CoEvolve: Training LLM Agents via Agent-Data Mutual Evolution

Shidong Yang*, **Ziyu Ma***, **Tongwen Huang***, **Yiming Hu**, **Yong Wang**†, **Xiangxiang Chu**
AMAP, Alibaba Group
https://github.com/AMAP-ML/CoEvolve

Abstract

Reinforcement learning for LLM agents is typically conducted on a static data distribution, which fails to adapt to the agent's evolving behavior and leads to poor coverage of complex environment interactions. To address these challenges, we propose CoEvolve, an agent-data mutual evolution framework that enables LLM agents to improve through closed-loop, interaction-driven training. Specifically, CoEvolve extracts feedback signals such as forgetting and uncertainty from rollout trajectories to identify failure-prone interaction patterns, and utilizes them to guide LLM-based task synthesis. The synthesized tasks are validated through environment interaction and utilized to update the data distribution, enabling joint adaptation of the agent and its data. Extensive experiments on AppWorld and BFCL across Qwen2.5-7B, Qwen3-4B, and Qwen3-30B-A3B demonstrate consistent and significant improvements over strong base models, yielding absolute gains of 19.43%, 15.58%, and 18.14%, respectively.

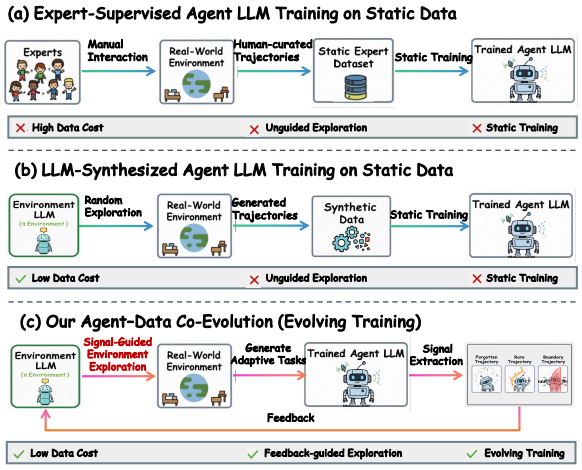

Figure 1: (a) Expert-Supervised. Agents learn from human-collected expert trajectories, incurring high data collection costs and limited generalization. (b) Static Synthetic. LLMs generate synthetic data in an offline and open-loop manner, yielding a static and non-adaptive training set. (c) Agent-Data Co-Evolution. Agents learn from tasks that evolve through feedback-driven interaction, enabling adaptive training without human supervision.

1 Introduction

The rapid advancement of large language models (LLMs) (Liu et al., 2024; Qwen, 2025; Gou et al., 2025) has driven the development of LLM-based agents, which have been widely applied to scenarios such as web information retrieval, software engineering, web navigation, and personal assistance (Jin et al., 2024; Ding et al., 2025; Trivedi et al., 2024; Ma et al., 2026, 2024). Reinforcement learning (RL) (Guo et al., 2025; Sun et al., 2024; Ji et al., 2025; Chu et al., 2025) has emerged as the dominant approach for training these agents with complex interactive capabilities, offering a general solution for acquiring adaptive behaviors in open-ended environments.

However, current agent RL training methods (Li et al., 2025; Mai et al., 2025b; Lin et al., 2025) heavily rely on human-written demonstrations, where experts manually interact with the environment to construct trajectory datasets. These curated trajectories are then used to train the agent's policy, as illustrated in Fig. 1(a). While effective on simple tasks, this reliance on manually curated data introduces several critical limitations: (1) Collecting interaction data in real environment is prohibitively expensive, with a single trajectory often requiring several minutes or more of human expert effort. Given the limited availability of expert time, broad exploration of the environment becomes difficult. (2) More fundamentally, these expert demonstrations represent static snapshots of interaction patterns and fail to cover the long-tail variations found in real-world settings (Wang et al., 2025c). As a result, agents trained on such data struggle to

*Equal contribution.
†Project lead and corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 23015–23036

generalize beyond the observed distribution. For instance, a web navigation agent may fail entirely if a button label changes from "Book Now" to "Reserve Now" (Gür et al., 2023).

The challenge of insufficient and static data has led to significant interest in synthetic data generation (Zhai et al., 2025; Mai et al., 2025a; Ding et al., 2024; Ye et al., 2024). A typical pipeline, illustrated in Fig. 1(b), prompts a large language model (LLM) with environment descriptions and task specifications to explore the environment. By leveraging its world knowledge and reasoning capabilities, the LLM generates synthetic trajectories that are subsequently used to train the agent. While synthetic data reduces reliance on human annotation, it is typically generated through random exploration guided solely by the LLM's world knowledge, without any feedback from the agent's actual performance or interaction signals. Therefore, the environment exploration remains shallow and incomplete, failing to sufficiently cover diverse environment configurations. Moreover, the generated data still constitutes a static corpus that cannot adapt to the agent's evolving capabilities, leading to inefficient training that neither targets specific weaknesses nor supports continual improvement.

To address these issues, we propose CoEvolve, an agent-data mutual evolution framework in which the agent and its training distribution evolve jointly through interaction-driven feedback, as shown in Fig. 1(c). Our core idea is to use feedback signals, such as forgetting signals, to identify failure-prone interaction patterns and guide LLM-based task discovery accordingly. Unlike previous methods that rely on static datasets, CoEvolve synthesizes new tasks targeting the agent's current weaknesses, validates them in the environment, and integrates them into training without human supervision. This closed loop allows the agent to reshape its learning distribution (data evolving) while continually overcoming its limitations (agent evolving).

We evaluate CoEvolve on two representative benchmarks, AppWorld (Trivedi et al., 2024) and BFCL (Patil et al., 2025), using Qwen2.5-7B, Qwen3-4B, and Qwen3-30B-A3B as backbones (Qwen, 2024b, 2025). By continuously synthesizing new tasks from training-time feedback, CoEvolve improves average performance by 19.43%, 15.58%, and 18.14%, respectively, demonstrating strong scalability and generalization across models and environments. Our contributions can be summarized as follows:

- We propose CoEvolve, an agent-data mutual evolution framework that alternates between agent optimization and data distribution updates without any human supervision.
- Unlike previous synthetic data generation based on unguided random exploration, we incorporate feedback signals (e.g., forgetting signals) into LLM-based environment exploration.
- CoEvolve yields large gains over baseline models (e.g., Qwen3-4B) across interactive benchmarks (e.g., AppWorld), demonstrating its effectiveness in complex environments.

2 Related Work

Large Language Model Agents. Recent work has shown that large language models (LLMs) can be instantiated as autonomous agents capable of long-horizon reasoning and action through iterative interaction with environments. Early frameworks such as ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023) demonstrate that coupling reasoning, tool use, and feedback enables LLMs to solve complex multi-step tasks, while later systems further enhance planning and memory for more persistent behaviors (Zhu et al., 2025). Despite these advances, most existing LLM agents are trained via imitation learning on static collections of expert trajectories (Nakano et al., 2021; Wang et al., 2023), which fundamentally limits exploration and constrains learning to the coverage of pre-collected data (Shinn et al., 2023). In contrast, our work departs from this static paradigm by enabling agents to learn in a dynamic, self-evolving training process without relying on fixed expert demonstrations.

Trajectory Synthesis for Agent Training. To reduce reliance on expert demonstrations, recent work explores synthetic trajectory generation for training LLM agents (Yu et al., 2025). Most prior approaches generate trajectories in an *offline* or weakly adaptive manner, including open-loop synthesis with reflection or correction (Ye et al., 2024; Ding et al., 2024; Chen et al., 2025c,b), as well as large-scale pipelines based on tutorials, scripted exploration, simulators, and self-training (Pahuja et al., 2025; Xu et al., 2024; Hoang et al., 2025; Yuan et al., 2025; Wang et al., 2025c; Song et al., 2024; Wang et al., 2025a). Recent extensions introduce more autonomous exploration or structured curricula (Wang et al., 2025b; Ramrakhya et al., 2025; Zhang et al., 2025b; Xiao et al., 2025; Chen et al., 2025a; Zhang et al., 2025a), yet trajectory

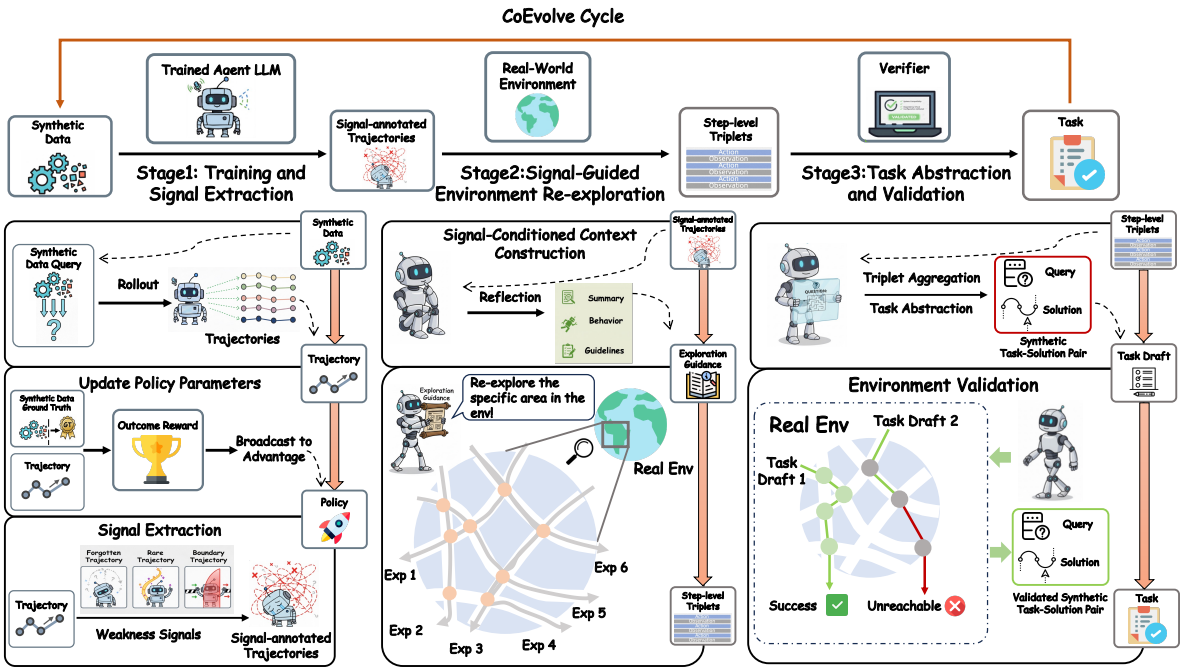

Figure 2: Overview of the **CoEvolve** framework. The agent is trained with GRPO, and feedback signals are extracted from rollout trajectories (Stage 1). These signals guide signal-conditioned re-exploration via an LLM (Stage 2) and are transformed into validated tasks to evolve the training set (Stage 3). This closed-loop process enables CoEvolve without human supervision.

generation remains largely *open-loop*, loosely coupled to the agent's evolving failure modes. In contrast, our method closes this loop by using environment feedback to synthesize trajectories on demand, enabling continuous adaptation of the training distribution. Conceptually, CoEvolve also differs from recent self-improving or curriculum-style frameworks that refine trajectories for a fixed pool of queries or generate variants around seed tasks. Our feedback is used to drive the agent back into the interactive environment to discover new executable queries and states, so data evolution is not limited to rewriting or filtering an offline query set.

3 Method

We propose CoEvolve, an agent-data co-evolution framework for training LLM agents without human supervision. In this section, we first introduce agent training on synthetic tasks and the extraction of weakness signals from rollout trajectories (Section 3.1). Then, Section 3.2 details how these signals are used as feedback to prompt LLM-based re-exploration for new task discovery. Section 3.3 finally describes how the discovered interactions are abstracted and validated into executable tasks and incorporated into training. The overall framework is illustrated in Fig. 2.

3.1 Training and Signal Extraction

Training on Synthetic Tasks. At training iteration t, we maintain a task set $\mathcal{D}_t$ consisting of executable synthetic tasks. The initial task set $\mathcal{D}_0$ is obtained via unguided exploration by a large language model interacting with the environment. As training proceeds, newly synthesized and validated tasks (described in later stages) are appended to $\mathcal{D}_t$, allowing the task distribution to evolve together with the agent.

For a task $x \in \mathcal{D}_t$, we sample a group of K trajectories $\{\tau_k\}_{k=1}^{K} \sim \pi_\theta(\cdot \mid x)$ and assign each trajectory a scalar reward $R(\tau_k)$. The agent is optimized using Group Relative Policy Optimization (GRPO) (Guo et al., 2025) by maximizing:

$$\mathcal{J}(\theta) = \frac{1}{\sum_{k=1}^{K} |\tau_k|} \sum_{k=1}^{K} \sum_{t=1}^{|\tau_k|} \mathrm{CLIP}(r_{k,t}(\theta), \hat{A}_k, \epsilon) - \beta \cdot \mathbb{D}_{\mathrm{KL}}[\pi_\theta \,\|\, \pi_{\mathrm{ref}}], \tag{1}$$

where $r_{k,t}(\theta) = \frac{\pi_\theta(a_t^k \mid s_t^k)}{\pi_{\theta_{\mathrm{old}}}(a_t^k \mid s_t^k)}$ is the importance ratio, and $\mathrm{CLIP}(r, A, \epsilon) = \min[r \cdot A, \mathrm{clip}(r, 1-\epsilon, 1+\epsilon) \cdot A]$. Here $\hat{A}_k$ denotes the group-relative advantage, π_{ref} is a fixed reference policy, and β weights the KL regularization term.

Signal Extraction. Beyond policy optimization, rollout trajectories generated during training contain instances of agent underperformance. To identify such weaknesses, we analyze these trajectories and define three types of behavioral signals: forgetting signals, boundary signals, and rare signals.

(1) Forgetting Signals. Following (Toneva et al., 2018), we use forgetting signals to detect cases where the agent previously succeeded on a task but now fails under the current policy. Let $s_{\text{now}} \in [0, 1]$ denote the task-level score of the current trajectory τ_{now}, computed from the environment's terminal reward or task-specific evaluation signal. For each task (or task type), we maintain a sliding window of recent scores:

$$\mathcal{H}_{\text{recent}} = \{s_{t-W+1}, \ldots, s_t\},$$

where W is the window size. A forgetting signal is triggered if

$$\exists\, s_i \in \mathcal{H}_{\text{recent}} \text{ such that } s_i \geq 0.5 \quad \text{and} \quad s_{\text{now}} < 0.5.$$

This condition indicates that the agent has previously succeeded on the task but now fails under the current policy. The current trajectory is marked as a forgetting signal and added to the set of signal-annotated trajectories.

(2) Boundary Signals. These signals identify tasks on which the agent exhibits high outcome variability under a fixed policy within a single training iteration. For a task $x \in \mathcal{D}_t$, we sample a group of K trajectories $\{\tau_k\}_{k=1}^{K} \sim \pi_\theta(\cdot \mid x)$, and obtain their normalized outcomes $\tilde{R}(\tau_k) \in [0, 1]$. A boundary signal is triggered if the sampled trajectories include both successful and failed outcomes:

$$\exists\, \tau_i, \tau_j \text{ such that } \tilde{R}(\tau_i) > 0.5 \quad \text{and} \quad \tilde{R}(\tau_j) < 0.5.$$

This condition captures tasks for which the agent's behavior is unstable, indicating proximity to the decision boundary. For any task that satisfies this condition, all sampled trajectories are marked as boundary signals and added to the set of signal-annotated trajectories.

(3) Rare Signals. These are defined as action patterns that have low empirical frequency over training yet recur across multiple trajectories, indicating systematic underexploration instead of one-off stochastic events (Shyalika et al., 2024). We extract an action pattern p from each trajectory and maintain its cumulative occurrence count c_p. Let N denote the total number of observed patterns. When $N \geq N_{\min}$, a rare signal is triggered if

$$\frac{c_p}{N} < \frac{\theta}{100} \quad \text{and} \quad c_p > 0,$$

where $\theta \in (0, 100)$ is a predefined frequency threshold (e.g., $\theta = 5$) that controls the rarity criterion. All trajectories containing such patterns are marked as rare signals and added to the set of signal-annotated trajectories. A single trajectory may trigger multiple signal types simultaneously. We evaluate forgetting, boundary, and rare signals independently and keep all activated signals because they capture complementary weaknesses.

3.2 Signal-Guided Environment Re-exploration

Given the signal-annotated trajectories identified in the previous stage, we perform signal-guided environment re-exploration to collect interaction data that targets the agent's identified weaknesses.

Signal-Conditioned Context Construction. For each signal-annotated trajectory, we provide the full interaction history to a large language model (LLM) and prompt it to reflect on the trajectory. Each trajectory contains the task description, the agent's executed action sequence, and the corresponding environment feedback. Based on this information, the LLM summarizes the underlying failure cause or behavioral instability that triggered the signal and produces a structured exploration context, which characterizes where and how the agent fails or behaves unstably.

LLM-Guided Re-exploration. Conditioned on the constructed context, the LLM is used to re-explore the environment to discover alternative behaviors. For each context, exploration is conducted along two orthogonal dimensions: (i) *multi-round exploration*, where multiple independent exploration runs are initiated from the same context to encourage behavioral diversity; and (ii) *multi-step exploration*, where each exploration run proceeds for multiple interaction steps, allowing the LLM to revise its actions based on intermediate observations. During re-exploration, at each step, the LLM produces an action a_t, the environment returns an observation o_t, and the interaction is recorded. As a result, the output of this stage is a collection of step-level interaction triplets (a_t, o_t, id), where id denotes the exploration rollout to which the step belongs. These triplets are subsequently grouped

Model	AppWorld-TestN		AppWorld-TestC		BFCL-V3	Avg.
	TGC	SGC	TGC	SGC	Multi-turn	
Closed-source LLMs						
Claude-Sonnet-4.5	73.81	55.36	49.88	32.37	69.00	56.08
GPT-4	30.40	21.40	14.60	9.30	54.00	25.94
Gemini-2.5-Flash	53.57	32.14	40.05	20.14	41.50	37.48
Open-source Models						
DeepSeek-V3.2	47.02	30.36	22.78	9.35	41.50	30.20
GPT-OSS-20B	17.86	3.57	5.52	0.00	14.00	8.19
LLaMA-3.3-70B	31.54	8.92	16.31	4.32	26.00	17.42
Gemma-3-27B	23.81	7.14	9.83	0.72	16.50	11.60
Baseline and CoEvolve						
Qwen2.5-7B-Instruct	1.19	0.00	0.72	0.00	13.50	3.08
-w/ CoEvolve	**27.98** (↑26.79)	**12.50** (↑12.50)	**8.39** (↑7.67)	**2.16** (↑2.16)	**61.50** (↑48.00)	**22.51** (↑19.43)
Qwen3-4B-Instruct	16.67	5.36	7.91	2.16	26.50	11.72
-w/ CoEvolve	**35.71** (↑19.04)	**14.28** (↑8.92)	**17.03** (↑9.12)	**6.47** (↑4.31)	**63.00** (↑36.50)	**27.30** (↑15.58)
Qwen3-30B-A3B	31.55	12.50	19.90	5.76	43.50	22.64
-w/ CoEvolve	**54.76** (↑23.21)	**33.93** (↑21.43)	**31.65** (↑11.75)	**16.55** (↑10.79)	**67.00** (↑23.50)	**40.78** (↑18.14)

Table 1: Performance comparison on AppWorld (Test-Normal TGC/SGC and Test-Challenge TGC/SGC) and BFCL-V3 (Multi-turn base). Results are reported for closed-source LLMs, open-source models, and the backbones with and without CoEvolve. Improvements introduced by CoEvolve are indicated by ↑.

by task and serve as the input to the next stage for task abstraction and validation.

3.3 Task Abstraction and Validation

Given the step-level action-observation triplets collected during the above stage, we next synthesize new executable tasks to update the task set $\mathcal{D}_t$.

Triplet Aggregation and Task Abstraction. We first group the collected interaction triplets by their associated task, where each group aggregates action-observation pairs from multiple exploration rollouts under the same task context. These groups capture diverse behavioral evidence on how the task may be completed. We then prompt a large language model to abstract each group into a task-level specification. Instead of copying step-level interactions, the model identifies the user intent, formulates a concise task query, and derives a plausible action sequence as a solution. This process transforms triplets into task-solution pairs.

Environment Validation. Each synthesized task-solution pair is validated through execution in the environment. Specifically, we instantiate the environment associated with the task and provide the generated task query and action sequence to an LLM agent for execution. If the execution successfully completes the task objective, the synthesized task is accepted. If execution fails but the environment returns a positive reward, the task is also retained. Tasks that fail both criteria are discarded. Validated tasks are appended to the current task set $\mathcal{D}_t$, forming the updated training distribution for the next iteration. By iteratively abstracting, validating, and incorporating new tasks, this stage allows the training data to adapt to the agent's weaknesses, completing the co-evolution loop.

4 Experiments

4.1 Experimental Setup

We evaluate our method on two widely used benchmarks: AppWorld (Trivedi et al., 2024) and BFCL-V3 Multi-Turn Base (Patil et al., 2025). For AppWorld, we report results on the official Test-Normal (TestN) and Test-Challenge (TestC) splits, using Task Goal Completion (TGC) and Scenario Goal Completion (SGC) to measure final task success and scenario-level execution accuracy, respectively. For BFCL-V3, we follow the standard Multi-Turn Base protocol and evaluate on the provided validation set, reporting multi-turn success rate.

4.2 Implementation Details

We implement all experiments with the VeRL framework (Sheng et al., 2025). Specifically,

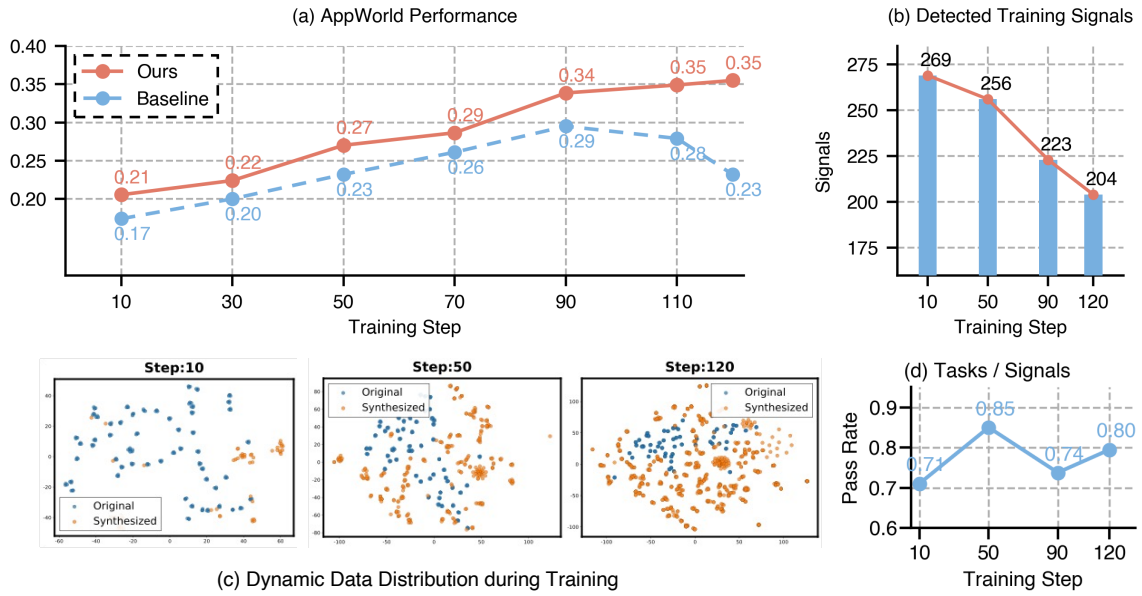

Figure 3: Dynamics of CoEvolve during training. (a) Performance comparison between CoEvolve and the baseline on AppWorld as training progresses. (b) Number of detected signals across training steps. (c) Evolution of the data distribution, showing the relationship between original and synthesized tasks at different stages. (d) Conversion from detected weakness signals to newly generated training tasks over training. Together, the figure shows how feedback signals guide data generation, reshape the data distribution, and support stable performance improvement.

Backbone	AppWorld TestN (TGC)			BFCL-V3		
	Zero-shot	GRPO	CoEvolve	Zero-shot	GRPO	CoEvolve
Qwen2.5-7B-Instruct	1.19	26.78	**27.98**	13.50	56.00	**61.50**
Qwen3-4B-Instruct	16.67	28.57	**35.71**	26.50	58.00	**63.00**
Qwen3-30B-A3B-Instruct	31.55	48.81	**54.76**	43.50	64.00	**67.00**

Table 2: Comparison with zero-shot and GRPO on AppWorld TestN and BFCL-V3. CoEvolve is built on top of GRPO and yields complementary gains across model scales.

Qwen2.5-7B-Instruct and Qwen3-4B-Instruct are trained on one node with 8 NVIDIA H20 GPUs, while Qwen3-30B-A3B-Instruct is trained on 16 H20 GPUs. We use GRPO with a constant learning rate of $e{-}6$, $n{=}8$ samples per prompt, and KL coefficient $e{-}3$. Rollout temperature is 0.9.

4.3 Main Results

Table 1 reports the main results on AppWorld and BFCL-V3, comparing closed-source LLMs, strong open-source baselines, and our backbone models trained with and without the proposed framework.

Overall Performance. CoEvolve consistently improves performance across all evaluated backbones, starting from weak instruction-following baselines. On Qwen2.5-7B, the average score increases by 19.4; Qwen3-4B improves by 15.6. These gains close the gap with much larger open-source models (e.g., DeepSeek-V3.2 at 30.20). Notably, all improvements are achieved without any human annotation or handcrafted task design, highlighting the scalability of CoEvolve as a broadly applicable training strategy rather than a model-specific trick.

Results on AppWorld and BFCL-V3. On AppWorld, CoEvolve brings +23.21 / +21.43 gains (TGC/SGC) on the challenge split and +11.75 / +10.79 on the normal split for Qwen3-30B-A3B, indicating that CoEvolve more effectively addresses failure-prone, unstable, and underexplored interaction patterns targeted by the proposed training-time feedback signals. On BFCL-V3, it improves Qwen2.5-7B-Instruct by +48.0 and Qwen3-4B-Instruct by +36.5, with smaller models benefiting more from feedback-driven training.

Comparison with Closed-source LLMs. CoEvolve enables mid-sized open models to outperform several closed-source baselines, despite lacking access to proprietary data. On BFCL-V3, Qwen3-4B

Training Phase	AppWorld	BFCL	Avg.
Zero-shot (Qwen3-4B)	16.67	26.50	21.59
+ Synthetic Data	28.57	58.00	43.29
+ Random Exploration	30.36	60.50	45.43
+ Feedback	**35.71**	**63.00**	**49.36**

Table 3: Ablation study of different training phases on Qwen3-4B across two benchmarks (AppWorld and BFCL). "Avg." denotes the mean success rate across AppWorld and BFCL, showing that each phase contributes incremental gains, with the best performance achieved after incorporating feedback.

Dimension	Value	AppWorld	BFCL
Initial Data Size (N)	50	26.79	53.00
	100	35.12	**63.00**
	200	**38.10**	60.00
Gen. Frequency (F)	5	**35.71**	58.00
	10	35.12	**63.00**
	20	33.93	57.50

Table 4: Hyperparameter sensitivity analysis for Qwen3-4B on AppWorld and BFCL benchmarks. We investigate the impact of initial synthetic data size (N), and generation frequency (F).

with CoEvolve reaches 63.00, surpassing GPT-4 (54.00) and Gemini-2.5-Flash (41.50). These results suggest that CoEvolve improves generalization to complex interactions rather than overfitting to task environments.

Comparison with GRPO. CoEvolve extends standard GRPO by introducing feedback-guided data evolution during RL training. Table 2 shows that CoEvolve consistently improves over GRPO across all three backbones, confirming that closed-loop data evolution provides complementary improvements on top of GRPO, rather than replacing it.

4.4 Ablation Study

Unless otherwise specified, all ablation experiments are conducted using the Qwen3-4B-Instruct backbone. For AppWorld, we report task-level goal completion (TGC) scores on the TestN split.

Dynamics of Agent–Data CoEvolve. Fig. 3 shows how agent performance, detected signals, and synthesized data evolve throughout training. Performance improves steadily (0.21 → 0.35), while the baseline rises initially before falling (0.17 → 0.29 → 0.23), indicating more stable optimization under closed-loop training. Generated tasks expand into previously underrepresented regions (Fig. 3(c)), showing that synthesis produces diverse, non-redundant data. The number of detected signals drops from 269 to 204 (Fig. 3(b)), suggesting progressive resolution of failure-prone cases. The pass rate of signal-driven tasks improves from 0.71 to 0.85 and stabilizes at 0.80 (Fig. 3(d)), confirming the effectiveness of feedback-guided generation. Overall, these trends support CoEvolve's core design: using feedback signals to adaptively reshape data distribution and target evolving model weaknesses.

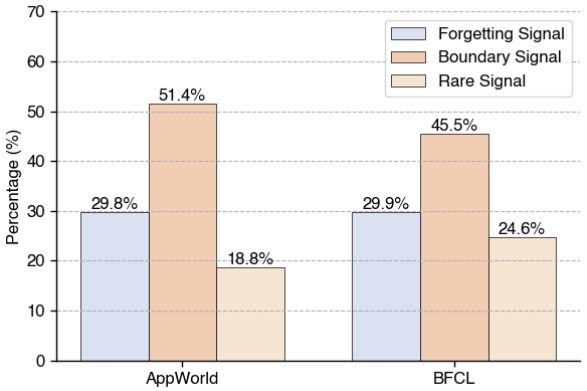

Figure 4: Distribution of extracted signals on AppWorld and BFCL. Boundary signals dominate (51.4% on AppWorld and 45.5% on BFCL), followed by forgetting and rare signals.

Effect of Closed-loop CoEvolve. Table 3 shows an ablation study on Qwen3-4B, isolating the impact of each training stage. Starting from a zero-shot baseline (21.59), static synthetic data already provides a strong boost (43.29), confirming the value of offline task construction. Adding random exploration brings further gains (45.43), indicating that online trajectory generation can help. However, the most significant improvement comes from incorporating feedback signals, which raises the average score to 49.36. Compared with random exploration, feedback-guided generation yields consistent gains on AppWorld (30.36 → 35.71) and BFCL-V3 (60.50 → 63.00), underscoring the importance of using model feedback to shape the evolving training set.

Hyperparameter Sensitivity. Table 4 shows that both initialization size (N) and generation frequency (F) affect performance. $N = 100$ gives the best BFCL score (63.00), while $N = 200$ improves AppWorld (38.10) but slightly degrades BFCL. For F, $F = 5$ achieves the highest AppWorld score (35.71), while $F = 10$ yields the best BFCL score (63.00). Overly sparse updates ($F = 20$) degrade both. These results suggest that moderate initializa-

Training Configuration	AppWorld	BFCL	Avg.
CoEvolve	**35.71**	**63.00**	**49.36**
w/o Forgotten Signals	30.36	60.00	45.18
w/o Rare Signals	33.92	60.50	47.21
w/o Boundary Signals	33.33	61.00	47.17

Table 5: Ablation study on feedback signals for Qwen3-4B. Each row represents the performance after removing a specific type of feedback-driven sample.

Training Domain	AppWorld	BFCL
Zero-shot Baseline (Qwen3-4B)	16.67	26.50
AppWorld (Ours)	**35.71**	45.00
BFCL (Ours)	19.04	**63.00**

Table 6: Cross-domain transferability analysis of Qwen3-4B. The diagonal entries represent intra-domain performance, while off-diagonal entries indicate zero-shot generalization to unseen tool-use environments.

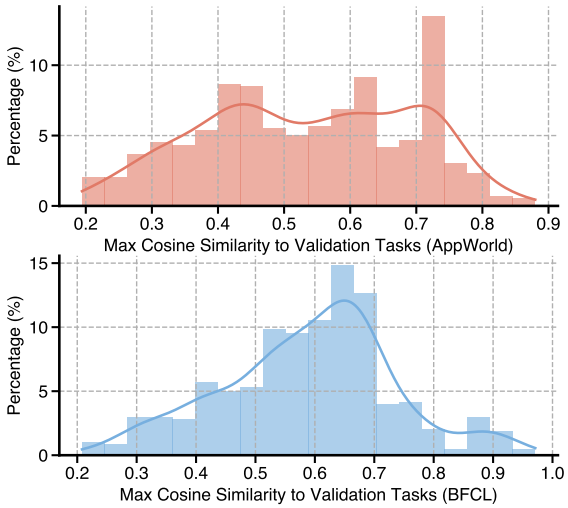

Figure 5: Distribution of maximum cosine similarity between synthesized tasks and their validation tasks.

tion and sufficiently frequent updates are important for feedback-driven training.

Ablation and Distribution of Feedback Signals. To better understand the role of individual feedback signals, we analyze both their distribution and their impact on performance. As shown in Figure 4, boundary signals account for the largest proportion across both AppWorld (51.4%) and BFCL (45.5%), followed by forgetting and rare signals. This suggests that agents frequently struggle at decision boundaries and with previously learned cases, justifying their use as guidance for task synthesis. Table 5 further shows that removing any single signal leads to performance degradation, confirming their complementary value. In particular, forgetting signals contribute the most, with a drop of nearly 4 points (49.36 $\rightarrow$ 45.18), reflecting their utility in correcting regressions during training. Boundary and rare signals also provide meaningful gains (1.6$\sim$1.9), indicating their importance in exposing edge cases and long-tail scenarios. Together, these results validate that CoEvolve benefits from a diverse signal set rather than a single heuristic.

Cross-domain Generalization. Table 6 evaluates whether CoEvolve-trained agents generalize across domains. Training on AppWorld improves zero-shot performance on BFCL from 26.50 to 45.00, and vice versa from 16.67 to 19.04. While in-domain performance remains highest (35.71, 63.00), the off-domain gains show that CoEvolve learns transferable strategies beyond environment-specific behaviors.

Analysis of Data Diversity. Fig. 5 analyzes the similarity between synthesized tasks and validation examples. Across both AppWorld and BFCL, most samples fall within a moderate similarity range (e.g., 0.4$\sim$0.7), with only a small fraction approaching 1.0. This indicates that the synthesis process produces novel tasks rather than near-duplicates. The consistent patterns observed across domains further suggest that the feedback-driven exploration effectively guides task discovery, maintaining meaningful data diversity without collapsing into repetitive samples.

Behavioral Comparison with GRPO Baseline. Table 8 compares CoEvolve against a GRPO-trained baseline without closed-loop evolution. On BFCL, CoEvolve preserves 53.00% of correct predictions and recovers 10.00% of previously failed cases. On AppWorld, the corresponding numbers are 19.04% and 16.67%. These results indicate that feedback-driven training not only retains prior strengths but also effectively corrects failure cases, yielding more reliable and adaptive agent behavior.

Cost-Efficiency of CoEvolve. Table 7 evaluates the additional training cost and corresponding performance improvement brought by CoEvolve, compared to a baseline that performs GRPO training without closed-loop task generation. Across both benchmarks, the CoEvolve framework introduces only $\sim$10% additional computation, yet leads to clear absolute gains (+6.53 on AppWorld, +5.00 on BFCL) and substantial relative improvements (+22.92%, +8.62%). Each feedback iteration incurs minimal cost, but collectively reshapes the

Benchmark	Feedback Time Ratio	Single Feedback Cost	Total Time	Performance Gain	Relative Gain
AppWorld	9.67%	811s	100,596s	28.57 → 35.12	+22.92%
BFCL	12.76%	480s	45,144s	58.00 → 63.00	+8.62%

Table 7: Cost and Efficiency of CoEvolve. CoEvolve introduces minimal computational overhead, yet yields substantial performance gains across benchmarks, showing its effectiveness as an efficient training strategy.

Benchmark	Baseline \ Ours	Correct	Wrong
BFCL	Correct	53.00%	5.00%
	Wrong	10.00%	32.00%
AppWorld	Correct	19.04%	9.53%
	Wrong	16.67%	54.76%

Table 8: Cross comparison between CoEvolve (Ours) and the baseline on BFCL and AppWorld.

training distribution to better address model weaknesses. These results confirm that CoEvolve is not only effective but also efficient, offering a favorable trade-off between cost and performance.

5 Conclusion

We introduce CoEvolve, a reinforcement learning framework that enables mutual evolution between the agent and its data distribution. By extracting feedback signals (e.g., forgetting signal) during policy optimization and using them to guide task synthesis, our method progressively adapts both the agent's capabilities and the data it learns from. Extensive experiments on AppWorld and BFCL validate its effectiveness and efficiency. We hope this work inspires future research on agent evolution toward agents that can autonomously improve via interaction-driven feedback.

Limitations

This work presents an exploration of feedback-driven agent-data co-evolution using a limited set of feedback signals, including forgetting signals, boundary signals, and rare signals. While effective, these signals cover only a subset of potentially informative feedback and may be further enriched in future work. In addition, the extracted signals are derived from the agent's own interaction trajectories and therefore depend on the current policy. At early stages of training, when the agent's behavior is still immature, the resulting signals may be noisy or incomplete, highlighting the need for more robust feedback extraction under low-competence regimes. Because CoEvolve autonomously reshapes its training distribution, adversarial or safety-critical settings may require human oversight, policy constraints, and continuous auditing before synthesized tasks are admitted into training. Future work should incorporate explicit safety filters and risk-triggered review so that feedback-driven adaptation remains controllable.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Anthropic. 2025. Claude sonnet 4.5. https://docs.anthropic.com/claude/docs/models-overview. Accessed: 2025-01.

Peter Chen, Xi Chen, Wotao Yin, and Tianyi Lin. 2025a. Compo: Preference alignment via comparison oracles. *arXiv preprint arXiv:2505.05465*.

Peter Chen, Xiaopeng Li, Ziniu Li, Xi Chen, and Tianyi Lin. 2025b. Stepwise guided policy optimization: Coloring your incorrect reasoning in grpo. *arXiv preprint arXiv:2505.11595*.

Yihan Chen, Benfeng Xu, Xiaorui Wang, Yongdong Zhang, and Zhendong Mao. 2025c. Training llm-based agents with synthetic self-reflected trajectories and partial masking. *arXiv preprint arXiv:2505.20023*.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. 2025. Gpg: A simple and strong reinforcement learning baseline for model reasoning. *arXiv preprint arXiv:2504.02546*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Bosheng Ding, Chengwei Qin, Ruochen Zhao, Tianze Luo, Xinze Li, Guizhen Chen, Wenhan Xia, Junjie Hu, Luu Anh Tuan, and Shafiq Joty. 2024. Data augmentation using llms: Data perspectives, learning paradigms and challenges. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1679–1705.

Hanxing Ding, Shuchang Tao, Liang Pang, Zihao Wei, Jinyang Gao, Bolin Ding, Huawei Shen, and Xueqi Cheng. 2025. Toolcoder: A systematic code-empowered tool learning framework for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17876–17891.

Chenhui Gou, Ziyu Ma, Zicheng Duan, Haoyu He, Feng Chen, Akide Liu, Bohan Zhuang, Jianfei Cai, and Hamid Rezatofighi. 2025. An empirical study on how video-llms answer video questions. *arXiv preprint arXiv:2508.15360*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, and 1 others. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Izzeddin Gür, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2023. Understanding html with large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2803–2821.

Thai Quoc Hoang, Kung-Hsiang Huang, Shirley Kokane, Jianguo Zhang, Zuxin Liu, Ming Zhu, Jake Grigsby, Tian Lan, Michael S Ryoo, Chien-Sheng Wu, Shelby Heinecke, Huan Wang, Silvio Savarese, Caiming Xiong, and Juan Carlos Niebles. 2025. LAM SIMULATOR: Advancing data generation for large action model training via online exploration and trajectory feedback. *Findings of the Association for Computational Linguistics: ACL 2025*.

Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. 2025. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240*.

Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2024. From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479*.

Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Xiaoxi Li, Wenxiang Jiao, Jiarui Jin, Guanting Dong, Jiajie Jin, Yinuo Wang, Hao Wang, Yutao Zhu, Ji-Rong Wen, Yuan Lu, and 1 others. 2025. Deepagent: A general reasoning agent with scalable toolsets. *arXiv preprint arXiv:2510.21618*.

Minhua Lin, Zongyu Wu, Zhichao Xu, Hui Liu, Xianfeng Tang, Qi He, Charu Aggarwal, Xiang Zhang, and Suhang Wang. 2025. A comprehensive survey on reinforcement learning-based agentic search: Foundations, roles, optimizations, evaluations, and applications. *arXiv preprint arXiv:2510.16724*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

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

Ziyu Ma, Chenhui Gou, Yiming Hu, Yong Wang, Bohan Zhuang, and Jianfei Cai. 2026. Where and what matters: Sensitivity-aware task vectors for many-shot multimodal in-context learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(10):7892–7900.

Ziyu Ma, Chenhui Gou, Hengcan Shi, Bin Sun, Shutao Li, Hamid Rezatofighi, and Jianfei Cai. 2024. Drvideo: Document retrieval based long video understanding. *arXiv preprint arXiv:2406.12846*.

Shinji Mai, Yunpeng Zhai, Ziqian Chen, Cheng Chen, Anni Zou, Shuchang Tao, Zhaoyang Liu, and Bolin Ding. 2025a. Cues: A curiosity-driven and environment-grounded synthesis framework for agentic rl. *arXiv preprint arXiv:2512.01311*.

Xinji Mai, Haotian Xu, Zhong-Zhi Li, Xing W, Weinong Wang, Jian Hu, Yingying Zhang, and Wenqiang Zhang. 2025b. Agent rl scaling law: Agent rl with spontaneous code execution for mathematical problem solving. *arXiv preprint arXiv:2505.07773*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan. 2025. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6300–6323.

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

Qwen. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2(3).

Qwen. 2024b. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Qwen. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Ram Ramrakhya, Andrew Szot, Omar Attia, Yuhao Yang, Anh Nguyen, Bogdan Mazoure, Zhe Gan, Harsh Agrawal, and Alexander Toshev. 2025. Scaling synthetic task generation for agents via exploration. *arXiv preprint arXiv:2509.25047*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*.

Chathurangi Shyalika, Ruwan Wickramarachchi, and Amit P Sheth. 2024. A comprehensive survey on rare event prediction. *ACM Computing Surveys*, 57(3):1–39.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024. Agentbank: Towards generalized llm agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141.

Chuanneng Sun, Songjun Huang, and Dario Pompili. 2024. Llm-based multi-agent reinforcement learning: Current and future directions. *arXiv preprint arXiv:2405.11106*.

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

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. 2018. An empirical study of example forgetting during deep neural network learning. *arXiv preprint arXiv:1812.05159*.

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. *arXiv preprint arXiv:2305.16291*.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. 2025a. STeCa: Step-level trajectory calibration for LLM agent learning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11597–11614, Vienna, Austria. Association for Computational Linguistics.

Yiming Wang, Da Yin, Yuedong Cui, Ruichen Zheng, Zhiqian Li, Zongyu Lin, Di Wu, Xueqing Wu, Chenchen Ye, Yu Zhou, and 1 others. 2025b. Llms as scalable, general-purpose simulators for evolving digital agent training. *arXiv preprint arXiv:2510.14969*.

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. 2025c. Co-evolving llm coder and unit tester via reinforcement learning. *arXiv preprint arXiv:2506.03136*.

Yang Xiao, Mohan Jiang, Jie Sun, Keyu Li, Jifan Lin, Yumin Zhuang, Ji Zeng, Shijie Xia, Qishuo Hua, Xuefeng Li, and 1 others. 2025. Limi: Less is more for agency. *arXiv preprint arXiv:2509.17567*.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. 2024. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. *arXiv preprint arXiv:2412.09605*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

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

Junjie Ye, Xuanteng Gao, Keqing Zhang, Guodian Gao, Yutao Li, and Xiaozhi Liu. 2024. LLM-DA: Data augmentation via large language models for few-shot named entity recognition. *arXiv preprint arXiv:2402.14568*.

Zhaochen Yu, Ling Yang, Jiaru Zou, Shuicheng Yan, and Mengdi Wang. 2025. Demystifying reinforcement learning in agentic reasoning. *arXiv preprint arXiv:2510.11701*.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. 2025. Agent-R: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*.

Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, and 1 others. 2025. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, and 1 others. 2025a. Agent learning via early experience. *arXiv preprint arXiv:2510.08558*.

Shaolei Zhang, Ju Fan, Meihao Fan, Guoliang Li, and Xiaoyong Du. 2025b. Deepanalyze: Agentic large language models for autonomous data science. *arXiv preprint arXiv:2510.16872*.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. 2025. Knowagent: Knowledge-augmented planning for llm-based agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3709–3732.

A Appendix

A.1 Dataset

For completeness, we summarize the datasets used in both the main paper and the appendix-only transfer experiments. Besides AppWorld and BFCL, the appendix additionally reports results on ALFWorld and WebShop to assess transfer beyond API-centric settings.

AppWorld. AppWorld is a simulated environment for real-world digital service interactions, covering applications such as calendar, email, music, and social platforms. Agents solve tasks via Python API calls (e.g., "find the most-liked song in my Spotify playlists"), typically requiring multi-step reasoning and cross-app information aggregation. We report Task Goal Completion (TGC) and Scenario Goal Completion (SGC). TGC measures success on an individual task, while SGC measures whether all tasks within a scenario are completed successfully, reflecting broader consistency across related subtasks.

BFCL. BFCL (Berkeley Function Calling Leaderboard) evaluates function/tool calling ability, including multi-turn, parallel, and nested tool-use scenarios. We use the BFCL v3 Multi-turn subset for evaluation and we evaluate models using multi-turn function calling accuracy. A test case is considered successful only if the model selects the correct function and generates semantically and syntactically valid arguments at every turn of the interaction. Any error at an intermediate step results in failure of the entire instance. This metric therefore provides a strict measure of long-horizon tool-use consistency, reflecting the model's ability to maintain correct function semantics and parameter grounding across multi-step interactions.

WebShop. WebShop (Yao et al., 2022) is an interactive environment that simulates an e-commerce shopping scenario. An agent interacts with the environment through two actions, `search[query]` and `click[element]`, to complete natural-language shopping requests via product search, attribute filtering, and purchase decisions. We evaluate performance using the attribute-matching score between the final selected product and the user request.

ALFWorld. ALFWorld (Shridhar et al., 2021) is a text-only environment derived from household embodied tasks in ALFRED. It requires an agent to solve long-horizon tasks in partially observable indoor environments through textual actions for navigation, container operations, and object manipulation. The task set includes pick-and-place, examination, cleaning, heating, cooling, and multi-object placement scenarios. We report success rate, where an episode is counted as successful only when the full goal is completed.

A.2 Implementation Details

We use the VeRL framework to train the agent with GRPO. The detailed hyperparameters are summarized in Table 9. For Qwen2.5-7B-Instruct and Qwen3-4B-Instruct, training is conducted on a single machine equipped with 8× NVIDIA H20 GPUs (Tensor Parallel = 1), while Qwen3-30B-A3B-Instruct is trained across two machines with 8× H20 GPUs each (Tensor Parallel = 2). During training, each interaction episode is capped at 30

Parameter	Value
Learning rate	1e−6
Group size (n)	8
Training batch size	32
Optimizer	AdamW
Clip ratio low	0.20
Clip ratio high	0.28
KL coefficient	1e−3
Rollout temperature	0.9
Evaluation temperature	0
Max response length	4096
Reward signal	success = 1, failure = 0
Max interaction step	30

Table 9: Hyperparameters for RL training.

environment steps for AppWorld and BFCL, and 15 steps for WebShop and ALFWorld. Exceeding these limits is treated as task failure. Unless otherwise specified, we initialize the synthetic task set with 100 tasks, train for a total of 120 steps, and regenerate feedback data every 10 training steps. We use Qwen3-Max (Team, 2025) as the exploration LLM.

We compare against closed-source LLMs (Claude Sonnet 4.5 (Anthropic, 2025), GPT-4 (Achiam et al., 2023), and Gemini-2.5-Flash (Comanici et al., 2025)) and open-source LLMs (DeepSeek-V3.2 (Liu et al., 2025), GPT-OSS-20B (Agarwal et al., 2025), LLaMA-3.3-70B (Grattafiori et al., 2024), and Gemma-3-27B (Kamath et al., 2025)). We also report results for backbone models (Qwen2.5-7B-Instruct (Qwen, 2024a), Qwen3-4B-Instruct (Qwen, 2025), and Qwen3-30B-A3B-Instruct (Qwen, 2025)) with and without CoEvolve.

A.3 Additional Experiments and Analyses

Comparison with adaptive data-generation methods on diverse environments. We further evaluate whether the gains of CoEvolve transfer beyond API/function-calling tasks. To this end, we compare CoEvolve with zero-shot, GRPO, and adaptive data-generation baselines, including Reflexion (Shinn et al., 2023) and ReST (Gulcehre et al., 2023), on ALFWorld (Shridhar et al., 2021), BFCL, AppWorld, and WebShop (Yao et al., 2022) under the same Qwen3-4B-Instruct backbone.

Table 10 shows that CoEvolve transfers beyond function-calling environments. Under the same Qwen3-4B-Instruct backbone, it consistently outperforms all baselines across ALFWorld, BFCL, AppWorld, and WebShop. In particular, CoEvolve beats Curriculum Learning and ReST (Gulcehre et al., 2023) on ALFWorld and WebShop, and surpasses Reflexion (Shinn et al., 2023) and Curriculum Learning on BFCL and AppWorld. These results suggest that the gains of CoEvolve extend to broader interactive settings such as household decision-making and web navigation.

Task Validation for Abstracted Tasks. Table 11 examines the role of task validation during abstracted task generation on BFCL (Multi-turn Base) and AppWorld (TestN). In this ablation, we remove the validation step that filters abstracted tasks through environment execution, while keeping all other components unchanged.

Removing task validation leads to a clear and consistent performance degradation across both benchmarks. On BFCL, the score drops from 63.00 to 58.50, while on AppWorld the performance decreases more sharply from 35.71 to 27.38. These results indicate that without validation, a substantial portion of synthesized tasks are either noisy or misaligned with the environment dynamics, which in turn degrades downstream training.

This ablation highlights the importance of validation as a critical component of the feedback-driven data evolution process. By grounding abstracted tasks in actual environment execution, validation ensures that newly added data reflects executable and informative interactions rather than spurious abstractions. As a result, task validation plays a key role in maintaining the quality of the evolving training distribution and enabling effective agent–data co-evolution.

Controlled study of exploration model quality and feedback. Table 12 controls the external exploration model used for both initial synthesis and re-exploration on BFCL, while keeping the training model fixed to Qwen3-4B-Instruct. Each row uses the same external model for the "Synthesis Only" baseline and the full CoEvolve variant, so the gap isolates the contribution of feedback-guided data evolution rather than model substitution alone.

The results support two conclusions. First, stronger exploration models raise the attainable ceiling (Qwen3-4B < Qwen-Plus < Qwen-Max), which is expected and consistent with intuition. Second, under the same exploration model, adding feedback improves over the corresponding "Only Synthesis" baseline, showing that CoEvolve bene-

Method	ALFWorld	BFCL	AppWorld	WebShop
Zero-shot	30.00	26.50	16.67	5.00
Reflexion	35.00	31.00	18.45	10.50
Curriculum Learning	78.57	54.50	29.17	78.60
ReST	77.94	52.00	32.74	77.20
GRPO	77.85	58.00	28.57	75.00
CoEvolve	**82.86**	**63.00**	**35.71**	**80.60**

Table 10: Comparison with adaptive data-generation methods on four interactive environments using Qwen3-4B-Instruct.

Original Sample

Transfer a duplicate of 'test_document.txt' over to the archives folder and rename it 'final_document.txt'

```
# step1
cp(source='test_document.txt',destination='archives')
# step2
cd(folder='archives')
# step3
mv(source='test_document.txt',destination='final_document.txt')
```

Synthetic Sample

Create a non-empty test file named 'test.txt', copy it to 'test_copy.txt', and verify both files have identical content.

```
# step1
touch(file_name='test.txt')
# step2
echo(content='This is test content for boundary case exploration.', file_name='test.txt')
# step3
cp(source='test.txt', destination='test_copy.txt')
# step4
cat(file_name='test.txt')
# step5
cat(file_name='test_copy.txt')
```

Figure 6: BFCL-V3 cases. Simple File Copy/Rename vs. Constraint-Based Copy with Content Verification

Setting	BFCL	AppWorld
Default	**63.00**	**35.71**
+ w/o validation	58.50	27.38

Table 11: Impact of removing validation for abstracted tasks, using Qwen3-4B-Instruct. Metrics are on BFCL Multi-turn Base and AppWorld Test-Normal.

Exploration Model	Synthesis Only	CoEvolve
Qwen3-4B	53.00	56.50
Qwen-Plus	54.00	59.50
Qwen-Max	**58.00**	**63.00**

Table 12: Exploration-model study on BFCL under matched external models.

fits from framework design beyond simply swapping in a stronger external model.

Similarity-controlled task synthesis. To better understand the relationship between task relevance and final performance, we group synthesized BFCL tasks by their maximum similarity to validation tasks into low, medium, high, and mixed settings. The mean similarity of the low, medium, and high bins is 38.43%, 53.28%, and 64.73%, respectively.

As shown in Table 13, these results provide two insights. First, performance is non-monotonic across single similarity bins, suggesting that similarity alone does not determine final performance. Second, the mixed setting performs best, indicating that balancing relevance and diversity across similarity levels is more effective than concentrating synthesis on a single range.

Setting	BFCL
High-similarity synthesis	56.50
Medium-similarity synthesis	59.00
Low-similarity synthesis	56.50
Mixed synthesis	**63.00**

Table 13: Similarity-controlled synthesis on BFCL using Qwen3-4B-Instruct.

Setting	BFCL
$N = 25$	52.00
$N = 500$	58.00
$F = 2$	62.00
$F = 40$	58.00

Table 14: Extended hyperparameter sensitivity on BFCL with Qwen3-4B-Instruct.

Extended hyperparameter range. We further evaluate hyperparameter values beyond the range considered in the main paper to test robustness outside the budgeted setting in Table 4. CoEvolve remains reasonably robust beyond the reported range, while more extreme settings mainly introduce a trade-off between data quality and update cadence rather than changing the overall conclusion.

A.4 Analysis of Interaction Turns.

Figure 7 compares the distribution of interaction turns between original and synthesized trajectories on BFCL. Relative to the original data, synthesized tasks exhibit a noticeable shift toward higher step counts and a heavier tail, indicating that they more frequently involve longer interaction sequences and multi-step dependencies. This distributional difference suggests that feedback-driven task synthesis tends to generate structurally more complex interaction scenarios, rather than concentrating on the short-horizon tasks that dominate the original dataset. In contrast, the original data shows a more concentrated distribution over interaction length, covering a narrower range of step counts. By introducing tasks with longer interaction sequences, CoEvolve expands the coverage of the training data distribution at the level of interaction structure. Overall, these observations indicate that feedback-driven data evolution can alter the distribution of interaction lengths in a systematic manner. This shift is consistent with the design goal of CoEvolve, which aims to complement static datasets by dynamically discovering underrepresented interaction patterns, without making claims beyond the data distribution itself.

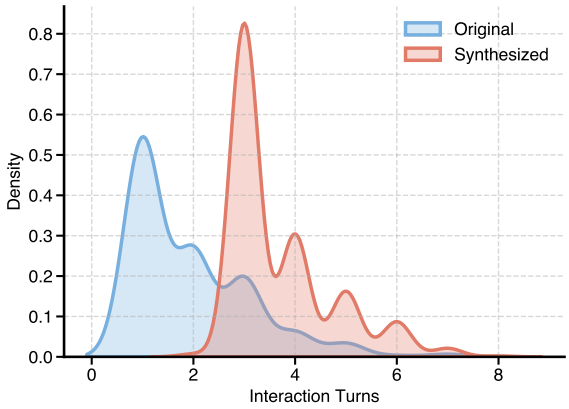

Figure 7: Distribution of interaction turns in BFCL-V3 for original versus synthesized tasks.

A.5 Diversity and Relevance Analysis.

We provide the diversity and relevance metrics used to evaluate the quality of the generated tasks.

Diversity. Diversity is measured using Self-Redundancy@k (SR@k). Specifically, based on sentence embeddings of the synthesized task intents, we compute, for each task, the average cosine similarity to its k nearest neighbors. Lower SR@k indicates less redundancy among tasks and thus higher diversity. The SR@k metric is calculated as follows:

$$\mathrm{SR@}k = \frac{1}{|Y|} \sum_{i} \frac{1}{k} \sum_{j \in \mathrm{kNN}_Y(i)} \langle y_i, y_j \rangle \quad (2)$$

Relevance. We measure relevance using the Relative Energy Distance ($\mathrm{ED}_{\mathrm{rel}}$), which quantifies the distributional discrepancy between the target (ground-truth) task-intent distribution (e.g., human-annotated intents or a predefined target distribution) and the generated task intents. Lower $\mathrm{ED}_{\mathrm{rel}}$ indicates that the generated tasks better match the desired/true task distribution. The relative energy distance is:

$$\mathrm{ED}_{\mathrm{rel}} = \frac{\mathrm{ED}(X, Y)}{\mathbb{E}_{i \neq i'} \|x_i - x_{i'}\|_2} \quad (3)$$

$$\begin{aligned}\mathrm{ED}(X,Y) =& \frac{2}{|X||Y|}\sum_{i=1}^{|X|}\sum_{j=1}^{|Y|}\|x_i - y_j\|_2 \\ &- \frac{1}{|X|^2}\sum_{i=1}^{|X|}\sum_{i'=1}^{|X|}\|x_i - x_{i'}\|_2 \\ &- \frac{1}{|Y|^2}\sum_{j=1}^{|Y|}\sum_{j'=1}^{|Y|}\|y_j - y_{j'}\|_2.\end{aligned} \tag{4}$$

Step	SR (%)	$\mathrm{ED}_{\mathrm{rel}}$ (%)
10	22.57	0.92
20	28.61	3.11
30	16.16	0.76
40	19.11	0.24
50	26.14	1.65
60	21.24	0.79
70	17.91	0.37
80	19.82	0.50
90	22.78	1.20
100	14.76	-0.11
110	22.22	0.46
120	25.92	1.49
Mean	21.44	0.95
Variance	17.30	0.73

Table 15: Trend of SR and $\mathrm{ED}_{\mathrm{rel}}$ Across Steps

Table 15 presents the trends of SR (Self-Redundancy) and $\mathrm{ED}_{\mathrm{rel}}$ (Relative Energy Distance) across training steps. The mean SR is 21.44%, indicating a moderate level of redundancy among synthesized task intents, while the relatively high variance (17.30%) highlights significant fluctuations in redundancy across different steps. Similarly, the mean $\mathrm{ED}_{\mathrm{rel}}$ is 0.95%, reflecting a stable alignment between the generated and target task distributions, with a much lower variance (0.73%) compared to SR. Such observations suggest distinct behaviors of redundancy and distribution alignment within the synthesis process.

A.6 Synthetic Sample Analysis.

We show representative synthetic and validation examples from AppWorld (Fig. 8) and BFCL-V3 (Fig. 6). On BFCL, the original sample reflects a typical short-horizon tool-usage pattern (3 steps: copy → cd → rename) where success is mostly determined by the final state, while the synthetic sample is a longer-horizon BFCL-style task (5 steps) that explicitly requires state construction and correctness verification (create a non-empty file, copy it, then inspect both files to confirm identical content). This increases the number of interaction rounds, tightens step dependencies, and introduces explicit constraints and validation, thereby better stressing multi-round task planning and correctness checking. On AppWorld, the original sample is essentially a single-query information retrieval task wrapped in tool calls: log into Spotify, fetch the currently playing track, look up the artist, and return the artist's follower count—despite multiple API-doc lookups, the logic is mostly linear and the success criterion is one scalar value. In contrast, the synthetic sample is a multi-step state-changing workflow with conditional control: it must authenticate via the supervisor/password flow, fetch the current queue, fetch the user's liked songs, compute a set difference to identify only unliked tracks, and then iterate to like each remaining song. This increases interaction rounds (about 8 vs. 11), introduces cross-endpoint state alignment (queue vs. liked library), adds non-trivial intermediate computation (ID extraction and filtering), and carries higher risk (avoiding duplicate likes), highlighting the greater compositional complexity and longer-horizon execution that synthetic AppWorld tasks are designed to stress.

Across both BFCL and AppWorld, the original samples are mostly linear, short-horizon tasks with simple end-state or single-answer goals, while the synthetic samples require more rounds, stronger cross-step dependencies, intermediate reasoning (e.g., filtering/set-difference), and explicit correctness constraints/verification—therefore better reflecting higher compositional complexity and longer-horizon tool-use.

Module	Prompt Figure
Exploration system/user prompts	Fig. 9
Signal-specific exploration guidance	Fig. 10
Signal-conditioned context summarization	Fig. 11
Task abstraction	Fig. 12
Task validation	Fig. 13

Table 16: Prompt-to-module mapping for feedback loop.

A.7 Prompts Used in the Feedback Loop.

We briefly describe the prompt templates used throughout the exploration and data evolution stages. These prompts define how the exploration model is instructed to generate candidate trajectories, interpret different feedback signals, and transform raw interaction traces into reusable training tasks. Table 16 maps each critical LLM call to its corresponding template.

Specifically, Fig. 9 shows the general prompt templates used for exploration, including the system-side prompt that specifies the role and constraints of the exploration model, and the user-side prompt that provides task context and feedback information. These prompts establish the basic interaction protocol for task proposal.

To specialize exploration toward different failure modes, we further design signal-conditioned prompt templates for three types of training-time feedback signals: forgetting, rare events, and boundary cases, as illustrated in Fig. 10. Each template explicitly conditions the exploration process on the corresponding signal, encouraging the model to generate tasks that target the agent's observed weaknesses.

Fig. 11 shows the prompt used for this signal-conditioned summarization step. Given the signal type and the full trajectory evidence, it extracts a concise recap of the failure case, identifies the likely failure cause or instability pattern, and produces structured fields such as focus patterns, exploration objectives, and "do-not-repeat" constraints. This intermediate representation serves as the bridge between low-level rollout traces and the downstream exploration prompts, ensuring that subsequent exploration is grounded in concrete behavioral evidence rather than loosely conditioned on the signal name alone.

After candidate tasks are proposed, task validation and abstraction are handled by dedicated prompt templates. Fig. 13 presents the prompt used to verify task executability through environment interaction, ensuring that only valid tasks are retained. Fig. 12 shows the abstraction prompt, which converts validated interaction traces into concise and reusable task specifications suitable for training.

A.8 Use of Large Language Models.

During manuscript preparation, we use large language models (LLMs) to (i) improve grammar and spelling without altering the intended scientific content, and (ii) provide lightweight coding assistance (e.g., scripts and formatting help). All reported numerical results, analyses, and claims are produced by the authors. The authors design the methods, conduct the experiments, and verify the findings.

Original Sample

How many people follow the artist of the currently playing song on Spotify?

```
#step1
print(apis.spotify.login(username="allisoncalhoun@gmail.com",password=spotify_password))
#step2
print(apis.spotify.get_now_playing(access_token=spotify_access_token))
#step3
print(apis.api_docs.show_api_descriptions(app_name="spotify"))
#step4
print(apis.api_docs.show_api_doc(app_name="spotify",api_name="show_current_song"))
#step5
print(apis.spotify.show_current_song(access_token=spotify_access_token))
#step6
print(apis.api_docs.show_api_doc(app_name="spotify", api_name="show_artist"))
#step7
print(apis.spotify.show_artist(artist_id=artist_id))
#step8
print(apis.supervisor.complete_task(answer=artist_details["follower_count"]))
```

Synthetic Sample

Like every song currently in my Spotify queue, but only if I have not already liked it before?

```
# step1
print(apis.api_docs.show_app_descriptions())
# step2
print(apis.api_docs.show_api_descriptions(app_name='supervisor'))
# step3
print(apis.supervisor.show_profile())
# step4
passwords = apis.supervisor.show_account_passwords()
spotify_password = next(p["password"] for p in passwords if p["account_name"] == "spotify")
login_result = apis.spotify.login(username='carbrown@gmail.com', password=spotify_password)
spotify_access_token = login_result["access_token"]
# step5
print(apis.api_docs.show_api_descriptions(app_name='spotify'))
# step6
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='show_song_queue'))
# step7
song_queue = apis.spotify.show_song_queue(access_token=spotify_access_token)
# step8
print(apis.api_docs.show_api_doc(app_name='spotify', api_name='show_liked_songs'))
# step9
liked_songs =apis.spotify.show_liked_songs(access_token=spotify_access_token)
# step10
queue_song_ids = {song["song_id"] for song in song_queue}
liked_song_ids = {song["song_id"] for song in liked_songs}
songs_to_like = [song for song in song_queue if song["song_id"] not in liked_song_ids]
for song in songs_to_like:
    apis.spotify.like_song(access_token=spotify_access_token, song_id=song["song_id"])
# step11
apis.supervisor.complete_task()
```

Figure 8: AppWorld cases. Now-Playing Artist Followers Lookup vs. Conditional “Like Queue” with Dedup Filtering.

(a) System Prompt Template for Exploration

You are exploring based on specific guidance to help improve an AI agent's capabilities.

Your Task:
1. Observe the current environment state and identify available actions
2. Analyze available actions and determine which ones will help with the exploration goal
3. Select a relevant action and execute it in the required format
4. Focus on thorough exploration of the targeted area

Action Format:
{action_format}

Instructions:
– Choose only one action at a time
– Carefully read the environment description and task instructions
– Ensure that the action is in the correct format
– Do not use undefined actions
– Always include a valid action and action tags in your reply
– First enter your reason, then enter your action

(b) User Prompt Template for Exploration

{exploration_guidance}

Environment Description:
{initial_obs}

Recent History:
{history_text}

Please select an appropriate action based on the exploration goal and current state.

Figure 9: Prompt templates for exploration: (a) system-side prompt and (b) user-side prompt.

(a) Template for Forgetting Signal Guidance

Exploration Goal: Reinforce Forgotten Skills
The agent previously succeeded but now FAILS on this type of task.
Your exploration should:
1. Practice the exact operations from the context below
2. Create variations with different parameters
3. Connect this skill to related operations
4. Build up from simple to complex usage
Context of forgetting:
{context}
Focus on thorough practice of these specific operations.

(b) Template for Rare Event Signal Guidance

Exploration Goal: Explore Rare Scenarios
The agent encountered a RARE scenario that needs more exposure.
Your exploration should:
1. Explore variations of the scenario below
2. Try different parameter combinations
3. Test edge cases and boundary conditions
4. Collect diverse examples of this rare pattern
Context of rare event:
{context}
Try to discover and document various forms of this scenario.

(c) Template for Boundary Case Signal

Exploration Goal: Explore Boundary Cases
The agent's performance is BORDERLINE (near success/failure threshold).
Your exploration should:
1. Explore boundary conditions for these operations
2. Try similar tasks with slight parameter variations
3. Focus on distinguishing factors between success and failure
4. Collect examples at various difficulty levels
Context of boundary case:
{context}
Focus on understanding what makes the difference between success and failure.

Figure 10: Prompt templates for three types of exploration signals: (a) forgetting, (b) rare event, and (c) boundary case.

Prompt Template for Signal-Conditioned Context Summarization

You are an expert at analyzing trajectory−level failure and behavioral instability for an LLM agent.

Analyze the following feedback signal and trajectory evidence:

Feedback Signal:
− Signal: {signal}

Trajectory Evidence:
{trajectory_context}

Your goal is to extract a structured exploration context that captures:
1) A concise recap of this trajectory
2) Why failure or instability happened
3) Which patterns or behaviors should be re−explored
4) What mistakes should be explicitly avoided

Return a JSON object with this schema:
{
"summary": "concise recap of the current trajectory evidence (task, key actions, and feedback outcome)",
"failure_cause": "1−3 sentence root cause of failure or instability",
"instability_pattern": "1−2 sentence pattern summary",
"focus_pattern": ["pattern or behavior to focus on", "..."],
"exploration_objectives": ["concrete exploration objective", "..."],
"do_not_repeat": ["common mistake to avoid", "..."]
}

Rules:
− Ground every statement in the provided evidence.
− Keep the output concise and actionable.
− First produce `summary` from the trajectory evidence, then derive the other fields from that summary.

Figure 11: Prompt template for signal-conditioned trajectory summarization.

Prompt Template for Task Abstraction

You are a ∗Task Abstraction Expert∗. Your specialty is to inspect an agent's interaction history and distill concrete, goal−oriented tasks from it.

========== YOUR JOB ==========
1. Inspect the interaction tuples (history, action, observation).
2. Identify the specific goal or task the agent is attempting to achieve.
3. Abstract each goal into a clear, concise ∗∗task description∗∗, a ∗∗query∗∗ (suitable for search or training), and the ∗∗minimal action sequence∗∗ that successfully completes the task.

========== ABSTRACTION RULES ==========
– Focus on clear, goal−directed behaviour; ignore purely random exploration.
– Please include as many steps as possible in ActionSequence.
– Group similar behaviour patterns into the same task.
– Every task must have ∗∗at least one∗∗ action sequence that was executed successfully.
– Each task needs an explicit completion criterion.
– All actions listed in an action sequence must be valid and directly executable.
– Ensure all actions are combined into a minimum sequence from initial state to completion.
– The ActionSequence should have at least 3 steps.

========== OUTPUT FORMAT ==========
For every task you identify, output exactly one block in the form below:
{output_format}

Figure 12: Prompt template for task abstraction.

Prompt Template for Task Validation

You are a strict task evaluation expert. Your goal is to determine whether the following multi−step agent trajectory successfully completed the assigned task.

Task Details
− Task Description: {task_description}
− Query: {query}
− Expected Outcome (API Call or Result): {ground_truth}
− Action Modality: {modality_hint}

Execution Summary
− Trajectory Summary:
{trajectory_summary}

− Final Observation: {final_observation}

Evaluation Instructions

Carefully analyze the trajectory to determine if the task was truly completed. Specifically, consider the following aspects:

1. **API Matching**: Did the agent correctly call the required APIs according to the task requirements?
2. **Parameter Usage**: Were the parameters used in API calls correct and sufficient?
3. **Logical Flow**: Was the sequence of steps logical without unreasonable skips?
4. **Final Result**: Did the final state achieve the expected outcome, reasonably solve the task, obtain all necessary information, and complete the task objectives?
5. **Failed or Skipped Steps**: Were there any critical errors, skipped steps, or invalid code that prevented the task from being actually executed?

Format Your Response Strictly As:

Success: [true/false]
Reason: [Concise and specific explanation, referring to the above criteria.]

Note: Ignore all Connection timeout or No valid action, because it is very likely that it is the former. Do NOT mark the task as successful if the correct API was never called, the parameters were incorrect, or the result was not achieved, even if the intent seemed right.

Figure 13: Prompt template for task validation.