0.5$ (utility), where n represents the scaling factor.

E Prompt Details

Prompt details for memory extraction and memory retrieval for each method: ACE (Zhang et al., 2025b), ReasoningBank (Ouyang et al., 2025), and ReMe (Cao et al., 2025) can be found on the original paper. We only modify the number of trajectory provided in the prompt template for RefCon, DivCon, and parallel scaling, while the instruction remains the same. For memory extraction, RefCon adopts ReasoningBank's self-contrast reasoning and ReMe's memory format; therefore, Table 11 presents the combined prompt template. We also provide the self-refine and self-diversity prompts for AppWorld in Table 12, and the corresponding prompts for BFCL-V3 in Table 13. In our experiments, applying the simpler AppWorld-style refinement prompt to BFCL-V3 was ineffective, since the trajectories showed little to no change. To address this, we include both a critique prompt and an ideation prompt to better guide tra-

Table 5: DivCon accuracy across runs for each method.

Methods	**AppWorld**			
	Avg@3		Pass@3	
	TGC	SGC	TGC	SGC
ACE				
First Run	**69.00**$^{\pm 2.16}$	**42.10**$^{\pm 4.33}$	**91.23**	**78.95**
Second Run (First Alternative)	65.50$^{\pm 2.98}$	40.37$^{\pm 6.57}$	89.47	73.68
ReasoningBank				
First Run	**76.03**$^{\pm 2.21}$	**52.67**$^{\pm 0.00}$	**94.74**	**89.47**
Second Run (First Alternative)	70.17$^{\pm 2.50}$	47.33$^{\pm 7.45}$	92.98	**89.47**
ReMe				
First Run	**71.93**$^{\pm 5.76}$	**47.37**$^{\pm 4.29}$	92.98	78.95
Second Run (First Alternative)	67.25$^{\pm 2.19}$	38.60$^{\pm 2.48}$	**96.49**	**89.47**

Table 6: Example of memories extracted using ACE with RefCon scaling.

Section: `strategies_and_hard_rules`
Content: For Spotify authentication, always call `apis.supervisor.show_account_passwords()` to retrieve stored credentials instead of creating dummy passwords.

Section: `strategies_and_hard_rules`
Content: When collecting songs across multiple libraries (songs, albums, playlists), use `set()` to deduplicate `song_ids` from all sources before processing.

Section: `strategies_and_hard_rules`
Content: Music library query workflow: 1) Authenticate → 2) Collect song IDs from all libraries (songs/albums/playlists) → 3) Deduplicate using `set()` → 4) Get song details via `show_song` for each ID → 5) Filter by criteria (genre, etc.) → 6) Sort by `play_count` → 7) Format output.

Section: `apis_to_use_for_specific_information`
Content: For Spotify authentication, always call `apis.supervisor.show_account_passwords()` to retrieve stored credentials instead of creating dummy passwords.

jectory refinement, as shown in Table 13. However, the multi-turn structure of BFCL-V3 makes self-refine and self-diversity more challenging, even after adding an extra guidance step. In some cases, self-refinement can even degrade trajectories that were already correct.

Table 7: Example of memories extracted using ReasoningBank with RefCon scaling.

Title: Authenticate Before Accessing Private Data
Description: Always authenticate with the service before attempting to access private user data like libraries.
Content: When working with music streaming services, private library data (songs, albums, playlists) requires authentication. First explore available APIs, then use stored credentials to obtain an access token before making any data retrieval requests.

Title: Handle Pagination for Complete Data Retrieval
Description: Implement pagination loops to retrieve all items from paginated API responses.
Content: Many APIs return data in pages with limits. Use while loops that increment `page_index` until an empty response is returned, collecting all items across pages to ensure complete data retrieval.

Title: Extract Song IDs from Different Library Types
Description: Different library types require different approaches to extract song identifiers.
Content: Song libraries directly return song objects with `song_id` fields, while album libraries return albums containing `song_ids` arrays, and playlist libraries return playlists with `song_ids` arrays. Extract and flatten these appropriately.

Title: Use Sets for Unique Counting
Description: Use set data structures to automatically handle uniqueness when counting items.
Content: After collecting all song IDs from various sources, add them to a set which automatically handles duplicates. The final count is simply the length of this set.

Table 8: Example of memories extracted using ReMe with RefCon scaling.

Original Memory
When To Use: When working with paginated API responses **Content:** All trajectories successfully implemented pagination with while loops and `page_index` incrementing. This consistent pattern across all successful trajectories shows that proper pagination handling is critical for complete data retrieval. The pattern of checking for empty responses to break the loop is reliable. **When To Use:** When handling authentication tokens in multi-step workflows **Content:** All trajectories successfully extracted and reused the `access_token` from login responses across multiple API calls. The consistent pattern of storing `spotify_access_token = login_result['access_token']` and passing it to subsequent calls demonstrates proper token management. None of the trajectories encountered token expiration issues, suggesting the token lifespan was sufficient for the workflow duration. **When To Use:** When handling data deduplication across multiple sources **Content:** All trajectories correctly used sets for deduplication, but Trajectory 3 demonstrated the most elegant approach by updating the set with comprehensions within the helper function. Trajectories 1, 2, and 4 used more verbose loop structures. The set-based approach was universally successful, proving it's the right pattern for this type of unique counting task across multiple data sources. **When To Use:** When deciding whether to explore API documentation before implementation **Content:** Trajectory 3 successfully skipped explicit API documentation exploration and went directly to implementation, demonstrating that prior knowledge or confidence in API structure can eliminate unnecessary exploration steps. However, Trajectories 1, 2, and 4 took a more cautious approach by examining API docs first, which is safer for unfamiliar APIs but adds overhead. The doc exploration didn't change the implementation approach, suggesting it was redundant in this case.
Rewritten Memory
To accurately count unique songs across your Spotify library, you'll need to implement a comprehensive data aggregation approach that handles multiple library sources with potential overlaps. **Start by authenticating** with Spotify's API using stored credentials, **then systematically explore the available endpoints** for songs, albums, and playlists. **The key is to use a set data structure** to track unique song IDs as you paginate through each library source. **Implement while loops** to handle pagination for songs library, albums library, and all playlists separately, adding each song ID to your set which automatically handles deduplication across all sources. This ensures you capture every unique song exactly once, even if the same track appears in multiple playlists or albums. After processing all paginated results from all three library types, the final count is simply the size of your unique song IDs set. This approach efficiently handles the overlapping nature of Spotify's library structure while providing an accurate count of your total unique songs.

Table 9: Retrieved memory comparison of sequential (failed) and RefCon (success) scaling.

Retrieved Memory (Sequential Scaling)
When To Use: When you encounter verification issue in Venmo **Content:** If you encounter issues with optional verification steps (like checking Venmo status), proceed with the main task (sending messages) to ensure completion. The primary goal of delivering the message is more important than perfect filtering, especially when verification may be unreliable. **When To Use:** When filtering contacts by a condition and finding no matches, but the user has requested to send a specific message **Content:** If no recipients match the specified filter criteria, consider alternative interpretations of the user's request before concluding the task. The message content itself may provide clues about the intended audience.
Retrieved Memory (RefCon)
When To Use: When filtering contact on Venmo **Content:** Across trajectories that attempted Venmo filtering, the reliable pattern was to query users by email first, then validate exact email matches (case-insensitive) before excluding a contact from messaging. Trajectories that treated any non-empty search response as a match risked false positives, while trajectories that performed exact email verification were more robust. In this setup, an empty search result consistently signaled no Venmo account. **When To Use:** When using Venmo API for sending money **Content:** Trajectory 1 initially failed trying to use `receiver_phone` parameter, then succeeded with `receiver_email`. All successful trajectories used `receiver_email` from contact information. PRINCIPLE: Always check API documentation for required parameters—Venmo specifically requires `receiver_email`, not `receiver_phone`, even when you have the phone number. This also applies to any endpoint when you failed to use it properly.

Table 10: Trajectory comparison with memory retrieved from sequential (failed) and RefCon (success) scaling.

Sequential Scaling

```
...
Assistant:
contacts_without_venmo = []
for contact in all_contacts:
  email = contact['email']
  search_result = apis.venmo.search_users(
    access_token=venmo_access_token, query=email, page_limit=1)
  if not search_result:
    contacts_without_venmo.append(contact)
print(contacts_without_venmo)
Environment: []
Assistant:
contacts_without_venmo = []
for contact in all_contacts:
  email = contact['email']
  search_result = apis.venmo.search_users(
    access_token=venmo_access_token, query=email)
  if not search_result:
    contacts_without_venmo.append(contact)
print(contacts_without_venmo)
Environment: []
Assistant:
contacts_without_venmo = []
for contact in all_contacts:
  full_name = f"{contact['first_name']} {contact['last_name']}"
  search_result = apis.venmo.search_users(
    access_token=venmo_access_token, query=full_name)
  if not search_result:
    contacts_without_venmo.append(contact)
print(contacts_without_venmo)
Environment: []
...
```

RefCon

```
...
Assistant:
all_contacts = siblings + roommates
contacts_without_venmo = []
for contact in all_contacts:
  search_result = apis.venmo.search_users(
    access_token=venmo_access_token, query=contact['email'])
  if not search_result:
    contacts_without_venmo.append(contact)
print(contacts_without_venmo)
Environment: []
Assistant:
print(apis.venmo.search_users(access_token=venmo_access_token, query='kevin.blac@gmail.com'))
Environment: [{"first_name": "Kevin", "last_name": "Blackburn", "email": "kevin.blac@gmail.com",
. . . }]
Assistant:
all_contacts = siblings + roommates
contacts_without_venmo = []
for contact in all_contacts:
  search_result = apis.venmo.search_users(
    access_token=venmo_access_token, query=contact['email'])
  found = any(user['email'] == contact['email'] for user in search_result)
  if not found:
    contacts_without_venmo.append(contact)
print(contacts_without_venmo)
Environment: [{"contact_id": 801, "first_name": "Martin", "email": "mar_blac@gmail.com", . . . }]
...
```

Table 11: Memory extraction prompt used for our context-evolving agent utilizing RefCon. We adopt ReMe memory format, but utilize self-contrast reasoning to extract memory.

Self-Contrast Prompt

You are an expert AI analyst comparing multiple step sequences which might be successful or failed to extract differential insights.
Your task is to compare and contrast these trajectories to identify the most useful and generalizable strategies as memory items using self-contrast reasoning.
Focus on critical decision points, technique variations, and approach differences.
COMPARATIVE ANALYSIS FRAMEWORK:
- DECISION CONTRAST: Compare critical decisions made in success vs failure cases
- TECHNIQUE VARIATIONS: Identify different approaches and their outcomes
- TIMING DIFFERENCES: Analyze when certain actions were taken and their impact
- SUCCESS FACTORS: Extract what specifically made the difference
EXTRACTION PRINCIPLES:
- Frame comparisons as PRINCIPLES as well as case-specific SOLUTIONS
- Identify PATTERNS that differentiate effective vs ineffective approaches
- Extract RULES that can guide future similar situations
- Focus on UNDERLYING MECHANISMS rather than surface-level differences
Trajectory 1:
`{trajectory_1}`
...
Trajectory N:
`{trajectory_n}`
OUTPUT FORMAT:
Generate up to 5 comparative insights as JSON objects:
`[`
`{`
`  "when_to_use": "Specific scenarios where this comparative insight applies",`
`  "experience": "Detailed comparison highlighting why success approach works better",`
`  "tags": ["comparative_analysis", "success_factors", "relevant_keywords"],`
`  "confidence": 0.8,`
`  "step_type": "reasoning|action|observation|decision",`
`  "tools_used": ["list", "of", "tools"]`
`}`
`]`

Table 12: RefCon and DivCon trajectory generation prompt for AppWorld.

Self-Refine Prompt for AppWorld
Let's carefully re-examine the previous trajectory, including your reasoning steps and action taken. Pay special attention to whether you used the best API sequence and whether you used the API correctly. If you find inconsistencies, correct them. If everything seems correct, make it more efficient. Now, solve the same problem again from scratch.
Self-Diversity Prompt for AppWorld
Below is the previous trajectory, the solution might be correct or wrong. Now solve the same problem using a DIFFERENT reasoning approach. Focus on exploring alternative strategies.

Table 13: RefCon and DivCon trajectory generation prompt for BFCL-V3.

Critique Trajectory Prompt for Self-Refine
You are an expert reviewer analyzing an AI assistant's multi-turn tool-calling trajectory. Your job is to identify mistakes, missed actions, and suboptimal decisions. For each turn in the trajectory, evaluate: 1. Did the assistant call the appropriate tools? If a user requested an action (e.g., book, cancel, update), did the assistant actually make a tool call, or did it just respond with text? 2. Were the tool arguments correct? Check for wrong parameter values, missing required arguments, or arguments that contradict the user's request. 3. Did the assistant use information from previous tool responses correctly? For example, if a lookup returned an ID, did the assistant use that ID in subsequent calls? 4. Were there any unnecessary or redundant tool calls? 5. Did the assistant follow the logical sequence of operations? (e.g., lookup before booking, authenticate before accessing protected resources) Be specific about which turns have issues and what should be done differently. If a turn looks correct, briefly note it as OK. Focus most of your analysis on turns that seem problematic. IMPORTANT: You must respond ONLY with your critique. Do not attempt to solve the task yourself.
Self-Refine Prompt for BFCL-V3
You are an AI assistant tasked with completing a multi-turn tool-calling objective. A reviewer has analyzed your previous attempt and provided a critique. 1. Read the critique carefully to understand the mistakes made in the previous trajectory. 2. Trust the Schemas: The critique is a helpful guide, but your provided tool schemas are the absolute truth. If the critique suggests using a tool or parameter that does not exist in your schemas, ignore it. 3. Do not ask the user for any additional information/clarification, you are authorized to do everything in behalf of the user. 4. Restart the task from the beginning. Your environment state has been reset. 5. You are currently at the start of the task, but the provided previous trajectory covers the entire multi-turn interaction. Do NOT execute or simulate future steps ahead of time. Focus ONLY on the current task instruction. 6. You can use related memory to guide your reasoning to solve the problem.
Ideate Solution Prompt for Self-Diversity
You are an expert AI brainstorming assistant analyzing a previous interaction trajectory. Your job is to read the user's request and the previous approach, and then propose a DIFFERENT but valid reasoning approach or sequence of tool calls that solves the same problem. Focus your analysis on: 1. Understanding the user's core intent. 2. Identifying the strategy used in the previous trajectory. 3. Proposing alternative strategies, using different tools, or a different sequence of operations that could achieve the same goal successfully. Be specific about the proposed alternative approach. IMPORTANT: You must respond ONLY with your alternative idea. Do not attempt to solve the task yourself.
Self-Diversity Prompt for BFCL-V3
You are an AI assistant tasked with completing a multi-turn tool-calling objective. A reviewer has analyzed your previous attempt and provided an alternative idea for a different reasoning approach. 1. Read the alternative idea carefully and use it to solve the problem. 2. Trust the Schemas: The alternative idea is a helpful guide, but your provided tool schemas are the absolute truth. If the alternative idea suggests using a tool or parameter that does not exist in your schemas, ignore it. 3. Do not ask the user for any additional information/clarification, you are authorized to do everything in behalf of the user. 4. Restart the task from the beginning. Your environment state has been reset. 5. You are currently at the start of the task, but the provided previous trajectory covers the entire multi-turn interaction. Do NOT execute or simulate future steps ahead of time. Focus ONLY on the current task instruction. 6. You can use related memory to guide your reasoning to solve the problem.