Ask Only When Needed: Proactive Retrieval from Memory and Skills for Experience-Driven Lifelong Agents

Anonymous ACL submission

Abstract

Online lifelong learning agents must decide not only how to act but also when to consult prior experience to continually improve on long-horizon tasks. Existing methods typically retrieve memories passively, such as at task initialization or after each step, and therefore miss knowledge gaps that arise during interaction. We propose PROACTAGENT, an experience-driven lifelong learning framework for proactive retrieval over a structured Experience Base. PROACTAGENT continually improves through EXPONEVO, which jointly updates policies and refines memory, organizing past interactions into factual, episodic, and skill repositories. It further introduces PROACTRL, which treats retrieval as an explicit policy action and learns when and what to retrieve. By comparing paired continuations from identical interaction prefixes with and without retrieval, PROACTRL provides step-level process rewards that encourage retrieval only when it improves task outcomes or efficiency. Experiments on SciWorld, AlfWorld, and StuLife show that PROACTAGENT consistently outperforms all baselines, achieving up to 32% relative improvement in success rate and over 33% reduction in interaction rounds. Our code will be publicly available at GitHub.

1 Introduction

Language agents are progressing beyond isolated task-solving toward online lifelong learning, a paradigm in which an agent engages with a continuous stream of interactive tasks while accumulating experience across episodes (Shridhar et al., 2021; Wang et al., 2022, 2023; Cai et al., 2025). Recent advances in chain-of-thought reasoning (Wei et al., 2022; Yao et al., 2023a), tool use (Schick et al., 2023; Yao et al., 2023b), and self-reflection (Madaan et al., 2023; Shinn et al., 2023) have dramatically expanded what an agent can accomplish within a single episode, yet these capabilities yield diminishing returns once the primary bottleneck shifts from within-episode problem-solving to cross-episode online knowledge utilization. Recent benchmarks confirm that even state-of-the-art proprietary models struggle with long-horizon lifelong tasks (Cai et al., 2025), suggesting that raw model capacity alone cannot substitute for the ability to learn from and build upon interaction history.

To realize online experience utilization, a natural and widely adopted approach is to equip agents with an external experience base that stores knowledge accumulated from past interactions (Park et al., 2023; Wang et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Cai et al., 2025). As illustrated in Figure 1, existing methods mainly differ in how retrieval is triggered. Static initialization (Park et al., 2023; Zhong et al., 2024; Wang et al., 2023; Zhao et al., 2024) injects memory once at episode onset, allowing agents to start with relevant context but leaving them unable to seek new information when task dynamics shift. Continuous retrieval (Zhang et al., 2025; Zhao et al., 2024; Tang and Yang, 2024) performs retrieval at every step, increasing access to external experience but often introducing substantial redundancy and context overload. These studies demonstrate the promise of memory-augmented lifelong agents, but also reveal that effective experience utilization depends critically on how retrieval is controlled and how accumulated experience is updated over time.

However, existing lifelong agents still face two fundamental limitations. First, current retrieval mechanisms remain essentially passive: retrieval is triggered by pre-defined positions, externally designed rules, or separate gating modules, rather than being learned as an intrinsic capability of the agent itself. As a result, agents struggle to recognize knowledge gaps during interaction and proactively retrieve the most useful past experience for the current decision. Second, existing online update strategies typically focus on either textual

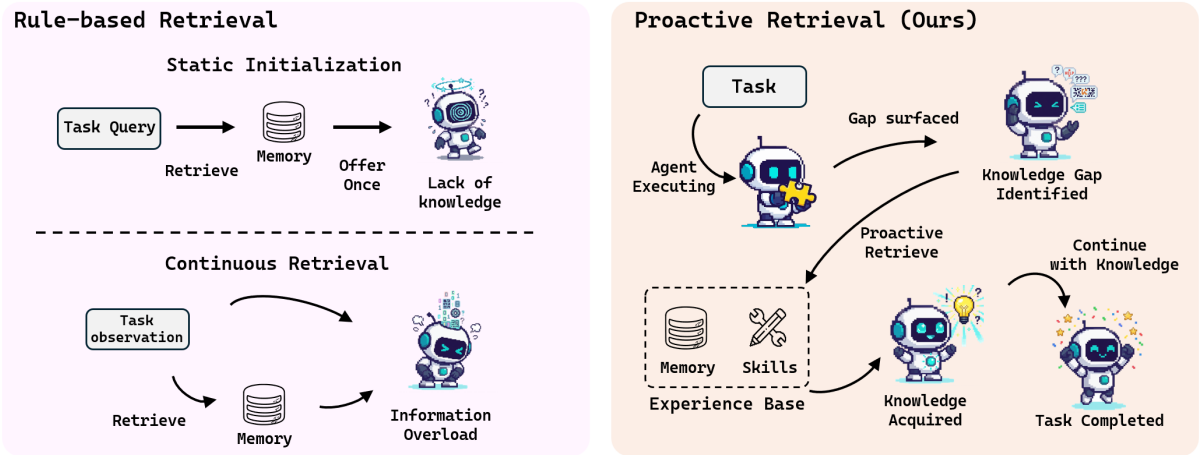

Figure 1: Comparison of retrieval strategies for online lifelong agents. Static initialization provides memory once at task start, failing when task dynamics shift. Continuous retrieval queries at every step, causing context overload. Proactive retrieval (ours) learns when to retrieve through paired-branch process rewards, enabling the agent to recognize knowledge gaps and retrieve precisely when needed.

memory accumulation or parameter optimization, while treating the two as largely independent processes. In practice, both are indispensable for lifelong adaptation. Textual memory preserves and expands externalized experience, including facts, episodes, and reusable skills, whereas parameter updates improve the agent's internal policy for future decision making. Updating only one side, or evolving them in isolation, limits the agent's ability to continually improve from interaction history. Therefore, an effective lifelong agent must address both challenges simultaneously: it should learn to retrieve experience proactively, while also jointly evolving its memory and policy online.

To address these challenges, we propose PROACTAGENT, an experience-driven lifelong learning framework that unifies proactive retrieval with experience-enhanced online evolution over a structured experience base. Specifically, PROACTAGENT consists of two tightly coupled components. The first component, Experience-Enhanced Online Evolution (EXPONEVO), jointly improves the agent through memory refinement and policy optimization, enabling it to update both what experience it stores and how it acts during online interaction. The experience base organizes historical interactions into typed repositories, including factual memory, episodic memory, and behavioral skills, so that retrieval can provide both relevant evidence and actionable guidance. The second component, Proactive Reinforcement Learning-based Retrieval (PROACTRL), formulates retrieval as an explicit policy action and learns both when and what to retrieve through paired-branch process rewards. By comparing continuations from identical interaction prefixes with and without retrieval, PROACTRL provides step-level supervision for retrieval decisions and encourages retrieval only when it leads to better task outcomes or higher efficiency.

We evaluate PROACTAGENT on SciWorld, AlfWorld, and StuLife. It achieves 73.50% and 71.28% success rates on SciWorld and AlfWorld with significantly reduced retrieval overhead, while attaining performance competitive with proprietary models on StuLife. These results demonstrate that proactive retrieval, coupled with joint memory and policy evolution, substantially enhances both the effectiveness and efficiency of online lifelong agents.

The main contributions are as follows:

- We propose PROACTAGENT, a proactive online lifelong learning framework enabling agents to continually evolve from interaction history rather than relying on passively invoked experience.
- We design two key components within PROACTAGENT: Experience-Enhanced Online Evolution (EXPONEVO), which jointly improves textual memory and policy parameters during online interaction, and Proactive Reinforcement Learning-based Retrieval (PROACTRL), which learns when and what to retrieve through paired-branch process rewards.
- We conduct extensive experiments on SciWorld, AlfWorld, and StuLife, showing that PROACTAGENT consistently improves lifelong agent performance and retrieval efficiency, and attains performance competitive with proprietary models.

2 Method

We propose PROACTAGENT, an experience-driven lifelong learning framework enabling agents to continually evolve from interaction history via joint experience refinement, policy optimization, and proactive retrieval over a structured experience base. It comprises two tightly coupled components: Experience-Enhanced Online Evolution (EXPONEVO) maintains a closed loop of acting, experience accumulation, and policy improvement; and Proactive Reinforcement Learning-based Retrieval (PROACTRL) formulates retrieval as an explicit policy action, learning when and what to retrieve through step-level supervision.

2.1 Formal Definition

We formulate each interactive task instance as a goal-conditioned partially observable Markov decision process. A task instance is defined as

$$x = \langle \mathcal{E}, o_0, g \rangle, \qquad \mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{G}, P, R, \Omega, O, \gamma), \tag{1}$$

where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{G}$, and Ω denote the state, action, goal, and observation spaces, respectively; $P(s' \mid s, a)$ represents the transition function; $R(s, a, g)$ represents the goal-conditioned reward; $O(o \mid s', a)$ denotes the observation model; and $\gamma \in [0, 1)$ is the discount factor. Given a horizon T, at step t, the agent receives a partial observation $o_t \in \Omega$ and constructs the interaction history

$$h_t = (x, o_1, a_1, \ldots, o_t), \tag{2}$$

where a_t is the action generated by the model at step t. We augment the action space to $\mathcal{A} = \mathcal{A}_{\text{env}} \cup \mathcal{A}_{\text{ret}}$, so the policy can output either an environment action $a_t \in \mathcal{A}_{\text{env}}$ or a retrieval action $a_t = \textsc{Retrieve}(q_t)$, where q_t denotes a natural-language query. If the agent triggers a retrieval action, the returned experience $\mathcal{D}_t$ is appended to the context before the subsequent decision; otherwise, $\mathcal{D}_t = \varnothing$. A trajectory is the ordered sequence $\tau = \big((h_t, a_t, \mathcal{D}_t, r_t^{\text{env}})\big)_{t=1}^{T}$.

As the agent processes successive tasks, its experience base $\mathcal{D}$ grows continuously while the behavioral policy adapts through interaction. The central challenge is learning to proactively identify knowledge gaps and selectively retrieve from this expanding repository, since outcome-level rewards alone cannot provide step-level supervision for individual retrieval decisions. PROACTAGENT addresses this through two tightly coupled mechanisms (Figure 2). Section 2.2 presents Experience-Enhanced Online Evolution, which organizes experience into a structured base and interleaves experience accumulation with policy optimization in a closed loop. Section 2.3 introduces PROACTRL, which provides step-level supervision for retrieval decisions through adaptive rollout with paired-branch process rewards, enabling the agent to learn when and what to retrieve. Implementation details are provided in the Appendix C.

2.2 Experience-Enhanced Online Evolution

In the online lifelong learning paradigm, an agent engages with a continuous stream of interactive tasks while two resources evolve through successive episodes: the accumulated experience base and the behavioral policy. These resources are naturally complementary. The experience base supplies factual knowledge, episodic precedents, and distilled behavioral patterns from which the policy can learn; a stronger policy, in turn, generates higher-quality trajectories that yield richer entries for the experience base. PROACTAGENT closes this loop by unifying experience accumulation and policy optimization into a single iterative cycle (Figure 2 (a)): given a stream of tasks, the agent (1) interacts with the environment under the current policy π_θ, augmented by proactive retrieval from a structured experience base $\mathcal{D}$; (2) distills completed trajectories into typed experience entries that are incorporated into $\mathcal{D}$; and (3) updates π_θ via reinforcement learning on the collected trajectories, where retrieval decisions are optimized jointly with task actions through paired-branch process rewards (Section 2.3). This creates a self-reinforcing loop: a richer experience base provides more relevant retrieval results, which improves trajectory quality, which in turn yields higher-quality experience entries and stronger policy gradients.

A critical design choice within this framework is how accumulated experience is organized for effective retrieval. At decision time, an interactive agent benefits from two distinct types of support: relevant memory (factual knowledge or similar prior episodes) and behavioral guidance (which action patterns succeed or fail in analogous states). Conflating all experience into a single undifferentiated pool mixes these two modes of support, producing redundant or competing retrieval results. To address this, the experience base maintains a struc-

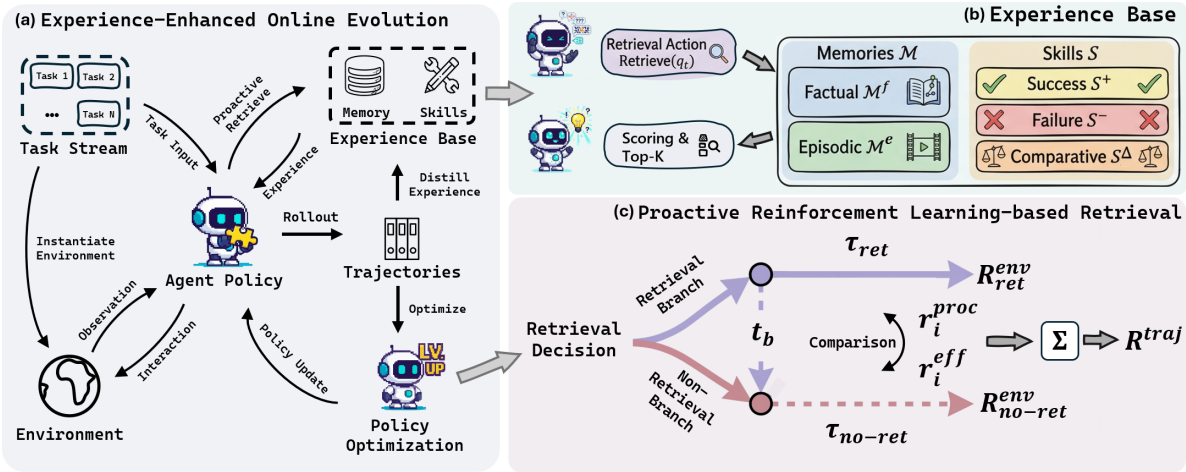

Figure 2: Overview of PROACTAGENT. (**a**) Experience-Enhanced Online Evolution (EXPONEVO) closes the loop between acting, experience accumulation, and policy optimization. (**b**) EXPERIENCE BASE partitions experience into five typed stores ($\mathcal{M}^{\mathrm{f}}$, $\mathcal{M}^{\mathrm{e}}$, $\mathcal{S}^+$, $\mathcal{S}^-$, $\mathcal{S}^\Delta$), so a single query returns complementary evidence and behavioral guidance. (**c**) Proactive Reinforcement Learning-based Retrieval (PROACTRL) replays the shared prefix to produce a retrieval branch τ^{ret} and a matched no-retrieval branch $\tau^{\mathrm{no\text{-}ret}}$; the outcome gap yields a process reward that is positive only when retrieval strictly improves the trajectory, providing supervision for when and what to retrieve.

tured repository (Figure 2 (b)):

$$\mathcal{D} = \mathcal{M} \cup \mathcal{S}, \quad \mathcal{M} = \mathcal{M}^{\mathrm{f}} \cup \mathcal{M}^{\mathrm{e}}, \\ \mathcal{S} = \mathcal{S}^+ \cup \mathcal{S}^- \cup \mathcal{S}^\Delta, \tag{3}$$

where $\mathcal{M}^{\mathrm{f}}$ and $\mathcal{M}^{\mathrm{e}}$ denote factual and episodic memories, and $\mathcal{S}^+$, $\mathcal{S}^-$, $\mathcal{S}^\Delta$ denote distilled behavioral skills from successes, failures, and comparative evaluations. Each entry $r \in \mathcal{D}$ stores textual content, a type label, an embedding vector $e(r)$, and a priority score $p(r)$. Given a query q_t, the experience base returns a type-balanced experience:

$$\mathcal{D}_t = \bigcup_{C \in \mathcal{C}} \mathrm{TopK}(q_t, C, k_C), \quad \sum_{C \in \mathcal{C}} k_C = K, \\ \mathcal{C} = \{\mathcal{M}^{\mathrm{f}}, \mathcal{M}^{\mathrm{e}}, \mathcal{S}^+, \mathcal{S}^-, \mathcal{S}^\Delta\}, \tag{4}$$

where entries within each subset are ranked by:

$$\mathrm{score}(q_t, r) = \mathrm{sim}(e(q_t), e(r)) + \lambda_p \, p(r). \tag{5}$$

The similarity term ensures query relevance, while the priority term softly favors entries with proven utility. This typed decomposition enables a single query to return complementary evidence (what is true, what happened before) and guidance (what to do, what to avoid), mitigating redundancy in monolithic memory systems.

After each episode, the evolution framework distills completed trajectories into new entries that grow the structured base. Factual and episodic entries capture environment states and interaction patterns from individual trajectories. Success and failure skills abstract reusable strategies and error patterns from outcome-specific trajectory subsets. Comparative skills, which encode why one continuation outperforms another, are distilled from the paired rollout branches produced by PROACTRL (Section 2.3), exploiting shared prefixes to provide the most localized contrastive signal. Priority scores are updated only for entries that were actually retrieved and associated with improved outcomes, creating a lightweight mechanism that progressively surfaces high-value experience.

Within the online evolution framework, retrieval is not a passive heuristic but an explicit policy action optimized jointly with task behavior. As the experience base grows through continued interaction, selective retrieval becomes increasingly critical: indiscriminate querying floods context with irrelevant information, while missed opportunities leave the agent repeating past errors. However, standard outcome-level rewards cannot provide the step-level supervision needed to learn which retrieval decisions are beneficial. The next section introduces PROACTRL, which addresses this through paired-branch process rewards.

2.3 Proactive Reinforcement Learning-based Retrieval

Within the online evolution framework, the agent must learn when retrieval actively improves outcomes versus when current knowledge suffices. Let $z_t = \mathbf{1}[a_t = \mathrm{RETRIEVE}(q_t)]$ denote whether a retrieval action is executed at step t. The core diffi-

culty is that outcome-level rewards cannot isolate whether a specific retrieval decision was beneficial. Upon failure, it remains unclear if retrieval was unnecessary, mistimed, or vague. Therefore, learning proactive retrieval requires step-level supervision answering: did this retrieval improve the trajectory, or would the agent have succeeded without it?

To address this challenge, we introduce an adaptive rollout mechanism that generates comparative evidence (Figure 2 (c)). Whenever an initial rollout triggers a retrieval action at step t_b, we adaptively sample alternative continuations from the same interaction prefix h_{t_b}:

$$\begin{aligned} \tau^{\text{ret}} &= \tau_{<t_b} \oplus \tau^{\text{ret}}_{\geq t_b}, \\ \tau^{\text{no-ret}} &= \text{Replay}(\tau_{<t_b}) \oplus \tau^{\text{no-ret}}_{\geq t_b}. \end{aligned} \tag{6}$$

Here $\tau_{<t_b}$ and $\tau_{\geq t_b}$ denote the ordered prefix and suffix subsequences of τ, respectively, and $\oplus$ denotes sequence concatenation. The rollout τ^{ret} preserves the retrieval decision and its subsequent context, whereas $\tau^{\text{no-ret}}$ explores alternative actions from the same prefix by temporarily suppressing the retrieval action at t_b. Because both rollouts share the same history before the branching point, the divergence between them provides a comparative signal: if τ^{ret} outperforms $\tau^{\text{no-ret}}$, the retrieval was proactive and necessary; if not, the retrieval was passive or redundant. This paired-branch comparison directly isolates the utility of the specific retrieval decision, enabling step-level process rewards for proactive control.

For a branched trajectory pair indexed by i, let $j(i)$ denote the matched no-retrieval counterpart of rollout i, and let $z_i \equiv z^{(i)}_{t_b} \in \{0, 1\}$ denote the retrieval indicator at the branching step. PROACTRL defines the trajectory-level reward for reinforcement learning as follows:

$$R^{\text{traj}}_i = R^{\text{env}}_i + r^{\text{proc}}_i + r^{\text{eff}}_i, \tag{7}$$

where $R^{\text{env}}_i = \sum_{t=1}^{T_i} r^{\text{env}}_t$ denotes the cumulative environment reward of trajectory i. For a trajectory i that includes a retrieval action, we evaluate its advantage over the adaptively sampled non-retrieval counterpart $j(i)$ by computing the rollout margin:

$$\Delta_i = \Big(R^{\text{env}}_i - R^{\text{env}}_{j(i)}\Big) + \lambda_T \frac{T_{j(i)} - T_i}{\max(T_{j(i)}, 1)}, \tag{8}$$

where T_i denotes the number of interaction steps. The retrieval process reward is then defined as

$$r^{\text{proc}}_i = \begin{cases} \alpha, & z_i > 0 \text{ and } \Delta_i > 0, \\ -\alpha, & z_i > 0 \text{ and } \Delta_i < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

This process reward directly enables proactive retrieval control. It penalizes passive or vague queries ($\Delta_i < 0$) where retrieval provides no benefit over proceeding without it, thereby teaching the agent when not to retrieve and what to avoid querying. Conversely, it provides explicit positive reinforcement (α) only when the specific query content actively unlocks a superior or more efficient continuation, teaching the agent exactly when a query is necessary and what information is most effective to retrieve. By learning from these paired comparisons, the agent develops genuine proactivity, i.e., the ability to recognize when current knowledge suffices versus when retrieval is needed.

To further penalize wasteful retrieval actions, we introduce the following efficiency penalty term:

$$\begin{aligned} r^{\text{eff}}_i = &-w_q \, \mathbf{1}[\text{repeat}_i] \\ &+ \text{clip}\Big(w_t \frac{\bar{T}_{g(i)} - T_i}{\max(\bar{T}_{g(i)}, 1)}, -|w_t|, |w_t|\Big), \end{aligned} \tag{10}$$

where repeat_i equals 1 if rollout i repeats a query string that already appeared earlier in the same trajectory and 0 otherwise, $g(i)$ denotes the goal associated with rollout i, and $\bar{T}_{g(i)}$ is the empirical average length of successful trajectories for goal $g(i)$. This term discourages repeating identical queries and provides rewards for shorter, successful trajectories. The formal proof showing that this process reward provides supervision for proactive retrieval is provided in the appendix. Given a task instance x, we sample a group of G rollouts $\{\tau^{(1)}, \ldots, \tau^{(G)}\}$ and compute the standard normalized advantage for GRPO as follows:

$$A_i = \frac{R^{\text{traj}}_i - \text{mean}(\{R^{\text{traj}}_j\}_{j=1}^G)}{\text{std}(\{R^{\text{traj}}_j\}_{j=1}^G) + \epsilon}. \tag{11}$$

The optimization of the policy is performed using the standard GRPO objective.

Together, the EXPONEVO and PROACTRL form a self-reinforcing cycle: improved trajectories yield higher-quality experience entries, and this richer experience enables more precise retrieval control via paired-branch comparisons. This mutual reinforcement distinguishes PROACTAGENT from isolated

approaches, empowering it to proactively identify knowledge gaps and initiate searches, achieving higher task success rates and efficiency.

3 Experiments

Our experiments are designed to answer three research questions:

- **RQ 1**: Does learned proactive retrieval outperform passive retrieval baselines? (§3.2, §3.4)
- **RQ 2**: Does joint co-evolution of experience and policy outperform approaches that evolve either resource in isolation, and what is the contribution of each component? (§3.2, §3.3)
- **RQ 3**: Does proactive retrieval improve both efficiency and accuracy, and generalize across model scales? (§3.4)

3.1 Experimental setup

Datasets. We evaluate PROACTAGENT on three interactive benchmarks: SciWorld (Wang et al., 2022), AlfWorld (Shridhar et al., 2021), and StuLife (Cai et al., 2025). SciWorld and AlfWorld report the success rate (SR) and the average number of interaction rounds, whereas StuLife reports the SR and StuGPA. For SciWorld and AlfWorld, we adopt the training splits of AgentGym-RL (Xi et al., 2025b) and evaluate on the test set of AgentGym-RL (SciWorld) and the official test set (AlfWorld). Because StuLife provides no training set, evaluation on this benchmark is restricted to methods capable of evolving entirely online. For methods that involve reinforcement learning, a cold-start phase on the training set precedes policy optimization; methods without a training set (StuLife) rely solely on online evolution.

Baselines. We evaluate against offline methods that remain static after training (ReAct (Yao et al., 2023b), SFT, GRPO (Shao et al., 2024)); online methods that evolve memory (AWM (Wang et al., 2024), Reflexion (Shinn et al., 2023), MemoryBank (Zhong et al., 2024), Mem0 (Chhikara et al., 2025)); online Test-time RL (Zuo et al., 2025) that evolves policy parameters via online GRPO without structured memory, and GRPO+Reflexion that combines parameter updates with memory accumulation but relies on passive retrieval.

Evaluation protocol. Unless noted otherwise, all primary experiments use Qwen2.5-7B-Instruct as the base model; a scaling study with Qwen2.5-3B-Instruct is also reported. PROACTAGENT performs online co-evolution in which experience accumulation and parameter adaptation are tightly interleaved at single-instance granularity. Implementation details are provided in Appendix E.

3.2 Main Results

Table 1 presents the main results across three benchmarks. We organize the key findings around the three research questions.

Learned proactive retrieval outperforms passive strategies. Online baselines with experience (AWM, MemoryBank, Mem0) lack learned retrieval control and yield only limited gains over the base model. Even GRPO+Reflexion leaves substantial room for improvement. PROACTAGENT achieves an 18.0-point gain over GRPO+Reflexion on SciWorld by learning retrieval as an explicit policy action through paired-branch process rewards.

Joint evolution outperforms isolated approaches. Test-time RL underperforms offline GRPO on SciWorld, suggesting that online updates without structured experience can be unstable on noisy trajectories. Memory-only baselines achieve at most 7.34% SR on StuLife. PROACTAGENT surpasses GRPO+Reflexion and approaches proprietary model performance on StuLife.

Proactive control yields efficiency gains alongside accuracy. PROACTAGENT reduces interaction rounds by 33.2% on SciWorld and 22.5% on AlfWorld relative to GRPO+Reflexion. By learning when current knowledge suffices and when retrieval is needed, proactive control avoids both unnecessary queries that flood the context and missed retrieval opportunities that force the agent to rediscover known solutions.

3.3 Ablation Study

Progressive ablation. We cumulatively remove components from the full PROACTAGENT to measure their layered contribution. Removing EXPONEVO results in the offline variant; the full model improves over it by 4.3% relative in SR, indicating that online parameter evolution is essential to unlock further performance improvements. Further removing PROACTRL reduces SR by 92.2% relative to the offline variant, confirming that learned proactive retrieval is the most impactful component. Finally, removing EXPONEVO and PROACTRL leaves only the raw EXPERIENCE BASE without training or proactive retrieval, demonstrating that structured experience alone is insufficient without online evolution.

Table 1: Main results across three lifelong agent benchmarks. The best results are highlighted in bold.

Method	SciWorld		AlfWorld		StuLife		Avg
	SR ↑	Rounds ↓	SR ↑	Rounds ↓	SR ↑	StuGPA ↑	SR ↑
Offline baselines							
Baseline	2.00	33.40	27.69	32.85	2.56	6.33	10.75
SFT	19.50	30.26	54.36	26.78	5.11	7.27	26.32
GRPO (Shao et al., 2024)	52.00	26.51	67.69	17.84	–	–	–
Online baselines							
Test-time RL (Zuo et al., 2025)	46.50	27.87	68.71	13.99	6.71	10.28	40.64
AWM (Wang et al., 2024)	1.00	35.65	31.28	33.56	3.51	7.60	11.93
Reflexion (Shinn et al., 2023)	4.00	33.73	36.92	38.57	6.82	11.31	15.91
MemoryBank (Zhong et al., 2024)	2.50	30.24	32.31	37.14	6.05	7.69	13.62
Mem0 (Chhikara et al., 2025)	3.00	34.94	33.85	32.11	7.34	10.83	14.73
GRPO+Reflexion	55.50	27.52	67.18	16.42	9.37	13.51	44.02
Ours							
PROACTAGENT	**73.50**	**18.38**	**71.28**	**12.73**	**12.35**	**19.26**	**52.38**

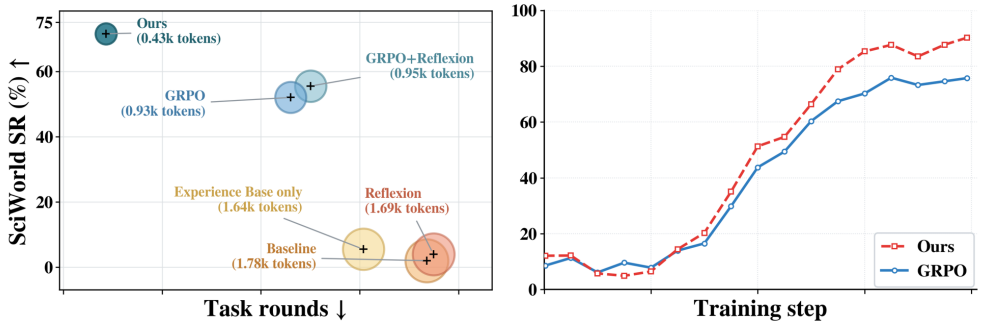

Figure 3: Inference efficiency and training dynamics on SciWorld. Left: PROACTAGENT achieves higher success rates with fewer interaction rounds and lower token consumption than all baselines, where bubble area indicates average prompt tokens per episode. Right: PROACTAGENT consistently outperforms GRPO throughout training, converging to a substantially higher final accuracy.

Table 2: Ablation study on SciWorld.

Variant	SR ↑	Rounds ↓
Progressive ablation		
PROACTAGENT	**73.50**	**18.38**
w/o EXPONEVO	70.50	17.15
w/o PROACTRL	26.50	27.80
w/o EXPONEVO + PROACTRL	5.50	30.18
Component ablation (from w/o online param. evol.)		
Replace $\mathcal{D}$ with Reflexion	62.50	23.15
Remove paired-branch reward	65.00	27.31
Remove cold-start stage	59.50	29.04

Table 3: Component ablation of EXPERIENCE BASE.

Variant	SciWorld SR ↑	AlfWorld SR ↑	StuLife SR ↑
Full	**73.50**	**71.28**	**12.35**
Skill only	69.00	67.69	10.74
Memory only	67.50	66.15	11.68

Component ablation. Starting from the variant without online parameter evolution, we isolate individual design choices. Replacing the structured $\mathcal{D}$ with Reflexion reduces SR by 11.3% relative, confirming that typed decomposition outperforms monolithic memory. Removing paired-branch process rewards while retaining standard GRPO training yields a 7.8% decrease, demonstrating that step-level supervision improves retrieval learning beyond what outcome-level rewards alone provide. Removing the cold-start phase causes the largest single-component decrease of 15.6%, confirming that proactive retrieval tool-calling requires supervised initialization before reinforcement learning can effectively shape retrieval decisions.

Experience ablation. As shown in Table 3, ablating either degrades performance across all benchmarks, confirming complementary roles. Skill-only performs slightly better on SciWorld and AlfWorld, suggesting action abstractions contribute more in task-oriented environments. Memory-only shows relative strength on StuLife, where long-horizon tasks with cross-task dependencies depend more on factual persistence.

3.4 Additional Analysis

Efficiency analysis. The left panel of Figure 3 shows that performance gains stem from proactive control rather than extended contexts. PROACTA-

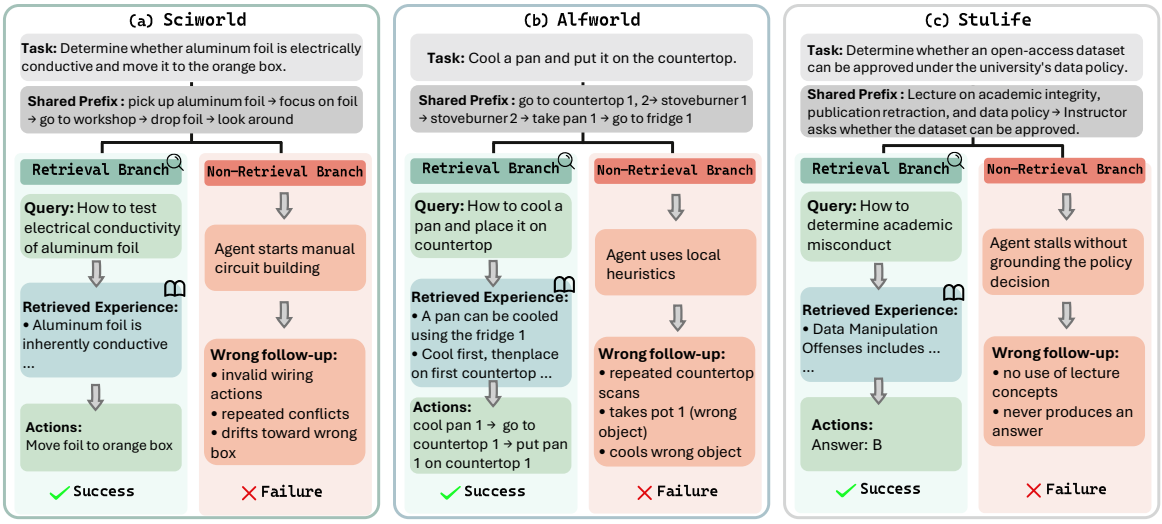

Figure 4: Case studies across SciWorld, ALFWorld, and StuLife. Each panel contrasts a query branch (green) against a matched no-query branch (red) from the same interaction prefix or task instance. In all three cases, a single targeted retrieval at the action-critical decision point leads to immediate success, while the no-query branch drifts into invalid actions, wrong-object selection, or stalled interaction and fails.

Table 4: Model scaling on SciWorld with Qwen2.5-3B-Instruct. Our 3B model nearly matches 7B GRPO+Reflexion in SR while using 48% fewer rounds.

Method	SR ↑	Rounds ↓
GRPO (3B)	24.00	24.14
GRPO+Reflexion (3B)	29.50	20.09
GRPO+Reflexion (7B)	55.50	27.52
PROACTAGENT (3B)	**53.50**	**14.35**

GENT achieves the highest success rate with the fewest interaction rounds and lowest token consumption, reaching 73.50% SR with only 0.43k tokens per task, compared to GRPO+Reflexion's 0.95k tokens for 55.50% SR. The right panel's training dynamics show PROACTRL maintaining a consistent advantage over vanilla GRPO from mid-training onward, confirming that proactive retrieval outperforms passive retrieval.

Model scaling. Table 4 shows that proactive retrieval remains effective at smaller scales. The 3B model achieves 53.50% SR (+24.0 points over 3B GRPO+Reflexion), nearly matching 7B GRPO+Reflexion (55.50%) while requiring 48% fewer interaction rounds. These results suggest that proactive retrieval can partially offset capacity gaps through more selective decision-making.

Benchmark-specific behavior. Each benchmark reveals distinct benefits of proactive retrieval. SciWorld shows the largest SR and efficiency gains, suggesting retrieval timing is the primary bottleneck in long-horizon scientific tasks. AlfWorld exhibits smaller SR but substantial efficiency improvements, indicating it mainly reduces wasted exploration. On StuLife, PROACTAGENT demonstrates strong online adaptation by rivaling leading proprietary systems, suggesting proactive retrieval helps narrow the gap between open-weight and closed-source models in long-dependency tasks.

Case study. Figure 4 contrasts query and no-query branches. Factual retrieval bypasses invalid wiring (SciWorld), procedural retrieval eliminates redundant scanning (ALFWorld), and retrieving misconduct criteria provides knowledge absent from parametric memory (StuLife). In all cases, one targeted query at a decision-critical step replaces a longer, failure-prone trajectory.

4 Conclusion

We introduced PROACTAGENT, an experience-driven lifelong learning framework enabling agents to proactively retrieve from a structured experience base and continually improve via interaction. EXPONEVO jointly refines typed memory and policies within a unified co-evolution loop. PROACTRL formulates retrieval as an explicit policy action, learning when and what to retrieve via paired-branch process rewards. Experiments on SciWorld, AlfWorld, and StuLife show PROACTAGENT consistently improves task success and interaction efficiency, with a 3B model rivaling 7B passively augmented baselines. These results highlight the effectiveness of proactive retrieval control for long-horizon online lifelong agents.

Limitations

PROACTAGENT relies on the prefix replayability assumption, which holds in the deterministic simulators used here but may not extend to environments with stochastic dynamics or irreversible state changes. The experience base grows without a capacity bound or learned eviction policy, so unbounded growth could degrade retrieval precision over substantially longer deployment horizons. Experience extraction depends on a separate, larger model, whose errors may propagate into stored entries. All three benchmarks are text-based with discrete actions, and generalization to multi-modal setting remains to be validated.

Ethical Considerations

This work focuses on improving the learning efficiency of language model agents in simulated environments (science experiments, household tasks, and campus navigation) and does not involve human subjects, private data, or real-world deployment. All benchmarks are publicly available and contain no personally identifiable information. The experience base is constructed from synthetic agent–environment interactions and stored locally without external dissemination. While our framework enhances autonomous decision-making capabilities, the controlled simulator settings pose minimal risk of societal harm. We acknowledge that deploying lifelong learning agents in open-ended real-world scenarios would require additional safeguards, including human oversight mechanisms and robust content filtering, to prevent unintended or harmful behaviors.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, and 9 others. 2022. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning*.

Yuxuan Cai, Yipeng Hao, Jie Zhou, Hang Yan, Zhikai Lei, Rui Zhen, Zhenhua Han, Yutao Yang, Junsong Li, Qianjun Pan, Tianyu Huai, Qin Chen, Xin Li, Kai Chen, Bo Zhang, Xipeng Qiu, and Liang He. 2025. Building self-evolving agents via experience-driven lifelong learning: A framework and benchmark. *arXiv preprint arXiv:2508.19005*.

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

Shangheng Du, Xiangchao Yan, Dengyang Jiang, Jiakang Yuan, Yusong Hu, Xin Li, Liang He, Bo Zhang, and Lei Bai. 2025. Automlgen: Navigating fine-grained optimization for coding agents. *arXiv preprint arXiv:2510.08511*.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Alex Graves, Greg Wayne, Malcolm Reynolds, and 1 others. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. 2024. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics*.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*.

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 *Advances in Neural Information Processing Systems*.

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

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 *Proceedings of the 36th*

Annual ACM Symposium on User Interface Software and Technology.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in NeurIPS*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in NeurIPS*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *ICLR*.

Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries. *arXiv preprint arXiv:2401.15391*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of ACL*.

Akshay Verma, Swapnil Gupta, Siddharth Pillai, Prateek Sircar, and Deepak Gupta. 2026. Reflectiverag: Rethinking adaptivity in retrieval-augmented generation.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. In *Advances in Neural Information Processing Systems*.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Cote, and Prithviraj Ammanabrolu. 2022. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *International Conference on Learning Representations*.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, and 1 others. 2025a. Agentgym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 27914–27961.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, and 1 others. 2025b. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning. *arXiv preprint arXiv:2509.08755*.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, and 1 others. 2026. Autoskill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

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

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19666–19674.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731.

Yuxin Zuo, Kaiyan Zhang, Shang Qu, Li Sheng, Xuekai Zhu, Biqing Qi, Youbang Sun, Ganqu Cui, Ning Ding, and Bowen Zhou. 2025. TTRL: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*.

A Related Work

Memory-augmented lifelong agents. External memory for neural models has progressed from differentiable controllers (Graves et al., 2014; Weston et al., 2015; Graves et al., 2016; Du et al., 2025) through retrieval-augmented generation (Lewis et al., 2020; Borgeaud et al., 2022) to memory systems designed for interactive agents, where memory must accumulate operational experience across episodes rather than index a static corpus. Recent agent memory systems manage persistent interactions through summarization or hierarchical architectures (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023; Chhikara et al., 2025) and maintain evolving stores that grow alongside continued interaction (Zhang et al., 2025; Yang et al., 2026). As these repositories scale, how retrieval is triggered becomes increasingly critical. Existing approaches differ primarily in triggering mechanism: static initialization (Park et al., 2023; Zhong et al., 2024) injects memory once at episode onset, providing startup context but leaving agents unable to adapt when conditions shift mid-episode; continuous retrieval (Zhang et al., 2025; Tang and Yang, 2024) queries at every step, increasing access at the cost of redundancy and context overload; and LLM-gated approaches (Verma et al., 2026; Zhang et al., 2026) employ a separate model to judge necessity, improving flexibility but introducing extra latency. Moreover, most systems store experience in a single undifferentiated repository (Park et al., 2023; Zhong et al., 2024), conflating factual evidence with behavioral guidance. More critically, retrieval across all three paradigms remains governed by fixed schedules, external rules, or separate modules rather than learned as an intrinsic agent capability, leaving agents unable to recognize when current knowledge is insufficient and proactively seek the most relevant experience.

Retrieval control for LLMs. A growing body of work studies how LLMs can control when and what to retrieve. In question answering, FLARE (Jiang et al., 2023) triggers retrieval based on generation confidence, Self-RAG (Asai et al., 2024) generates reflection tokens to assess retrieval necessity, and Adaptive-RAG (Jeong et al., 2024) routes queries by estimated complexity. In the agent setting, Toolformer and ReAct (Schick et al., 2023; Yao et al., 2023b) train models to interleave tool calls with reasoning, while IRCoT (Trivedi et al., 2023) further interleaves retrieval with chain-of-thought steps. These methods advance retrieval precision but share two limitations in the lifelong agent context. First, they operate over static or slowly changing knowledge bases (Jeong et al., 2024; Jiang et al., 2023), whereas a lifelong agent must selectively access a repository that grows continuously through its own interaction history. Second, and more critically, retrieval decisions in all these systems remain passive, governed by confidence thresholds, supervised classifiers, or generation-time heuristics that receive no outcome-level feedback on whether a particular retrieval action improved the downstream result (Trivedi et al., 2023). When retrieval introduces noise or displaces useful context, these methods cannot attribute the failure to a specific retrieval decision, because supervision does not isolate individual retrieval actions from the surrounding generation process.

Online evolution of interactive agents. A parallel line of research enables agents to improve autonomously through interaction. Within-episode methods such as iterative self-critique (Madaan et al., 2023) strengthen single-episode capabilities but do not accumulate cross-episode experience. Cross-episode approaches pursue continual improvement through two largely independent channels. Memory-centric methods accumulate textual experience: Reflexion (Shinn et al., 2023) stores verbal reflections from past failures, Voyager (Wang et al., 2023) builds reusable skill libraries, and ExpeL (Zhao et al., 2024) distills lessons from trial outcomes. Parameter-centric methods optimize the behavioral policy through online reinforcement learning (Xi et al., 2025a; Zuo et al., 2025). Recent lifelong benchmarks (Cai et al., 2025) further confirm that raw model capacity cannot substitute for sustained online knowledge accumulation. Despite the value of both memory and policy evolution, existing systems treat them in isolation: memory repositories grow without policy-level guidance on retrieval quality, and policy updates proceed without structured access to accumulated experience.

B Theoretical Analysis and Proofs

In this section, we provide a formal analysis of how the adaptive rollout and process reward mechanism in ProactRL aids credit assignment. Rather than claiming strict variance reduction or strict isolation of causal factors, we show that constructing a matched trajectory pair from a shared interaction

prefix yields a localized, counterfactual-correlated empirical signal at the branching step.

To formalize this, write $h_b \equiv h_{t_b}$, $a_b \equiv a_{t_b}$, and $\pi_{\text{env}}(\cdot \mid h_b) \equiv \pi(\cdot \mid h_b, \mathcal{A}_{\text{env}})$. We define the expected environment-level marginal utility of a specific retrieval action $\textsc{Retrieve}(q)$ at history h_b relative to the expected outcome of continuing without retrieval under the current behavior policy:

$$\begin{aligned} m(h_b, q) &\triangleq \mu_{\text{ret}}(h_b, q) - \mu_{\text{env}}(h_b), \\ \mu_{\text{ret}}(h_b, q) &\triangleq \mathbb{E}_{\tau\sim\pi}\left[R^{\text{env}} \mid h_b, a_b = \textsc{Retrieve}(q)\right], \\ \mu_{\text{env}}(h_b) &\triangleq \mathbb{E}_{\tau\sim\pi}\left[R^{\text{env}} \mid {}^{h_b,}_{a_b\sim\pi_{\text{env}}(\cdot\mid h_b)}\right]. \end{aligned} \tag{12}$$

The paired-branch mechanism does not provide an unbiased estimator of this quantity, but it aims to construct a single-sample empirical proxy that is locally aligned with it. To interpret the pairwise comparison, we require the following assumption regarding the underlying environment and the retrieval system.

Assumption 1 (Prefix Replayability and Consistency) For any branching step t_b, the environment state and the agent's internal history h_{t_b} can be exactly restored to explore alternative continuations. The transition dynamics $P(s' \mid s, a)$ and observation model $O(o \mid s', a)$ remain consistent across branched rollouts. Furthermore, the memory repository $\mathcal{D}$ and the retrieval function outputs are deterministic and fixed during these rollouts. For the matched no-retrieval branch, the branching action is sampled from $\pi(\cdot \mid h_{t_b}, \mathcal{A}_{\text{env}})$, and the subsequent suffix is generated from the same behavior policy under the replayed state.

Proposition 1 (Local Credit Assignment via Counterfactual Rollouts) Let h_{t_b} be the shared interaction history up to the branching step t_b. Consider a matched rollout pair sampled under the PROACTRL objective: τ^{ret} (indexed by i) with retrieval action $a_{t_b}^{(i)} = \textsc{Retrieve}(q_{t_b})$, and its counterfactual $\tau^{\text{no-ret}}$ (indexed by j) with an environment action $a_{t_b}^{(j)} \in \mathcal{A}_{\text{env}}$. Under Assumption 1, and under the local GRPO approximation in which $\rho \approx 1$ while clipping and the KL term are omitted, write $\pi_i^b \triangleq \pi_\theta(a_{t_b}^{(i)} \mid h_{t_b})$ and $\pi_j^b \triangleq \pi_\theta(a_{t_b}^{(j)} \mid h_{t_b})$. The branching-step contribution to the pairwise policy gradient admits the decomposition

$$\begin{aligned} g_{t_b} &\triangleq A_i \nabla_\theta \log \pi_i^b + A_j \nabla_\theta \log \pi_j^b \\ &= \frac{A_i + A_j}{2} \nabla_\theta \log\big(\pi_i^b \pi_j^b\big) \\ &\quad + \frac{A_i - A_j}{2} \nabla_\theta \log \frac{\pi_i^b}{\pi_j^b}. \end{aligned} \tag{13}$$

Hence the local update of the log-odds between the sampled retrieval action and the sampled environment action is governed by the empirical advantage gap $A_i - A_j$. Let $\sigma_R \triangleq \text{std}(\{R_k^{\text{traj}}\}_{k=1}^G) + \epsilon$. Moreover, for the matched no-retrieval branch, where $r_j^{\text{proc}} = 0$,

$$\begin{aligned} A_i - A_j = \sigma_R^{-1}\Big[&(R_i^{\text{env}} - R_j^{\text{env}}) + r_i^{\text{proc}} \\ &+ (r_i^{\text{eff}} - r_j^{\text{eff}})\Big], \end{aligned} \tag{14}$$

and therefore

$$\begin{aligned} A_i - A_j = \sigma_R^{-1}\Big[&\Delta_i - \lambda_T \frac{T_j - T_i}{\max(T_j, 1)} + r_i^{\text{proc}} \\ &+ (r_i^{\text{eff}} - r_j^{\text{eff}})\Big]. \end{aligned} \tag{15}$$

Thus, the branch comparison provides a shaped, noisy empirical proxy that combines the sampled environment margin, the explicit process reward, and the efficiency terms.

We analyze the unclipped surrogate gradient of the GRPO objective for a specific matched pair (i, j) sampled from a group of G rollouts for a task instance x. Assuming the policy remains close to the old behavior policy ($\rho \approx 1$) and temporarily omitting the KL divergence penalty for algebraic clarity, the empirical gradient contribution of this pair is:

$$\begin{aligned} \nabla_\theta \mathcal{J}_{\text{pair}}(\theta) \approx\ & A_i \nabla_\theta \log \pi_\theta(\tau^{(i)} \mid x) \\ &+ A_j \nabla_\theta \log \pi_\theta(\tau^{(j)} \mid x). \end{aligned} \tag{16}$$

Under Assumption 1, both trajectories strictly share the prefix $\tau_{<t_b}$. Let $u_t \triangleq \nabla_\theta \log \pi_\theta(a_t \mid h_t)$ denote the shared-prefix score term and $u_t^{(k)} \triangleq \nabla_\theta \log \pi_\theta(a_t^{(k)} \mid h_t^{(k)})$ denote the branch-specific score term. Decomposing the joint log-probability of each trajectory temporally yields three distinct update phases:

$$\begin{aligned} \nabla_\theta \mathcal{J}_{\text{pair}}(\theta) \approx\ & \sum_{t=1}^{t_b-1} u_t\big(A_i + A_j\big) && \text{(shared prefix)} \\ &+ A_i u_{t_b}^{(i)} && \text{(branch)} \\ &+ A_j u_{t_b}^{(j)} \\ &+ A_i \sum_{t>t_b} u_t^{(i)} && \text{(suffix)} \\ &+ A_j \sum_{t>t_b} u_t^{(j)}. \end{aligned} \tag{17}$$

Focus on the **Branching Step Update**. Writing $\pi_i \triangleq \pi_\theta(a_{t_b}^{(i)} \mid h_{t_b})$ and $\pi_j \triangleq \pi_\theta(a_{t_b}^{(j)} \mid h_{t_b})$, we have

$$\begin{aligned} g_{t_b} &= A_i \nabla_\theta \log \pi_i + A_j \nabla_\theta \log \pi_j \\ &= \frac{A_i + A_j}{2}\big(\nabla_\theta \log \pi_i + \nabla_\theta \log \pi_j\big) \\ &\quad + \frac{A_i - A_j}{2}\big(\nabla_\theta \log \pi_i - \nabla_\theta \log \pi_j\big) \\ &= \frac{A_i + A_j}{2}\nabla_\theta \log(\pi_i \pi_j) \\ &\quad + \frac{A_i - A_j}{2}\nabla_\theta \log \frac{\pi_i}{\pi_j}. \end{aligned} \tag{18}$$

Therefore, the coefficient multiplying the local log-odds direction $\nabla_\theta \log \frac{\pi_i}{\pi_j}$ is exactly $(A_i - A_j)/2$.

Next, because A_k is defined by group normalization, the sample mean cancels in the pairwise difference:

$$A_i - A_j = \frac{R_i^{\text{traj}} - R_j^{\text{traj}}}{\sigma_R}. \tag{19}$$

For the matched no-retrieval branch, $r_j^{\text{proc}} = 0$, so substituting $R^{\text{traj}} = R^{\text{env}} + r^{\text{proc}} + r^{\text{eff}}$ gives

$$\begin{aligned} A_i - A_j = \sigma_R^{-1}\Big[&(R_i^{\text{env}} - R_j^{\text{env}}) + r_i^{\text{proc}} \\ &+ (r_i^{\text{eff}} - r_j^{\text{eff}})\Big]. \end{aligned} \tag{20}$$

Substituting the rollout margin definition $\Delta_i = (R_i^{\text{env}} - R_j^{\text{env}}) + \lambda_T \frac{T_j - T_i}{\max(T_j, 1)}$, we obtain

$$\begin{aligned} A_i - A_j = \sigma_R^{-1}\Big[&\Delta_i - \lambda_T \frac{T_j - T_i}{\max(T_j, 1)} + r_i^{\text{proc}} \\ &+ (r_i^{\text{eff}} - r_j^{\text{eff}})\Big]. \end{aligned} \tag{21}$$

Under Assumption 1, R_j^{env} is a single-sample Monte Carlo proxy for the second expectation in $m(h_{t_b}, q_{t_b})$, while Δ_i further augments this shared-prefix comparison with the length regularizer. Therefore, the branching step anchors the retrieval action against a concrete counterfactual continuation and yields a shaped, outcome-correlated empirical signal. The result is local rather than global: it identifies the exact branch direction affected by the pairwise comparison, but it does not imply that $A_i - A_j$ is an unbiased estimator of $m(h_{t_b}, q_{t_b})$.

Discussion. Proposition 1 shows that the retrieval-versus-no-retrieval preference update at the branching step is controlled by $A_i - A_j$. Under the reward definition of PROACTRL, this coefficient is determined by the sampled environment margin, the process reward, and the efficiency-difference term, all scaled by the group-normalization factor.

- **Learning when to retrieve.** At the branching step, the pairwise gradient contains the term

  $$\frac{A_i - A_j}{2}\nabla_\theta \log \frac{\pi_\theta(\textsc{Retrieve}(q_{t_b}) \mid h_{t_b})}{\pi_\theta(a_{t_b}^{(j)} \mid h_{t_b})}.$$

  Therefore, if the retrieval branch yields a larger shaped return than its matched no-retrieval continuation, then $A_i - A_j > 0$ and the local update increases the relative log-probability of retrieving at history h_{t_b}. If the retrieval branch underperforms, then $A_i - A_j < 0$ and the update suppresses retrieval at that same decision point. Because the two branches share the identical prefix, this signal is attached to the necessity of retrieval at the current state rather than to unrelated earlier actions.

- **Learning what to retrieve.** The updated action is not a generic retrieval flag, but the instantiated action $\textsc{Retrieve}(q_{t_b})$. Consequently, the same gradient step also reinforces or suppresses the specific query content q_{t_b}. Queries that retrieve a useful experience $\mathcal{E}_{t_e}$ increase the downstream environment return, are more likely to induce a positive rollout margin Δ_i, and receive positive process-level reinforcement through r_i^{proc}. In contrast, vague or irrelevant queries tend to produce weak or negative margins and therefore lose relative probability. This is why PROACTRL learns the wording of the query jointly with the timing of retrieval.

Consequently, the shared-prefix paired-branch construction explains why PROACTRL can learn both when to retrieve and what to retrieve: the branch comparison supplies a local preference update for triggering retrieval, and that same update is tied to the concrete query instance that produced the observed downstream outcome.

C Algorithm Details

This section provides the complete algorithmic description of PROACTAGENT, complementing the

formal definitions in Section 2. Algorithm 1 outlines the full online co-evolution loop that interleaves policy optimization with experience accumulation. Algorithm 2 details the paired-branch reward computation that supplies step-level supervision for retrieval decisions.

Algorithm 1 Experience-Enhanced Online Evolution

Require: Base policy π_θ, empty experience base $\mathcal{D}$, task stream $\mathcal{X}$, group size G
Ensure: Evolved policy π_θ, populated experience base $\mathcal{D}$

Cold Start: Train π_θ via SFT on successful trajectories ▷ Appendix E.1
for each training iteration **do**
  Sample task batch $\{x_1, \ldots, x_B\}$ from $\mathcal{X}$
  for each task x in batch **do**
    for $i = 1, \ldots, G$ **do**
      Retrieve initial context $\mathcal{D}_0 \leftarrow \mathrm{TopK}(q_{\text{init}}, \mathcal{D}, K)$
      for $t = 1, \ldots, T$ **do**
        Sample action $a_t \sim \pi_\theta(\cdot \mid h_t, \mathcal{D}_{<t})$
        if $a_t = \textsc{Retrieve}(q_t)$ **then**
          $\mathcal{D}_t \leftarrow \mathrm{TopK}(q_t, \mathcal{D}, K)$; append to context
        else
          Execute a_t in environment; observe $o_{t+1}, r_t^{\text{env}}$
        end if
      end for
      Record trajectory $\tau^{(i)}$
    end for
    Construct paired branches via Algorithm 2
    Compute $\{R_i^{\text{traj}}\}_{i=1}^G$ and group-normalized advantages $\{A_i\}_{i=1}^G$
  end for
  Update π_θ via GRPO objective $\mathcal{J}_{\textsc{ProactRL}}(\theta)$ ▷ advantages in Eq. 11
  Extract typed entries from completed trajectories; update $\mathcal{D}$ ▷ Appendix D.3
end for

Interaction protocol. During each rollout, the agent processes the task prompt augmented with an initial retrieval context $\mathcal{D}_0$ drawn from $\mathcal{D}$. At each subsequent step, the policy produces either an environment action or a retrieval action $\textsc{Retrieve}(q_t)$. When a retrieval action is triggered, the experience base returns a type-balanced set of entries $\mathcal{D}_t$ (Equation 4 in Section 2.2), which is appended to the agent's context before the next decision. The policy treats retrieval as a regular action within the augmented action space $\mathcal{A} = \mathcal{A}_{\text{env}} \cup \mathcal{A}_{\text{ret}}$, requiring no separate gating module or confidence threshold.

Experience update protocol. After each training iteration, completed trajectories are processed by the extraction model (Appendix D.3) to produce typed entries. Factual and episodic memories are extracted from individual trajectories, success and failure skills are distilled from outcome-specific subsets, and comparative skills are generated from paired branches produced by Algorithm 2. All new

Algorithm 2 PROACTRL Paired-Branch Reward Computation

Require: Trajectory $\tau^{(i)}$ with retrieval at steps $\{t_1, \ldots, t_n\}$, policy π_θ, experience base $\mathcal{D}$
Ensure: Trajectory-level reward R_i^{traj}

if $n \geq 3$ **then**
  Select t_b uniformly from $\{t_2, \ldots, t_{n-1}\}$ ▷ Prefer interior steps
else
  Select t_b uniformly from $\{t_1, \ldots, t_n\}$
end if
Restore environment state at h_{t_b} via prefix replay ▷ Assumption 1
Suppress retrieval at t_b; sample $a_{t_b}^{(j)} \sim \pi_\theta(\cdot \mid h_{t_b}, \mathcal{A}_{\text{env}})$
Generate no-retrieval continuation $\tau^{(j)}$ from $(h_{t_b}, a_{t_b}^{(j)})$ under π_θ
Evaluate: $R_i^{\text{env}} \leftarrow \sum_t r_t^{\text{env}}(\tau^{(i)})$, $R_j^{\text{env}} \leftarrow \sum_t r_t^{\text{env}}(\tau^{(j)})$
Compute rollout margin: $\Delta_i \leftarrow (R_i^{\text{env}} - R_j^{\text{env}}) + \lambda_T \frac{T_j - T_i}{\max(T_j, 1)}$
if $\Delta_i > 0$ **then**
  $r_i^{\text{proc}} \leftarrow +\alpha$ ▷ Retrieval improved outcome
else if $\Delta_i < 0$ **then**
  $r_i^{\text{proc}} \leftarrow -\alpha$ ▷ Retrieval was redundant or harmful
else
  $r_i^{\text{proc}} \leftarrow 0$
end if
$r_i^{\text{eff}} \leftarrow -w_q \cdot \mathbf{1}[\text{repeat}_i] + \mathrm{clip}\left(w_t \frac{\bar{T}_{g(i)} - T_i}{\max(\bar{T}_{g(i)}, 1)}, -|w_t|, |w_t|\right)$
return $R_i^{\text{traj}} \leftarrow R_i^{\text{env}} + r_i^{\text{proc}} + r_i^{\text{eff}}$

entries are deduplicated against the existing repository before insertion. Priority scores of retrieved entries associated with successful trajectories are incremented, progressively surfacing high-utility experience.

D Experience Base Implementation

This section details the implementation of the structured experience base introduced in Section 2.2, whose role in the overall system is illustrated in Algorithm 1. We describe the repository schema, the retrieval backend, and the entry extraction procedure.

D.1 Repository Schema

The EXPERIENCE BASE maintains five typed entry families, organized into two complementary groups. The memory group ($\mathcal{M}$) provides evidence by recording what is true or what has happened, while the skill group ($\mathcal{S}$) provides behavioral guidance by encoding what to do or what to avoid. Table 5 summarizes the schema.

Each entry $r \in \mathcal{D}$ stores four fields: (1) a natural-language `when_to_use` description that specifies the triggering context, (2) a `content` field containing the actual knowledge or guidance, (3) an embedding vector $e(r)$ computed from the `when_to_use` field, and (4) a priority score $p(r)$ that is updated based on retrieval utility. The `when_to_use` field also serves as the exact-match deduplication key, preventing the repository from accumulating near-identical entries.

D.2 Retrieval Backend

The retrieval backend is built on the following components:

- **Encoder.** All `when_to_use` fields and runtime queries are embedded using `all-MiniLM-L6-v2` sentence embeddings (384 dimensions).
- **Vector index.** Embeddings are L2-normalized and stored in a FAISS inner-product index, which is equivalent to cosine similarity search over normalized vectors.
- **Scoring.** Given a query q_t, each candidate entry r is scored as $\text{score}(q_t, r) = \text{sim}(e(q_t), e(r)) + \lambda_p\, p(r)$, combining semantic similarity with the priority term (Section 2.2).
- **Type-balanced retrieval.** The retrieval budget K is divided equally across the five entry types, with one slot allocated to each by default. This prevents any single type from dominating the returned context and encourages complementary evidence.
- **Priority update.** Only entries that are actually retrieved during a trajectory receive priority updates. Specifically, when a trajectory succeeds, the priority of each retrieved entry is incremented by one. Entries that are stored but never retrieved receive no update. This mechanism progressively surfaces high-utility entries without requiring explicit supervision.
- **Deduplication.** Exact-match deduplication on the `when_to_use` field is enabled by default. The current implementation does not impose a fixed repository capacity cap or employ a learned eviction policy.

D.3 Entry Extraction and Maintenance

After each episode, completed trajectories are filtered to retain episodes with valid response tokens and then grouped by task instance. Within each group, a dedicated extraction model (Qwen3-32B) produces typed entries according to the following procedure:

1. **Factual and episodic memories** are extracted from individual trajectories via summarization. The extractor produces at most two entries of each type per trajectory, focusing on environment facts and trajectory-specific plans or constraints.
2. **Success and failure skills** are distilled from outcome-specific trajectory subsets. Each distiller returns one to three JSON-formatted entries that encode reusable strategies (from successes) or corrective rules (from failures).
3. **Comparative skills** are distilled from matched trajectory pairs. Paired A/B branches produced by PROACTRL are prioritized because they share the same task prefix and therefore expose the most localized contrastive signal. When such pairs are unavailable, the extractor falls back to outcome-ranked trajectory pairs from the same task group.

The complete prompts used for each extraction type are provided in Appendix F.

E Training Protocol

This section describes the full training pipeline, including the cold-start phase, the paired-branch

Table 5: The five entry families in the EXPERIENCE BASE. Memory entries ($\mathcal{M}$) supply factual evidence, while skill entries ($\mathcal{S}$) supply behavioral guidance. This separation ensures that a single query can retrieve complementary support from both groups.

Group	Entry Type	Content	Source
Memory ($\mathcal{M}$)	Factual ($\mathcal{M}^{\text{f}}$)	Environment facts, tool outputs, persistent states	Per-trajectory
	Episodic ($\mathcal{M}^{\text{e}}$)	Local plans, constraints, trajectory-specific reminders	Per-trajectory
Skill ($\mathcal{S}$)	Success ($\mathcal{S}^{+}$)	Reusable strategies from successful completions	Successful trajectories
	Failure ($\mathcal{S}^{-}$)	Error patterns and corrective rules	Failed trajectories
	Comparative ($\mathcal{S}^{\Delta}$)	Contrastive insights on why one continuation outperforms another	Paired A/B branches

Table 6: Core hyperparameters of the PROACTAGENT implementation. Annealing schedule entries are formatted as (no-retrieval fraction, learning rate warmup ratio).

Component	Value
Base model	Qwen2.5-7B-Instruct (main), Qwen2.5-3B-Instruct (scaling study)
Extraction model	Qwen3-32B (memory and experience extraction)
Learning rate	1e-6
Training epoch	3
Batch size	16
GRPO group size	8
Training budget	8 H100 GPUs for approximately 40 hours
Retriever embedding model	`all-MiniLM-L6-v2`
Vector index	FAISS inner-product index over normalized embeddings
Retrieval size	$k = 5$
Type quota	1 each for factual / episodic / success / failure / comparative
Paired-branch process reward (α)	0.5
Repeated-query penalty weight (w_q)	0.5
Step-efficiency weight (w_t)	0.25
Annealing schedule	calibration: $(0.5, 0.2)$, transition: $(0.25, 0.3)$, refinement: $(0.0, 0.5)$
No-retrieval branch	Retrieval suppressed

rollout procedure, and the retrieval annealing schedule. These stages correspond to the procedural flow outlined in Algorithm 1 and Algorithm 2. The hyperparameters used in all experiments are summarized in Table 6.

E.1 Pipeline Overview

The training pipeline proceeds through six sequential stages:

1. **Cold start.** The base policy is trained via supervised learning on successful trajectories to learn the interaction format, valid action syntax, and retrieval-tag conventions. This stage is critical for initializing the policy with sufficient tool-calling competence before reinforcement learning begins (as confirmed by the ablation in Section 3.3).

2. **Rollout sampling.** Multiple rollouts are sampled for each training prompt under the current policy. A portion of these rollouts is configured as no-retrieval trajectories through the `retrieval_enabled` switch, whose probability is annealed across training phases.

3. **Paired-branch construction.** When paired branching is active, the system identifies retrieval-trigger steps in retrieval-enabled rollouts, replays the corresponding prefixes, and creates matched no-retrieval branches (Section E.2).

4. **Reward computation.** The environment outcome is combined with the paired-branch process reward and the efficiency bonus to produce the PROACTRL trajectory-level reward (Section 2.3).

5. **Policy update.** The policy is updated using GRPO-style group normalization with PPO-style clipped surrogate optimization.

6. **Experience base update.** The experience base $\mathcal{D}$ is updated by extracting factual, episodic, success, failure, and comparative entries from the new trajectories (Appendix D.3).

This organization ensures that policy learning and memory growth remain tightly interleaved throughout training, realizing the co-evolution loop described in Section 2.2.

E.2 Paired-Branch Rollout Procedure

When a rollout τ contains one or more retrieval actions, the system constructs a matched no-retrieval branch as follows. If τ contains at least three retrieval actions, the branching step t_b is selected uniformly at random from the interior retrieval steps of τ, excluding the first and last retrieval actions. Boundary retrievals are excluded when possible because they are more susceptible to confounds from initialization effects (at the first step) or termination effects (at the last step), which could introduce noise into the process reward signal. If τ contains fewer than three retrieval actions, t_b is sampled uniformly from all available retrieval steps. The environment state at t_b is restored via prefix replay (Assumption 1), and the no-retrieval branch $\tau^{\text{no-ret}}$ is generated by suppressing the retrieval action at t_b and sampling subsequent actions from the current policy.

E.3 Retrieval Annealing

During training, the probability of disabling retrieval is annealed across three phases:

- **Calibration phase.** Corresponding branch paths were generated for all retrieval trajectories, ensuring abundant no-retrieval trajectories for paired A/B branch construction.
- **Transition phase.** The no-retrieval fraction decreases to 25%, maintaining some paired branches while allowing the retrieval-enabled policy to receive more optimization signal.
- **Refinement phase.** All rollouts are retrieval-enabled (0% disabled), concentrating optimization on the fully proactive policy.

The detailed hyperparameters are reported in Table 6.

F Experience Extraction Prompts

This section presents the complete prompts used by the extraction model (Qwen3-32B) to distill completed trajectories into typed experience entries for the EXPERIENCE BASE . Each prompt targets a specific entry family and is designed to produce structured JSON outputs with two fields: `when_to_use` (the triggering context for future retrieval) and `content` (the actual knowledge or guidance). All prompts enforce domain-agnostic language to ensure that extracted entries generalize across task types rather than overfitting to specific benchmark instances.

F.1 Memory Extraction Prompt

The following prompt extracts factual and episodic memories from individual trajectories. Factual memories capture verifiable environment facts, while episodic memories capture experiential insights and temporal constraints.

F.2 Success Skill Extraction Prompt

The following prompt distills reusable best practices from successful trajectories. It identifies the critical strategic decisions that enabled task completion and formulates them as forward-looking, imperative rules.

F.3 Failure Skill Extraction Prompt

The following prompt extracts corrective rules from failed trajectories. It focuses exclusively on the forward-looking solution rather than the diagnosis of past mistakes, ensuring that extracted entries are immediately actionable.

F.4 Comparative Skill Extraction Prompt

The following prompt distills contrastive insights from matched trajectory pairs. It compares a higher-quality trajectory against a lower-quality one sharing the same task, identifies the divergence point, and formulates a reusable rule that captures why the better continuation succeeded. Comparative entries produced from PROACTRL paired branches carry the strongest contrastive signal because the two trajectories share the same interaction prefix up to the branching step.

Memory Extraction Prompt

```
# Role
You are a Meta-Cognitive Memory Manager for an AI Agent. Your goal is to analyze the agent's recent execution trajectory to extract high-
  value, reusable memories.

# Input Data
## Execution Process
{execution_process}

## Result
{execution_result}

# Task
Review the provided [Execution Process] and [Result]. You must extract two distinct types of memories to assist in future tasks.
**Constraint: You must extract at least ONE memory in total across the two categories.**

1.  **Factual Memory (Objective Truths):**
    * *Definition:* Verifiable facts learned during execution.
    * *Examples:* Info/Preferences, Domain Knowledge, Tool/System Facts.
2.  **Episodic Memory (Experience, Reflection & Temporal Events):**
    * *Definition:* Insights derived from the flow of events, strategies, errors, OR **specific real-world time constraints/schedules**.
    * *Logic Examples:* "If I take step A, error B occurs," "Method X is faster than Y."
    * *Temporal Examples:* "User has a class at 8 AM on Mondays," "The deadline is this Friday."

# Constraints & Logic
1.  **Selection:** Extract a MAXIMUM of **2 Factual** and **2 Episodic** memories.
2.  **"when_to_use" Strategy:**
    * For *Factual*: Focus on the **Context Trigger** (e.g., "When using `pandas`...").
    * For *Episodic (Logic)*: Focus on the **Situational Trigger** (e.g., "When the dataset is empty...").
    * For *Episodic (Time)*: Focus on the **Temporal Trigger** (e.g., "When it is Monday morning," "At 8:00 AM").
3.  **Minimum Output:** Do not output an empty result. You must find the most valuable takeaway, even if minor.


# Output Format
Output a single valid JSON object strictly following this schema.

{{
  "factual_memories": [
    {{
      "when_to_use": "<Precise context trigger>",
      "memory": "<The objective fact>"
    }}
  ],
  "episodic_memories": [
    {{
      "when_to_use": "<Precise situational OR temporal trigger>",
      "memory": "<The insight, strategy, or scheduled event>"
    }}
  ]
}}
```

Success Skill Extraction Prompt

```
You are an expert Meta-Cognitive Success Analyst reviewing a successful trajectory of an LLM Agent.

Your goal is to analyze a "Success Trajectory" (a sequence of actions that successfully achieved a goal) and extract high-utility **Best
  Practice Memories** that will reinforce efficient strategies and ensure consistent success in similar future scenarios.
You must look past routine actions and identify the **critical strategic decisions** or **tool usage patterns** that streamlined the
  solution.

# INPUT DATA
## Original Query
{query}

## Step Sequence (Successful Trace)
{step_sequence}

## Context & Environment
{context}

## Outcome
This sequence ended in a {outcome} state.

# CRITICAL GUIDELINES FOR EXTRACTION

1. **FIELD: "when_to_use" (The Trigger Scope)**
   - **Definition:** Precisely define the context where this best practice applies. You MUST consider three dimensions:
     a. **Task Requirement:** What is the user specifically asking for? (e.g., "When the task requires verifying code execution results
  ...")
     b. **Specific Scenario:** What is the current state of the agent? (e.g., "When possessing a vague error message but no source code
  ...")
     c. **Observation:** What favorable condition appeared? (e.g., "When the search results return multiple high-confidence API documents
  ...")
   - **STRICT CONSTRAINT:**
     - Do NOT use specific dataset names (e.g., "In AlfWorld").
     - Do NOT use specific entity names (e.g., "When analyzing 'main.py'"). Instead, generalize it (e.g., "When analyzing the entry-point
  script").

2. **FIELD: "experience" (The Solution)**
   - **Definition:** A strict, actionable standard operating procedure.
   - **Constraint:** **PURELY FORWARD-LOOKING.** Do NOT explain why the approach was superior. Do NOT include phrases like "The agent
  succeeded because..." or "It is better to...".
   - **Structure:** Directly provide the step-by-step instruction on how to execute this efficiency. Use imperative language (e.g., "
  Always verify X before doing Y", "Use tool A in combination with parameter B").
   - **Requirement:** Focus on **Non-Trivial** insights. Do not extract generic advice like "Think carefully." Extract specific tool
  combinations or logic flows that proved highly effective.

# OUTPUT FORMAT
Output strictly a JSON object containing 1-3 memory objects. No markdown text outside the JSON.

```json
{{
  "skills": [
    {{
      "when_to_use": "When [Task Requirement / Specific Scenario / Observation] matches...",
      "experience": "[Actionable, step-by-step standard procedure to replicate success]."
    }}
  ]
}}
```
```

Failure Skill Extraction Prompt

```
You are an expert Meta-Cognitive Forensic Analyst reviewing a failed trajectory of an LLM Agent.

Your goal is to extract high-utility **Correction Memories** that will prevent this specific failure from recurring.
You must look past surface-level errors to identify the solution, ensuring the memory **only contains the correct path forward**, not a
    record of the mistake.

# INPUT DATA
## Original Query
{query}

## Step Sequence (Failed Trace)
{step_sequence}

## Context & Environment
{context}

## Outcome
This sequence ended in a {outcome} state.

# CRITICAL GUIDELINES FOR EXTRACTION

1. **FIELD: "when_to_use" (The Trigger Scope)**
   - **Definition:** Precisely define the context where this memory applies. You MUST consider three dimensions:
     a. **Task Requirement:** What is the user specifically asking for? (e.g., "When the task requires distinct counting of similar objects
    ...")
     b. **Specific Scenario:** What is the current state of the agent? (e.g., "When the retrieved context is too long to fit in one prompt
    ...")
     c. **Observation:** What feedback did the environment provide? (e.g., "When the tool execution returns a Timeout Error...")
   - **STRICT CONSTRAINT:**
     - Do NOT use specific dataset names (e.g., "In AlfWorld", "In HotpotQA").
     - Do NOT use specific entity names (e.g., "When looking for file 'data.csv'"). Instead, generalize it (e.g., "When looking for a target
     file that does not exist").

2. **FIELD: "experience" (The Solution)**
   - **Definition:** A strict, actionable instruction on how to handle this exact situation correctly.
   - **Constraint:** **PURELY FORWARD-LOOKING.** Do NOT explain why the previous attempt failed. Do NOT include diagnosis or "The agent
    failed because..." statements.
   - **Content:** Directly provide the optimized logic or step-by-step procedure to succeed immediately.

# OUTPUT FORMAT
Output strictly a JSON object containing 1-3 memory objects. No markdown text outside the JSON.

```json
{{
  "skills": [
    {{
      "when_to_use": "When [Task Requirement / Specific Scenario / Observation] matches...",
      "experience": "[Actionable, step-by-step strategy to succeed]."
    }}
  ]
}}
```
```

Comparative Skill Extraction Prompt

```
You are an expert Meta-Cognitive Analyst reviewing two trajectories for the exact same task.

Your goal is to extract high-utility **Contrastive Memories** by comparing a higher-quality trajectory against a lower-quality one and
  distilling a reusable, forward-looking rule.

# INPUT DATA
## Original Query
{query}

## High Score Trajectory
{high_steps}

## Low Score Trajectory
{low_steps}


# ANALYSIS INSTRUCTIONS

## Step 1: Identify the "Divergence Point"
- Compare the two trajectories step-by-step.
- Acknowledge that the low-quality trajectory may have started correctly. Do not evaluate valid initial steps as failures.
- Locate the earliest step where the higher-quality trajectory makes a better choice that changes the outcome or improves progress.

## Step 2: Extract the Causal Mechanism
- Determine the concrete, actionable reason the high-quality choice works better (e.g., checks preconditions, uses the right tool
  parameter, avoids a loop).
- Avoid retrospective narration; focus on what should be done next time.

## Step 3: Formulate a Reusable Rule
- Convert the insight into an imperative rule the agent can follow in future similar situations.
- Ensure the advice is **actionable**. It should tell the agent *what to do* in future similar states, not just describe what happened in
  the past.

# CONSTRAINTS FOR OUTPUT FIELDS

## 1) `when_to_use`
- **DO:** Describe the abstract scenario, the nature of the observation, or the specific type of logical constraint (e.g., "When the user
  asks for a file modification but the file path is ambiguous...").
- **DO NOT:** Mention specific dataset names, benchmark titles, or proper nouns of the evaluation framework (e.g., NEVER use "AlfWorld", "
  HotPotQA", "Mind2Web", etc.).

## 2) `experience`
- This string must be dense and instructional.
- This string must be a **Direct Actionable Command** only.
- **NEGATIVE CONSTRAINT:**
  - DO NOT explain "Agent A succeeded because..."
  - DO NOT mention "The failed agent did X..."
  - DO NOT include retrospective analysis or justification.
- **TONE:** Imperative, authoritative, and concise.
- **Format:** A concrete rule for future tasks like "Always [Action] to [ensure Result/avoid Error]." or "If [Condition], then [Action]."

# OUTPUT FORMAT
Output strictly a JSON object containing 1-3 memory objects. No markdown text outside the JSON.

```json
{{
  "skills": [
    {{
      "when_to_use": "Abstract description of the triggering state...",
      "experience": "Imperative rule or actionable advice for the next action. (e.g., 'Always verify the file path with ls before
  attempting to read.')"
    }}
  ]
}}
```
```