CoEvo-RL: Co-Evolving Policies and Experiential Memories for LLM Agents

Anonymous ACL submission

Abstract

Experience-augmented LLM agents can learn from both parametric reinforcement learning and explicit trajectory-level memories, but existing methods often optimize the policy and the experience bank in isolation. This separation creates a mismatch: as the policy improves, earlier experiences may be internalized and lose marginal value, while a stale or independently updated bank may fail to provide the knowledge the current policy actually needs. We propose CoEvo-RL, a policy-experience co-evolution framework that updates a GRPO-trained policy and an experiential memory bank concurrently. Across AppWorld, τ^2-Bench, and LifelongAgentBench, CoEvo-RL outperforms policy-only RL, experience-only learning, and other experience-augmented baselines, improving average success rate from 0.781 to 0.820. Further analyses show faster learning and persistent gains from refreshed experience, indicating that co-evolution improves both final performance and training dynamics.

1 Introduction

LLM agents need learning mechanisms beyond a single policy update (Zhang et al., 2026b). Interactive LLM agents must repeatedly perceive an environment, select actions, observe feedback, and improve over long horizons. Reinforcement learning can improve the parametric policy from task rewards, but many agent failures are also episodic and reusable: a failed trajectory may reveal a missing precondition, a tool-use convention, or a domain-specific action pattern that should be preserved explicitly. Recent experience-augmented agents therefore store reflections, skills, or distilled lessons from past trajectories in an external experiential memory bank and retrieve them as context for future decisions (Shinn et al., 2023; Wang et al., 2023; Zhao et al., 2024).

Optimizing policy and experience in isolation is limiting. Existing learning pipelines typically emphasize one side of this system. A policy-only agent can improve its model parameters from environment rewards, but it lacks a persistent non-parametric store for preserving explicit lessons about failures, partial successes, and task-specific decision conditions. An experience-only agent can accumulate reusable entries, but a frozen policy cannot internalize recurring action patterns or learn when and how to exploit retrieved experience. This mismatch becomes especially important in sparse-reward agent benchmarks, where early rollouts are often uniformly unsuccessful and provide weak advantage signals for policy learning. Conversely, useful experience entries are hard to write before the policy has discovered informative trajectories.

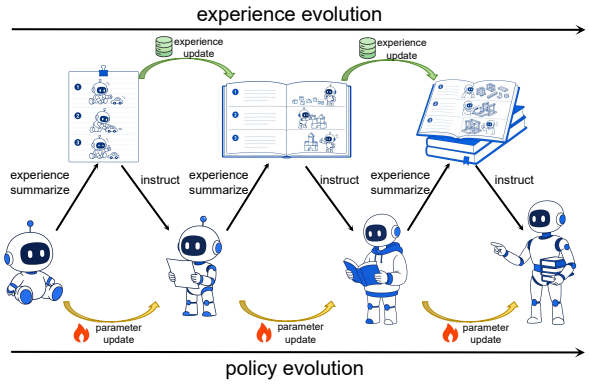


Figure 1: The key idea of CoEvo-RL. A stronger parametric policy can discover more useful experiences, while better experiences guide the policy to complete tasks more effectively.

In this paper, we propose CoEvo-RL, which moves beyond isolated policy or experience optimization by jointly learning a parametric policy and a non-parametric experiential memory bank. As shown in Figure 1, the core idea is that policy improvement and experience construction should reinforce each other: a stronger policy produces more informative trajectories for experience extraction, while better experience entries reshape future

rollouts and provide more useful learning signals for policy optimization.

CoEvo-RL alternates between two update steps over a shared stream of agent trajectories. In the policy step, the agent retrieves relevant entries from the current experiential memory bank and performs experience-conditioned rollouts; the resulting rewards are then used to update the parametric policy with GRPO. In the experience step, the same trajectories are converted into candidate experience entries that summarize reusable strategies, task constraints, or failure patterns. Candidate entries are committed only when replay indicates that they improve over the current experiential memory bank. This creates a closed training loop: the experiential memory bank shapes the rollout distribution seen by the policy, while the improved policy produces new trajectories from which better entries can be extracted.

We empirically evaluate CoEvo-RL on three interactive agent benchmarks: AppWorld, τ^2-Bench, and LifelongAgentBench. Compared with policy-only RL, experience-only learning, and prior policy-experience training, CoEvo-RL achieves the best success rate across all evaluated settings. Beyond final performance, our ablations and training dynamics show why co-evolution matters: an evolving bank continues to expose policy-relevant knowledge after earlier entries have been partly internalized.

Our main contributions are:

- We propose CoEvo-RL, a policy–experience co-evolving pipeline that alternates between policy updates and experience extraction, converting improved agent behaviors into reusable entries while using accumulated experience to further guide subsequent policy learning.
- We evaluate CoEvo-RL on AppWorld, τ^2-Bench, and LifelongAgentBench, comparing against GRPO, experience-only pipelines (MemRL), and existing policy-experience baselines (SkillRL). The results demonstrate that our method achieves more effective optimization and consistently outperforms competing approaches.
- We provide ablations and training-dynamics analyses showing that policy learning and experience evolution are mutually reinforcing, and that continually refreshed experience are necessary as earlier entries are gradually internalized by the policy.

2 Related Work

2.1 Reinforcement Learning for LLM Agents

Reinforcement learning has become a central approach for improving LLM agents on interactive tasks, where agents must learn effective action policies from sparse task feedback. Recent methods often adopt critic-free group-relative policy optimization (GRPO) to improve training stability under verifiable or outcome-based rewards. Furthermore, GiGPO extends this idea to agent tasks by incorporating anchor-state step-level advantages in addition to episode-level feedback (Feng et al., 2025); and RLVMR combines outcome rewards with verifiable meta-reasoning rewards to improve generalization on embodied text-based environments (Zhang et al., 2025). These methods demonstrate the effectiveness of RL for optimizing the parametric policy of LLM agents. However, they typically treat experiential memory as fixed components rather than optimization targets. In contrast, our work targets a different direction: we treat both the parametric policy and experiential memory as learnable components, and optimize them jointly under task reward.

2.2 Memory Systems for LLM Agents

Experiential memory has emerged as an important direction for LLM agents, aiming to distill reusable knowledge from the agent's own past trajectories and reuse it on future tasks. Most existing systems build such memories on top of a frozen policy. For example, Reflexion stores verbal self-critique from failed trajectories (Shinn et al., 2023), Voyager accumulates executable skills (Wang et al., 2023), ExpeL distills cross-task natural-language insights (Zhao et al., 2024), A-Mem organizes interconnected memory notes (Xu et al., 2025), and Mem0 consolidates long interaction histories into hybrid vector-graph memory (Chhikara et al., 2025). Beyond these frozen-policy memory pipelines, a more recent line makes memory construction itself learnable, such as MemSkill (Zhang et al., 2026a), EvoSkill (Alzubi et al., 2026), and AtomMem (Huo et al., 2026), which learn how to update or organize reusable memories. However, these methods still mainly optimize the memory mechanism itself, while the task policy is either frozen or not jointly evolved with the memory bank. In contrast, our CoEvo-RL framework treats policy learning and memory optimization as a coupled process: reward-driven policy updates produce improved

trajectories for memory extraction, and the resulting memories further support subsequent policy learning.

3 Method

We present CoEvo-RL, a framework that jointly evolves a parametric policy and a non-parametric experience bank from the same stream of agent trajectories. The policy learns from experience-conditioned rollouts, while new experience entries are distilled from trajectories produced by stronger policies. Figure 2 gives an overview of this shared training loop and its two coupled update paths.

3.1 Preliminaries

We consider an LLM agent interacting with an environment on tasks sampled from a distribution $\mathcal{D}$. A task is specified by a natural-language instruction $x \sim \mathcal{D}$. During execution, the agent produces a multi-turn trajectory $\tau = (a_1, o_1, \ldots, a_T, o_T)$ and receives a terminal reward $R(\tau) \in [0, 1]$ from the environment.

At training step t, the agent state consists of two learnable components:

$$S_t = (\theta_t, E_t), \tag{1}$$

where θ_t parameterizes the policy π_{θ_t} and $E_t = \{e_1, e_2, \ldots\}$ is an experience bank of distilled trajectory-level knowledge. Each experience entry stores a reusable lesson from previous trajectories, including what strategy to apply and under what task condition it is useful. Given a task x, a query $q(x)$ is derived from the task instruction and scored against each experience entry with a scoring function s:

$$C_t(x) = \mathrm{TopK}_{e \in E_t}\big(s(q(x), e), k\big). \tag{2}$$

where k is the retrieval budget and $C_t(x)$ is the compact experience block inserted into the rollout prompt. In our implementation, s is instantiated with structured TF-IDF similarity. The policy then acts under the experience-conditioned context:

$$\tau \sim \pi_{\theta_t}(\cdot \mid x, C_t(x)). \tag{3}$$

CoEvo-RL updates both components of S_t through a co-evolutionary loop: policy optimization changes how trajectories are generated, and experience evolution changes what reusable knowledge conditions future rollouts. The two updates are coupled through a shared rollout stream.

3.2 Policy-Experience Co-Evolution

CoEvo-RL alternates between policy and experience updates on a shared rollout stream. At each policy update, the current bank E_t conditions rollouts through retrieved experiences, providing task-specific constraints, failure warnings, and action recipes; the resulting rewards update θ_t with policy optimization.

Periodically, recent trajectories from the updated policy are distilled into candidate experience entries. Candidates are admitted only when they improve future rollouts under the validation procedure in Section 3.4. Thus, retrieved experience shapes the policy's training distribution, while the improving policy supplies new evidence for experience evolution.

The resulting feedback loop can be summarized as

$$E_t \to \{\tau_t, R(\tau_t)\} \to \theta_{t+1} \to \{\tau_{t+1}\} \to E_{t+1}. \tag{4}$$

This loop is not a one-time experience initialization step. It runs throughout training, so the policy and experience bank continually adapt to each other's current capabilities. Algorithm 1 summarizes the resulting training procedure.

3.3 Experience-Conditioned Policy Optimization

For each task x, we retrieve $C_t(x)$ from the current experience bank and sample a group of N trajectories under the same retrieved context:

$$\{\tau^{(i)}\}_{i=1}^{N} \sim \pi_{\theta_t}(\cdot \mid x, C_t(x)). \tag{5}$$

We optimize the policy with GRPO (Shao et al., 2024). For each trajectory in the group, the advantage is computed from relative task outcomes:

$$A^{(i)} = \frac{R(\tau^{(i)}) - \mathrm{mean}_j R(\tau^{(j)})}{\mathrm{std}_j R(\tau^{(j)}) + \varepsilon}. \tag{6}$$

The normalized advantages are used in the standard clipped GRPO objective with KL regularization to a reference policy.

Since all trajectories in a group share the same $C_t(x)$, their relative advantages compare policy behavior under a fixed experience context. This reduces retrieval-induced variance and leaves experience quality to the experience-evolution step.

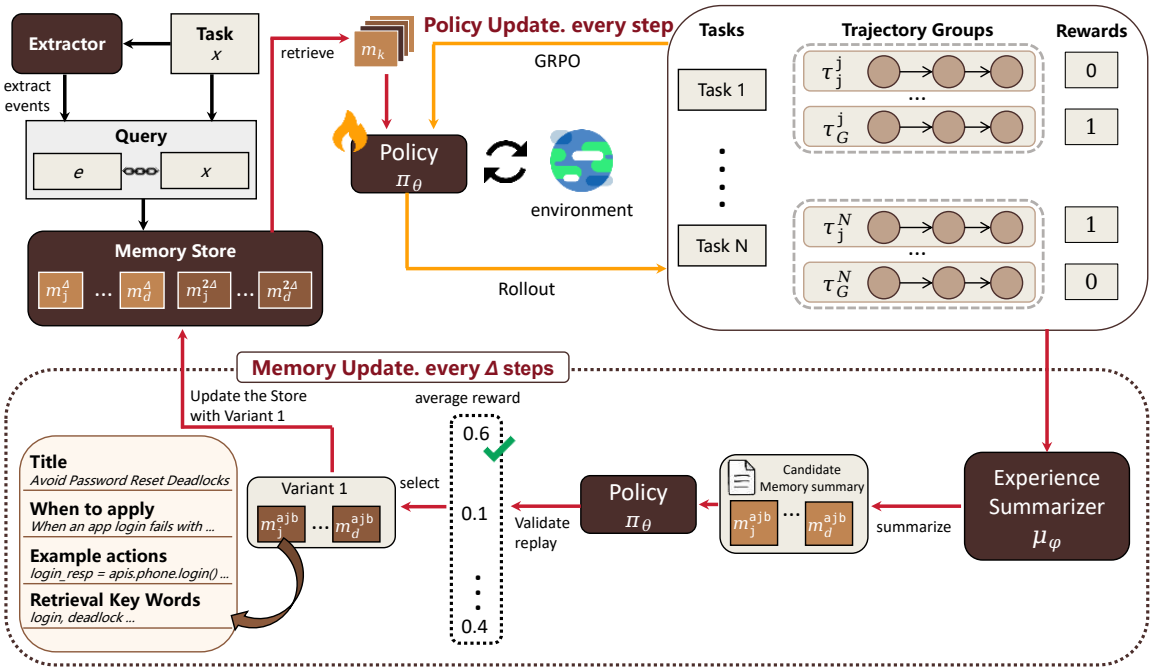


Figure 2: Overview of CoEvo-RL, where experience-conditioned rollouts update the policy and the resulting trajectories are distilled into validated experience entries.

3.4 Experience Evolution

Every Δ policy-update steps, we update the experience bank from recent trajectories. The experience updater μ_ϕ receives a small trajectory buffer together with the current bank E_t and distills candidate entries that capture reusable experience. It prioritizes tasks with both successful and failed rollouts, because these contrastive pairs expose the pivotal decision point under the same task context. When such pairs are unavailable, it falls back to success-only clusters for reusable action recipes and failure-only clusters for error causes and corrections.

Each candidate experience entry is written in a structured natural-language form. Besides a reusable strategy and its applicability condition, an entry also includes an example action that grounds the strategy in concrete behavior, as well as retrieval keywords used by the structured retriever. This schema keeps entries readable to the policy while giving the retriever explicit cues for matching future tasks to the right experience.

Because trajectory-derived entries can be noisy, we validate them by replay before committing them to the live bank. For a candidate set $\mathcal{V}$ of new entries, we form $E' = E_t \cup \mathcal{V}$ and compare its mean reward $\hat{R}(E')$ with the current bank $\hat{R}(E_t)$ on the tasks that produced the candidates. The next experience bank is

$$E_{t+1} = \begin{cases} E', & \text{if } \hat{R}(E') > \hat{R}(E_t) + \delta, \\ E_t, & \text{otherwise.} \end{cases} \tag{7}$$

Thus, the evolving experience bank is tied to task performance instead of treating experience writing as a separate objective.

This design lets the two components co-evolve: few early successes or revealing failures can provide experiences that guide later exploration, and as the policy improves, its trajectories provide stronger evidence for updating the experience bank.

4 Experiments

Our experiments evaluate whether policy learning and experience construction are more effective when they evolve together rather than in isolation. We organize the evaluation around four questions: (1) Does co-evolution improve over policy-only, experience-only, and existing policy-experience baselines? (2) Which retrieval mechanism best selects useful experience for the current policy? (3) Does the timing of experience evolution matter? (4) Do the learning dynamics show evidence that experience is progressively internalized while the bank keeps exposing new policy-relevant knowledge?

Algorithm 1: Policy-Experience Co-Evolution in CoEvo-RL

Input: Initial policy π_{θ_0}, bank E_0, updater μ_ϕ, interval Δ, budget k

Output: Optimized policy π_θ and evolved bank E

for $t = 0, 1, 2, \ldots$ **do**
  Sample a task batch $\mathcal{B}_t \sim \mathcal{D}$;
  foreach $x \in \mathcal{B}_t$ **do**
    $C_t(x) \leftarrow \mathrm{TopK}_{e \in E_t}(s(q(x), e), k)$;
    $\{\tau_x^{(i)}\}_{i=1}^{N} \sim \pi_{\theta_t}(\cdot \mid x, C_t(x))$;
  $\theta_{t+1} \leftarrow \mathrm{GRPO}\big(\theta_t, \{(\tau, R(\tau))\}\big)$;
  if $t \bmod \Delta = 0$ **then**
    $\mathcal{V} \leftarrow \mu_\phi(\{\tau\}, E_t)$;
    $E^+ \leftarrow E_t \cup \mathcal{V}$;
    Estimate $\hat{R}(E^+)$ and $\hat{R}(E_t)$ by replay;
    if $\hat{R}(E^+) > \hat{R}(E_t) + \delta$ **then**
      $E_{t+1} \leftarrow E^+$;
    else
      $E_{t+1} \leftarrow E_t$;
  else
    $E_{t+1} \leftarrow E_t$;

4.1 Experimental Setup

Benchmarks. We evaluate on three interactive agent benchmarks. **AppWorld** (Trivedi et al., 2024) evaluates interactive coding agents in a simulated world of everyday applications. **τ^2-Bench** (Yao et al., 2024) evaluates agents in customer-service workflows that require following domain policies; we use its *telecom* domain following the Artificial Analysis intelligence-benchmarking methodology[1]. **LifelongAgentBench** (Zheng et al., 2025) (LLB) provides skill-grounded interactive tasks in three environments: database, operating system, and knowledge graph. Following MemRL (Zhang et al., 2026b), we use the DBBench database domain and the OSInteraction operating-system domain with the same train/test split as MemRL.

Baselines and Metrics. We compare our method against three method families: **Policy-only** (GRPO with no experience module), **Experience-only** (MemRL (Zhang et al., 2026b)), and **Policy & Experience** (SkillRL (Xia et al., 2026)), which update both policy and experience during training. All benchmarks are evaluated by task-level *success rate*, using the official success criterion of each benchmark.

Implementation Details. In our experiments, all methods share the same policy backbone (Qwen3-8B (Yang et al., 2025)) and are trained with the GRPO update of Eq. (6) (N=8 rollouts per task, batch of 16 tasks, learning rate 10^{-6}, low-variance KL coefficient 10^{-3}) on a single 4×A100-80G node with asynchronous SGLang rollout. The experience pipeline of §3.4 runs every Δ steps (Δ=20 on AppWorld/LLB, Δ=15 on τ^2-Bench) with retrieval top-k=8; the default retriever uses the structured TF-IDF scoring of Eq. (2) and the experience updater μ_ϕ is a closed-source LLM API.

4.2 Main Results

Table 1 summarizes the main comparison across four task settings that cover tool use, agentic workflow following, operating-system interaction, and database operation. CoEvo-RL achieves the best result on every benchmark and improves the average success rate from 0.781 with SkillRL to 0.820. We draw three observations.

First, policy optimization and experience evolution are complementary. Compared with the policy-only GRPO baseline, CoEvo-RL improves the average success rate by 8.0 points (0.740 → 0.820). In contrast, experience-only methods are weaker than policy optimization on average: MemRL reaches 0.606, and *Ours-Exp* reaches 0.571 under a frozen policy. External experience therefore complements, rather than replaces, policy optimization.

Second, reward-validated experience evolution improves over prior policy-experience training. SkillRL also augments reinforcement learning with evolving skills distilled from the current policy's rollouts, making it a strong comparison for coupled policy-experience learning. Yet CoEvo-RL improves over SkillRL on all four settings: +5.4 points on AppWorld, +4.5 points on τ^2-Telecom, +2.0 points on LLB-OS, and +3.6 points on LLB-DB. This consistent gain suggests that effective policy-experience co-evolution requires not only shared on-policy rollouts, but also a shared optimization objective, with environment reward guiding which experiences are retained.

Third, the advantage is largest on environments where explicit reusable constraints matter. Com-

[1] https://artificialanalysis.ai/methodology/intelligence-benchmarking

Family	Method	AppWorld	τ^2-Telecom	LLB-OS	LLB-DB	Avg.
None	`Qwen3-8B` (frozen)	0.211	0.225	0.493	0.791	0.430
Policy-only	GRPO (Shao et al., 2024)	0.478	0.900	0.667	0.914	0.740
Experience-only	MemRL (Zhang et al., 2026b)	0.481	0.475	0.626	0.840	0.606
	Ours-Exp (frozen policy)	0.454	0.350	0.653	0.827	0.571
Policy & Experience	SkillRL (Xia et al., 2026)	0.554	0.930	0.700	0.942	0.781
	Ours	0.608	0.975	0.720	0.978	0.820

Table 1: Main success-rate results across AppWorld, τ^2-Bench, and two LifelongAgentBench (LLB) settings. The average is computed over the four task settings.

pared with LLB-OS and LLB-DB, whose domains are already familiar to the base model, AppWorld and τ^2-Bench telecom impose more environment-specific interaction rules, hidden APIs, and procedural manuals. On difficult tasks in these environments, policy rollouts alone rarely produce successful trajectories under sparse terminal rewards, so RL receives too little positive signal to reliably learn the required rules from scratch. These are exactly the settings where CoEvo-RL gains most over SkillRL and GRPO, suggesting that co-evolved experience is most useful when the base policy is not familiar with the environment.

4.3 Ablation Studies

We next isolate the design choices in CoEvo-RL: how entries are retrieved, how often the experience bank is updated, how policy and experience updates are scheduled, and how many entries should be placed in the context. Unless otherwise specified, ablations use the same backbone and training recipe as §4.1.

4.3.1 Effectiveness of Retrieval Methods

We evaluate different retrievers in Eq. (2) using 47 query–target pairs, where each experience's recorded source task serves as the query and the experience itself is the ground-truth target. We report Hit@1, Hit@3, MRR, and downstream success rate.

Table 2 shows that TF-IDF achieves the best retrieval quality and downstream performance, with 88.6% Hit@1, 95.5% Hit@3, 0.871 MRR, and a 72.0% success rate. In contrast, embedding similarity has the lowest Hit@1 (65.9%), MRR (0.754), and success rate (65.3%), indicating that broad semantic similarity alone misses task-specific operational cues such as command names, API objects, file paths, and retrieval keys. We therefore use TF-IDF as the default retriever in subsequent experiments.

Retriever	Hit@1	Hit@3	MRR	SR (%)
Embedding similarity	65.9	81.8	0.754	65.3
TF-IDF (Ours)	**88.6**	**95.5**	**0.871**	**72.0**
BM25	81.8	93.2	0.849	68.0
TF-IDF + Embedding	79.5	89.7	0.861	68.7
BM25 + Embedding	75.0	87.4	0.865	67.3

Table 2: Comparison of retrievers on LLB-OSInteraction. A retrieval is judged correct if the retrieved entry is distilled from the same task as the current task. **Success Rate** is the downstream success rate on the test split under each retriever.

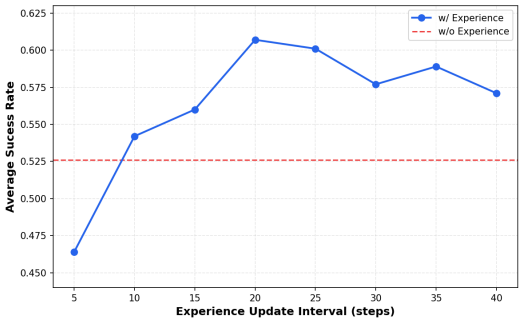


Figure 3: Effect of the experience update interval Δ on AppWorld. One AppWorld epoch corresponds to 5 training steps; we sweep $\Delta \in \{5, 10, 15, 20, 25, 30, 35, 40\}$ steps (i.e., 1–8 epochs). The y-axis is task success rate at convergence; all other hyper-parameters follow §4.1.

4.3.2 Sensitivity to Update Interval

The experience update interval controls a trade-off between freshness and stability. Updating too frequently can commit entries from noisy early rollouts, while updating too sparsely leaves the policy with stale experience. We therefore sweep Δ from 5 to 40 steps on AppWorld, where 5 steps correspond to one training epoch.

Figure 3 shows a clear non-monotonic relation-

Training schedule	Success Rate
P→E	0.68
E→P	0.70
Joint Co-Evolution (Ours)	0.72

Table 3: Effect of the policy-experience training schedule on LIFELONGAGENTBENCH OSInteraction.

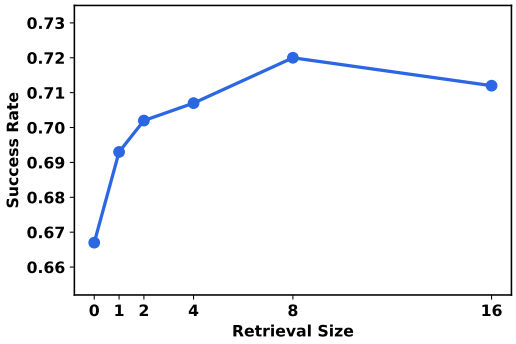


Figure 4: Effect of the budget of retrieved experience entries on LLB-OSInteraction. x-axis: number of retrieved entries; y-axis: success rate.

ship between update interval and final performance. The success rate rises from 0.464 at $\Delta{=}5$ to a peak of 0.607 at $\Delta{=}20$, then gradually drops as updates become less frequent. Updating every epoch is even worse than the GRPO-without-experience reference (0.526). A likely explanation is twofold. First, very early rollouts yield immature entries that are not yet reliably reusable. Second, if the bank is refreshed too often, the retrieved context distribution seen by the policy changes too quickly across training, increasing non-stability in the learning signal and making credit assignment noisier. At the other extreme, overly infrequent updates leave the bank stale. The best interval therefore balances freshness and stability, allowing useful experience to remain available long enough for the policy to benefit from it.

4.3.3 Effect of Co-Evolution Schedule

We compare three policy-experience training schedules on LIFELONGAGENTBENCH OSInteraction. In P→E, the policy is first trained without experience, then frozen while the experience bank evolves. In E→P, the experience bank is first grown under a frozen policy, then fixed while the policy is trained. These staged schedules isolate the two learning channels, whereas our joint co-evolution schedule alternates policy and experience updates every Δ steps under the same total compute budget. This comparison tests whether policy and experience benefit from adapting to each other during training.

As shown in Table 3, joint co-evolution achieves the highest success rate (0.72), outperforming both staged alternatives. The results suggest that staged schedules may introduce a mismatch between one component and the distribution induced by the other, whereas alternating updates better maintain alignment between retrieved entries and the current policy's failure modes.

4.3.4 Effect of Retrieved Experience Num.

The retrieval budget controls the amount of external experience available at inference time, trading broader coverage against longer contexts and potentially distracting redundancy. To examine this effect, we evaluate CoEvo-RL on LLB-OSInteraction with varying entry budgets while keeping the retriever and policy fixed.

Figure 4 shows that performance improves as the budget increases from 0 to 8 entries ($0.667 \rightarrow 0.720$), then slightly drops at 16 entries (0.712). This suggests that retrieval benefits come from a small set of high-precision entries rather than from exposing the policy to as much external text as possible. We therefore use a relatively compact retrieval budget in the default setting.

5 Discussion

We further analyze how co-evolving experience shapes CoEvo-RL's training dynamics.

5.1 Convergence and Sample Efficiency

Final success rates summarize the endpoint of training, but they do not characterize how quickly performance improves. We therefore use learning curves to examine whether co-evolving experience changes the learning dynamics. If validated entries provide reusable task constraints during training, CoEvo-RL should begin to improve earlier than policy-only optimization rather than matching it until the final stage.

Figure 5 shows this pattern across all three benchmarks. CoEvo-RL rises earlier than the policy-only baseline and generally maintains a higher success rate throughout training, with especially clear gains before convergence on AppWorld and τ^2-Bench telecom. This suggests that experience updates improve sample efficiency by exposing the policy to validated constraints and strategies.

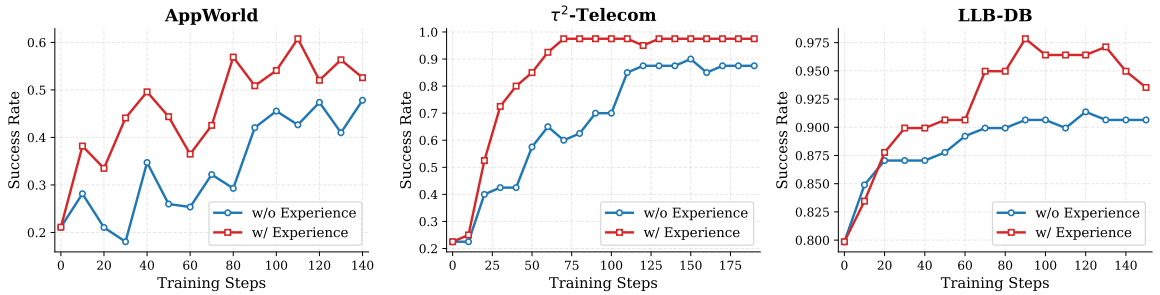
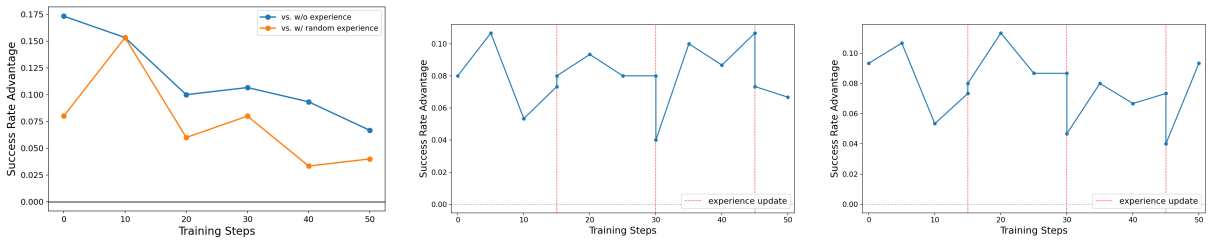


Figure 5: Learning curves on AppWorld, τ^2-Bench telecom, and LLB-DB. x-axis: training step; y-axis: success rate. We compare *Policy-only* (GRPO without retrieved experience) and *CoEvo-RL* (joint policy-experience co-evolution).



(a) Frozen experience bank. (b) Co-evolving bank (vs. no experience). (c) Co-evolving bank (vs. random experience).

Figure 6: Experience advantage dynamics on LIFELONGAGENTBENCH OSInteraction. x-axis: training step; y-axis: success-rate advantage of top-1 retrieved experience over each baseline. **(a)** Initial experience bank is frozen. **(b–c)** Experience is updated every 15 steps.

5.2 Experience Internalization Dynamics

We study whether the policy progressively internalizes retrieved experience on LIFELONGAGENTBENCH OSInteraction. At each checkpoint θ_t, we compare top-1 retrieved experience with two baselines that either remove experience or inject one random entry from the same bank. We use top-1 rather than the top-k setting in the main experiments to sharpen credit assignment, since a larger k makes the random baseline more likely to include a useful entry. Figure 6 reports the success-rate advantage under a frozen initial bank and a co-evolving bank updated every 15 steps.

With a frozen bank, the advantage steadily shrinks, falling from about 0.175 to 0.075 against no experience and from an early peak of about 0.15 to roughly 0.04 against random experience. This decline suggests that the policy gradually absorbs the behavior encoded in the initial entries. With a co-evolving bank, the advantage remains recurrent instead of vanishing, staying around 0.04–0.11 against no experience and 0.04–0.12 against random experience. It also dips just after updates, most visibly near step 30, where the advantage falls to about 0.04–0.05 before recovering. These dips suggest that newly committed entries capture errors and decisions that the current policy has not yet learned to exploit, so their value appears after several policy updates. Thus, policy-experience co-evolution is necessary because internalized entries lose marginal value, while the bank must keep introducing new experience the current policy has not yet absorbed.

6 Conclusion

We study reinforcement learning for experience-augmented LLM agents from a policy-experience co-evolution perspective. COEVO-RL couples GRPO-based policy optimization with non-parametric experience construction through shared rollouts, where structured TF-IDF retrieval provides relevant experience and periodic replay validation commits only reward-improving entries. Experiments on AppWorld, τ^2-Bench, and LifelongAgentBench show that jointly optimizing policy and experience consistently outperforms optimizing either component alone. Further analyses validate the effects of retrieval design, update schedules, proposal validation, convergence behavior, and experience internalization. Future work may explore cross-environment experience transfer and compact experience-bank maintenance.

Limitations

The experience updater is instantiated with a closed-source LLM API, so its parameters cannot be updated to summarize entries that best match the evolving needs of the policy. We also leave experience consolidation for future work, including how to remove stale internalized entries and prevent unbounded growth of the experience bank.

References

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. EvoSkill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*.

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*.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for LLM agent training. *arXiv preprint arXiv:2505.10978*.

Yupeng Huo, Yaxi Lu, Zhong Zhang, Haotian Chen, and Yankai Lin. 2026. Atommem : Learnable dynamic agentic memory with atomic memory operation. *Preprint*, arXiv:2601.08323.

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, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.

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.

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*.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *Preprint*, arXiv:2602.08234.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

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

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Zhuo Li, Yujie Zheng, Weinan Zhang, Ying Wen, Zhiyu Li, and 1 others. 2026b. Memrl: Self-evolving agents via runtime reinforcement learning on episodic memory. *arXiv preprint arXiv:2601.03192*.

Zijing Zhang, Ziyang Chen, Mingxiao Li, Zhaopeng Tu, and Xiaolong Li. 2025. RLVMR: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. *arXiv preprint arXiv:2507.22844*.

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

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. 2025. Lifelongagentbench: Evaluating llm agents as lifelong learners. *arXiv preprint arXiv:2505.11942*.