

EAU: ENVIRONMENT-AGENT-USER CO-EVOLUTION FOR TRAINING USER-CENTRIC AGENTS

Min Cai^{1,*} Dan Zhang^{2,*} Gao Xing³ Libin Ma³ Zhaohui Xu⁴ Xi Ye^{1,5}

¹University of Alberta ²National University of Singapore ³Kuaishou Technology

⁴Independent Researcher ⁵Alberta Machine Intelligence Institute (Amii)

*Equal contribution {mcai8, xi.ye}@ualberta.ca

 eau-meta-agent.github.io

ABSTRACT

Training LLM agents to interact with users over multiple turns is challenging because the learning signal depends not only on the agent, but also on the user simulator and environment, which jointly shape the agent’s experience. Simulated users can be stochastic or inconsistent, while environments may contain unreliable tools, incomplete state coverage, and incorrect verifiers. Prior work mainly addresses the resulting instability through reward shaping or denser supervision, while treating these components as fixed. We instead introduce **EAU**, a meta-agent framework for environment-agent-user co-evolution. During training, **EAU** monitors trajectories, tool traces, rewards, and training statistics; diagnoses failures and uninformative interactions; and proposes targeted modifications to the user simulator, environment, reward design, and RL configuration. Validated changes are hot-swapped into the ongoing run, allowing the training setup to evolve together with the policy. These adaptations also persist across runs, so later training can build on previously discovered improvements rather than restarting from a fixed setup. We enable this process through an editable symbolic representation of user and environment dynamics, while the agent remains a neural LLM optimized with RL. Across three interactive-agent benchmarks and two backbone models, **EAU** consistently outperforms prior RL methods, improving the overall score over the strongest baseline by 14.7–15.0% and increasing successful task completion on ColBench by up to 36.7%, showing that co-evolving the training system with the policy can substantially improve multi-turn RL.

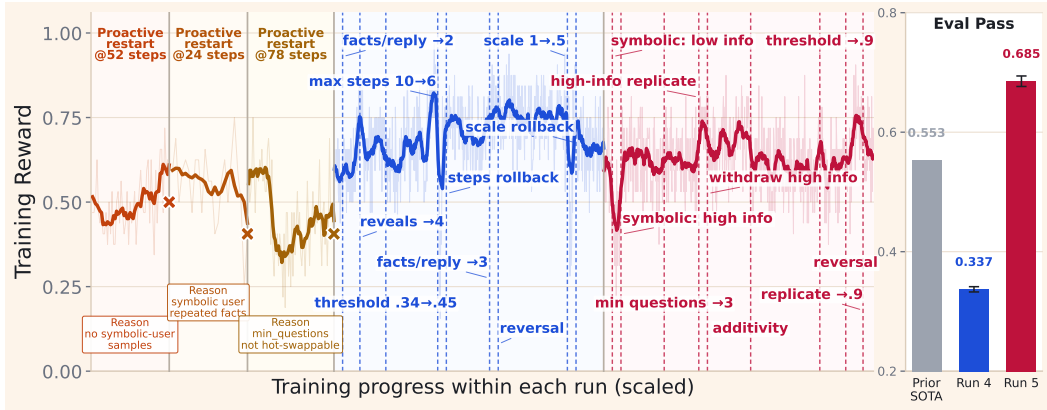


Figure 1: Illustration of how our meta-agent, **EAU**, manages RL training runs. **EAU** surpasses the prior SOTA on ColBench. **Left:** For the first three runs, **EAU** proactively restarts when it identifies issues that can only be resolved with offline patches. **Middle:** Run 4 and 5 displayed in blue and red are two full runs. Dashed lines indicate online, intra-run modifications. Run 4 explored individual modifications through local perturbations, whereas Run 5 used endpoint interventions followed by replication, reversal, and interaction tests.

1 INTRODUCTION

Large language models are increasingly deployed as agents that collaborate with humans rather than simply execute fully specified tasks. As shown in Figure 2, in many realistic applications, users can provide incomplete or evolving specifications, requiring an agent to interact over multiple turns: asking clarifying questions, resolving ambiguities, negotiating constraints, and adapting its behavior as new information emerges. This has motivated a growing body of interactive benchmarks in which agents communicate with simulated users while acting in an external environment (Yao et al., 2024; Barres et al., 2025; Qian et al., 2025a). Such simulation makes large-scale evaluation and reinforcement learning (RL) possible, but it also introduces a fundamental challenge: *the learning signal is only as reliable as the interaction that produces it*. LLM-based user simulators are stochastic and can behave inconsistently, for example by drifting from their assigned persona, revealing information prematurely, or changing their intent across otherwise similar episodes. At the same time, environments may contain incomplete state coverage, unreliable tool behavior, unreachable tasks, or incorrect verifiers. Moreover, because both components are typically fixed throughout training, the interactions they generate need not remain informative as the policy evolves. Together, these issues can make rewards highly variable or systematically misleading, substantially destabilizing RL for interactive agents.

Prior work has largely addressed this instability by improving the supervision presented to the policy, for example through reward shaping that credits partial progress (Qian et al., 2025b) or auxiliary supervision that provides denser learning signals (Kong et al., 2026; Wang et al., 2026). These approaches can make better use of a noisy signal, but they leave the sources of that signal—the user simulator and the environment—largely unchanged. We argue that this misses an important root cause of instability. If a tool silently fails for a particular argument, a verifier incorrectly rejects a successful trajectory, or a simulated user reveals information that the agent was supposed to elicit, modifying the downstream reward cannot recover the intended interaction. More generally, as the policy improves, a fixed simulator and environment may continue producing trajectories that are redundant, uninformative, or poorly targeted to the policy’s current weaknesses. We therefore propose shifting part of the intervention upstream: rather than treating the user simulator and environment as fixed infrastructure and optimizing only the policy’s learning signal, we continually diagnose and revise the interaction process itself during training.

To realize this idea, we introduce **EAU**, a meta-agent that continuously supervises RL training and improves the components that generate its learning experience. Throughout training, **EAU** observes rollout transcripts, tool traces, rewards, and aggregate training statistics, and diagnoses whether problematic or uninformative trajectories arise from the policy, the user simulator, the environment, or the training configuration. As shown in Figure 1, it proposes targeted interventions based on this diagnosis, including correcting environment or verifier failures, refining the simulator’s persona and information-disclosure behavior, and adjusting reward terms or training hyperparameters. Importantly, these interventions are not limited to fixing errors. As the policy improves, **EAU** can also adapt the simulator and environment to expose unresolved weaknesses and generate more informative training trajectories. Candidate modifications are validated by a bounded integrity gate before being accepted and hot-swapped into the ongoing training loop, allowing the training system itself to evolve without repeatedly restarting from scratch.

This produces a different view of interactive-agent training: rather than optimizing a policy against a fixed user and environment, **EAU** enables the training system to adapt alongside policy learning. Instead of prescribing a fixed sequence of component updates, the meta-agent diagnoses emerging problems and determines what to modify and when to intervene. Simulator revisions, reward and curriculum adjustments, hyperparameter changes, and implementation repairs are therefore part of

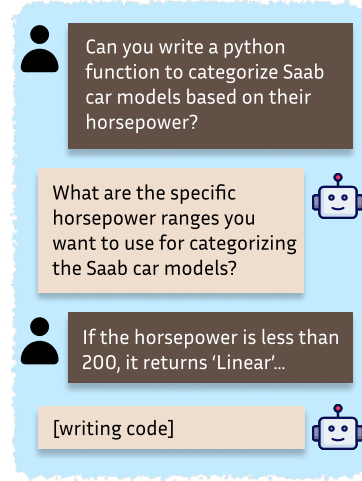


Figure 2: Illustration of a user-centric agentic task. An agent needs to ask for clarifications about a user query.

the intervention space. The contribution lies in enabling and organizing this adaptive process during and across training runs.

We evaluate **EAU** on three interactive-agent benchmarks and two backbone models. Across these settings, **EAU** discovers interpretable modifications to both simulators and environments and improves final agent performance by up to 36.7% over the previous state of the art, while our analysis shows how the three components evolve together throughout training.

To summarize, our contributions are:

- We identify fixed user simulators and environments as a key bottleneck in interactive RL.
- We introduce **EAU**, a meta-agent that diagnoses and updates the training system online.
- We demonstrate improved agent performance on three user-centric benchmarks, with targeted analyses of symbolic-user mixing, intra-run intervention, and alternative training algorithms and meta-agent implementations.

2 RELATED WORK

User Centric Agentic Reinforcement Learning. User-centric tasks (Yao et al., 2024; Barres et al., 2025; Qian et al., 2025a) focus on interacting and fulfilling the requirement of a simulated user under multi-turn conversations. Different from previous multi-turn Reinforcement Learning (Jimenez et al., 2024; Xi et al., 2025; Zhang et al., 2026b) that solely interacts with a fixed deterministic environment, user-centric RL (Qian et al., 2025b; Kong et al., 2026; Yu et al., 2026) poses an additional challenge: the introduction of LLM user simulator may cause several issues such as stochastic state transition, inconsistent behaviors and collapse in training due to overfitting a fixed behavior (Yu et al., 2026). Recent methods mainly focus on improving supervision signal presented to the policy (Zhou et al., 2025; Kong et al., 2026). Yu et al. (2026) co-train the user to avoid overfitting. Our work identifies that issues may lie in all three parts, i.e., RL training dynamics, user simulation and environment, and we endeavor to address all at once with a meta agent, **EAU**.

Meta-Agents and Automated Optimization of Training Components. A growing body of work uses one LLM to optimize another system or the system in which it operates (Hu et al., 2025; Lee et al., 2026; Zhang et al., 2026a; Liu et al., 2026; Chen et al., 2026b; Surina et al., 2025). **EAU** differs mainly in that the meta-agent acts *within* a run: modifications are validated and hot-swapped into a live training loop rather than applied between runs or before training begins.

Co-evolution of Agent and its World. More recently, there’s a trend in improving the world, e.g., environment and user simulator, that an agent is interacting with. While most of them prepare the worlds before training (Huang et al., 2026; Tu et al., 2026), some begin to discover how the world can evolve along with the agent training (Yu et al., 2026; Kim et al., 2026; Yang et al., 2026). Unlike prior methods that evolve the world through predefined update rules or workflows, **EAU** can flexibly determine what to modify, when to intervene, and how to adapt the training system online.

3 PROBLEM SETUP

We focus on user-centric agentic tasks, where an agent interacts with both an environment through tool calls and a (simulated) user over multiple turns. The goal is to accomplish a task while satisfying the user’s intent, which may be underspecified at the outset and gradually refined through the interaction.

We formulate each task in our setting as a partially observable Markov decision process (POMDP). Formally, the environment for a task is $\mathcal{E} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{Z}, \mathcal{O} \rangle$ with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S} \times \mathcal{O})$, reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, observation space $\mathcal{Z}$, and a function $\mathcal{O} : \mathcal{S} \rightarrow \Delta(\mathcal{Z})$ that maps states to observations. Δ here refers to the stochasticity in transitions and observations. We consider each task to have a simulated user as a part of the environment, therefore an agent needs to interact with the environment by tool calling and the user simulator by conversations, i.e., $\mathcal{S} = \mathcal{S}_{\text{tool}} \otimes \mathcal{S}_{\text{user}} \otimes \mathcal{S}_{\text{query}}$, where $\mathcal{S}_{\text{query}}$ can be considered as the first state. And we have $\mathcal{A} = \mathcal{A}_{\text{tool}} \cup \mathcal{A}_{\text{user}}$, $\mathcal{Z} = \mathcal{Z}_{\text{tool}} \cup \mathcal{Z}_{\text{user}}$. Specifically, for transition $\mathcal{T}$, interactions

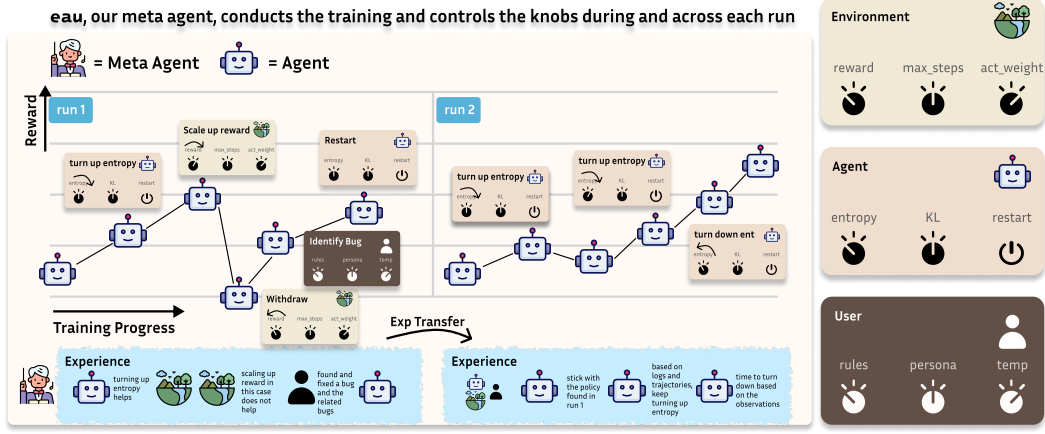


Figure 3: Illustration of how **EAU**, our meta agent, explores, learns, and absorbs experience online during and across each run. Reading logs and trajectories, **EAU** explores different strategies online by controlling the *knobs* of each module, learns which strategies help and which do not. Experience is brought across different runs, serving as a solid foundation for continual exploration and learning.

with the environment through tool calls are deterministic, while interactions with simulated users are stochastic. Thus, we have $\mathcal{T}_{\text{tool}} : (s_{\text{tool}}, a_{\text{tool}}) \mapsto (s'_{\text{tool}}, z_{\text{tool}})$ and $\mathcal{T}_{\text{user}} : (s_{\text{user}}, a_{\text{user}}) \rightarrow (\mathcal{S}_{\text{user}}, \mathcal{Z}_{\text{user}})$.

Let π_{θ} be the policy (agent), our goal is to maximize the number of successful tasks, i.e., $\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^T R(s_t, a_t) \mid \text{query} \right]$. To achieve this, a policy needs to first interact, negotiate, and even persuade the user, while at the same time understanding users' intents and calling the correct tools. Thus, we term it a bi-level optimization:

$$\max_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R_{\text{task}}(\tau, u)] \quad \text{s.t.} \quad \pi_{\theta} = \arg \max_{\pi} \mathbb{E}_{\tau' \sim \pi} \left[\sum_{t=0}^{T-1} \gamma^t r_{\text{conv}}(s_t, a_t) \right], \quad (1)$$

where R_{task} refers to rewards related to task success, and r_{conv} refers to rewards related to conversations specifically designed for each environment. Notably, R_{task} is usually verifiable yet sparse while r_{conv} can be dense but noisy.

4 METHOD: ENVIRONMENT-AGENT-USER CO-EVOLUTION

In this section, we introduce **EAU** (Environment-Agent-User Co-Evolution) for user-centric agentic RL. Recall that agentic RL with simulated users presents several challenges. First, sparse rewards and stochastic interactions can make learning difficult. Second, user simulators may provide unreliable rewards, deviate from instructions, or leak answers during conversations.

4.1 METHOD OVERVIEW

The core idea of **EAU** is to couple policy learning with adaptation of the system that generates its training experience. The agent is updated through RL, while a meta-agent uses training trajectories and optimization statistics to revise the user simulator, environment, and configurable training dynamics. These revisions change the interactions and learning signals encountered by subsequent policy updates, closing the feedback loop between policy behavior and training-system adaptation.

We use *co-evolution* to denote this meta-agent-mediated, asymmetric co-adaptation. It does not require the environment and user simulator to be independently learned models: their adaptation is implemented through programmatic interventions. As illustrated in Figure 3, the meta-agent observes training feedback, selects interventions through editable interfaces, and accumulates experience for subsequent decisions. The following subsections describe the user-, environment-, and agent-side interventions, followed by their coordination within and across training runs.

4.2 EVOLVING USER

LLM-simulated users are inherently stochastic and may imperfectly follow its specification, for example by revealing information that should remain hidden or by improvising answers to questions unsupported by the task. To provide a more controllable signal, we introduce a *symbolic user simulator*, u_{sym} , which consists of a set of explicit rules that deterministically map interaction states, obtained from parsed agent outputs, to user responses. Figure 4 illustrates an example of such a rule. These rules provide precise control over what information is revealed and under which conditions. However, designing and updating them manually throughout training is intricate and labor-intensive. We therefore expose the symbolic rules as tunable *knobs*, allowing the meta-agent to revise them in response to the agent’s evolving behavior.

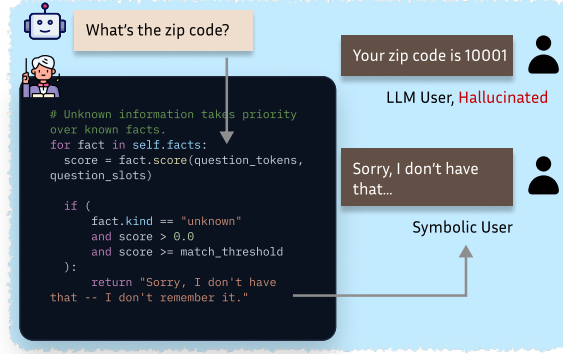


Figure 4: Illustration of symbolic user simulator.

The symbolic and LLM simulators offer complementary strengths: the former provides controllability and determinism, while the latter offers richer and more natural interactions. We combine them through an ϵ -symbolic strategy:

$$u_{\epsilon} := \begin{cases} u_{\text{sym}}, & \text{with probability } \epsilon, \\ u_{\text{LLM}}, & \text{with probability } 1 - \epsilon, \end{cases} \quad \epsilon \in [0, 1]. \quad (2)$$

The mixing coefficient ϵ thus controls the balance between structured, deterministic supervision and flexible LLM-based interaction, and can itself be adapted by the meta-agent during training.

Beyond the symbolic rules and mixture coefficient ϵ , **EAU** can also evolve the LLM user by tuning controllable generation parameters, including temperature and persona-specific prompt instructions. Together, these mechanisms allow the user side of the training loop to adapt alongside the policy rather than remaining fixed throughout training.

4.3 EVOLVING ENVIRONMENT

Environment-side adaptation in **EAU** modifies the training feedback and supported interaction conditions, rather than learning a new world model. In our implementation, this primarily involves adding or revising intermediate reward signals in R , together with benchmark-specific interaction constraints such as episode length. Recall that user-centric agentic tasks typically involve multi-turn interactions, where verifiable rewards are often sparse, such as task success. As a result, the agent may struggle to explore effectively when it rarely reaches successful outcomes. When such behavior is observed, **EAU** can introduce intermediate reward rules that identify and reward states indicative of progress toward task completion. An illustrative example is shown in Figure 5.

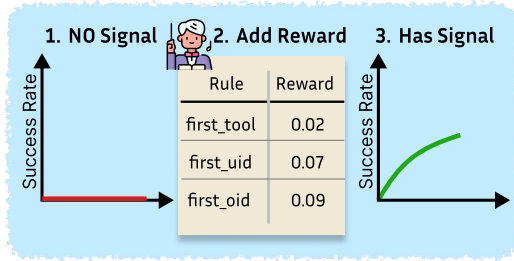


Figure 5: Illustration of environment evolution by introducing additional rewards.

EAU can also revise existing manually designed rewards when they induce undesirable behaviors or divert the agent from the final objective. By dynamically adding and modifying intermediate rewards, **EAU** provides more informative learning signals while also accumulating experience about

which reward designs facilitate or hinder learning. This experience can in turn guide future environment evolution.

4.4 EVOLVING AGENT

The agent naturally evolves through gradient updates during RL training. We keep the underlying *training algorithm fixed* and let the meta-agent continuously monitor the optimization process and intervene in its configurable training dynamics when necessary.

Training Algorithm. We use and fix Proximal Policy Optimization (Schulman et al., 2017, PPO) as the RL algorithm. Recent work on LLM RL mainly use critic-free methods, such as GRPO (Shao et al., 2024), for low training cost and unbiased advantage estimation. However, current critic-free methods suffer from the sparse credit assignment, where intermediate rewards are accumulated to a single trajectory-level reward (Qian et al., 2025b), or are not applicable to user-centric tasks where the same states are hard to replicate for group advantage estimation (Feng et al., 2026).

We therefore use a learned critic V_ϕ , parameterized by ϕ , with Generalized Advantage Estimation (GAE). The critic is optimized by minimizing the value error $\mathcal{L}_\phi^{\text{VF}} = \mathbb{E}[(V_\phi(q, y_{<t}) - R)^2]$, where R is the cumulative reward. Both critic and policy are restricted to action tokens: the critic is evaluated only where the policy acts, and GAE is computed across actions rather than across all tokens, so the advantage of a turn is not diluted by the observation text surrounding it (Hou et al., 2026). Formally, the advantage $\hat{A}_t^{\text{GAE}}$ is computed as:

$$\hat{A}_t^{\text{GAE}} = \sum_{l=0}^{|y|-t-1} (\gamma\lambda)^l \delta_{t+l}, \quad (3)$$

where $\delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$, and s_t refers to the t -th action token.

Monitoring Training During training, the meta agent is allowed to tune the hyper-parameters such as entropy coefficient and KL coefficient, based on training logs, including metrics like estimated rollout entropy and log likelihood. It can also read the rollout trajectories and the interactions with the environment and the user.

4.5 INTRA- AND INTER-RUN INTERVENTION

A notable design choice of **EAU** is to support flexible intervention at two timescales: both *intra-run* and *inter-run*. As illustrated in Figure 3, for intra-run intervention, updates can be hot-swapped into the training loop without restarting from scratch. For example, **EAU** can revise symbolic-user rules, add or modify environmental rewards, and adjust RL hyperparameters during training.

It can also decide when a restart (inter-run) is necessary, such as after an irreversible failure, or resume from an intermediate checkpoint. Inter-run intervention, in contrast, enables **EAU** to consolidate observations from completed runs and use them to revise the setup for subsequent training.

Unlike previous paradigm that only allows inter-run modifications (Chen et al., 2026a; Xia et al., 2026; Liu et al., 2024; Yamada et al., 2025; Ferreira et al., 2026), by granting this flexibility, **EAU** can explore different operations on-the-fly, which is more efficient than inter-run interventions alone. Therefore, it can find out more effective operations based on the intra- and inter-run experience.

4.6 INSTANTIATION OF META-AGENT

We instantiate our meta agent with coding agents, such as Claude Code and Codex, following recent works on meta agent (Lee et al., 2026). The meta agent has access to a local filesystem which includes the source code, training logs, etc. It reads the logs with a fixed interval during training, and decides which part to modify for environment-user-agent evolution. A list of *knobs* to tune is prepared at the beginning, but **EAU** can also modify the list if necessary. It is also asked to maintain a experience pool as a directory of documents in the filesystem. Before each modification, it may reference to the experience pool and whenever there’s a new finding, it updates the pool accordingly.

Table 1: Comparison across UserGym and ColBench on *Qwen3-4B* and *Qwen2.5-7B-Instruct*. The best and second-best scores in each category are highlighted in **bold** and underlined respectively. Standard deviations across 3 repeated evaluations are shown in the subscripts.

Type	Method	UserGym								ColBench			Overall
		ID				OOD				Avg.	Pass	Succ.	
Travel	Func.	Pers.	Tau	Turtle	Inten.	Tele.							
Closed-Source Models													
Prompting	Gemini-3-Flash	0.574	0.423	0.695	0.167	0.153	1.718	0.829	0.651	0.515	0.382	0.449	0.606
Prompting	GPT-4.1	0.554	0.051	0.599	0.109	0.267	1.867	0.732	0.597	0.529	0.403	0.466	0.568
Prompting	GPT-4o-mini	0.596	0.089	0.683	0.091	0.117	1.277	0.512	0.481	0.463	0.342	0.403	0.463
Qwen3-4B													
Prompting	Qwen3	0.277	0.026	0.452	0.006	0.071	1.660	0.488	0.426	0.272	0.153	0.213	0.378
Prompting	ReAct	0.279	0.053	0.476	0.005	0.147	1.782	0.454	0.457	0.269	0.145	0.207	0.401
Prompting	Reflexion	0.291	0.045	0.474	0.035	0.124	1.717	0.501	0.455	0.303	0.184	0.244	0.408
RL Training	RAGEN	0.477	0.104	0.511	0.006	0.117	1.572	0.488	0.468	0.479	0.361	0.420	0.457
RL Training	Search-R1	0.482	0.103	0.425	0.059	0.113	1.711	0.412	0.472	0.467	0.355	0.411	0.459
RL Training	UserRL	0.538	0.095	0.507	0.053	0.121	1.732	0.520	0.509	0.468	0.342	0.405	0.486
RL Training	InfoPO	0.589	0.115	0.556	0.097	0.154	1.862	0.542	0.559	0.553	0.439	0.496	0.545
RL Training	PPO	0.468	0.051	0.560	0.006	0.180	1.880	0.512	0.522	0.284 $\pm$ 0.046	0.200 $\pm$ 0.044	0.242	0.462
RL Training	Ours	0.615	0.186	0.726	0.109	0.188	2.023	0.512	0.623	0.685 $\pm$ 0.009	0.600 $\pm$ 0.015	0.643	0.627
Qwen2.5-7B-Instruct													
Prompting	Qwen2.5	0.441	0.026	0.289	0.000	0.062	1.254	0.292	0.338	0.242	0.140	0.191	0.305
Prompting	ReAct	0.452	0.064	0.325	0.037	0.078	1.378	0.316	0.379	0.238	0.135	0.187	0.336
Prompting	Reflexion	0.445	0.052	0.312	0.022	0.074	1.320	0.305	0.361	0.276	0.168	0.222	0.330
RL Training	RAGEN	0.538	0.124	0.548	0.000	0.148	1.815	0.416	0.513	0.449	0.348	0.399	0.487
RL Training	Search-R1	0.565	0.113	0.412	0.043	0.154	1.805	0.416	0.501	0.457	0.352	0.405	0.480
RL Training	UserRL	0.546	0.115	0.444	0.048	0.152	1.826	0.424	0.508	0.436	0.327	0.382	0.480
RL Training	InfoPO	0.588	0.167	0.535	0.091	0.178	1.892	0.488	0.563	0.534	0.426	0.480	0.544
RL Training	PPO	0.596	0.229	0.651	0.018	0.135	1.860	0.488	0.568	0.356 $\pm$ 0.012	0.240 $\pm$ 0.000	0.298	0.508
RL Training	Ours	0.595	0.333	0.730	0.048	0.170	2.035	0.512	0.632	0.659 $\pm$ 0.017	0.533 $\pm$ 0.018	0.596	0.615

Unlike previous methods that allow the meta-agent to collect feedback on an evaluation/test set (Panfilov et al., 2026; Lee et al., 2026; Xia et al., 2026), **EAU** uses only training-side feedback to guide its interventions. The workflow described above is implemented as a skill comprising instructions and supporting scripts.

5 EXPERIMENTS

5.1 EVALUATION PROTOCOL

We evaluate **EAU** on three user-centric benchmarks. Experiments on UserGym and ColBench use two policy backbones, *Qwen3-4B* (Yang et al., 2025) and *Qwen2.5-7B-Instruct* (Qwen et al., 2025), while the τ^2 -bench experiments use *Qwen3-4B*. We follow the benchmark evaluation protocol of Kong et al. (2026).

Benchmarks. We consider **ColBench** (Zhou et al., 2025), **UserGym** (Qian et al., 2025b), and τ^2 -**bench** (Barres et al., 2025). These benchmarks require agents to uncover hidden information through conversation and tool calls to fulfill users’ requests. Benchmark details are provided in Appendix B.

Baselines. We compare against prompting methods (ReAct (Yao et al., 2022) and Reflexion (Shinn et al., 2023)) and RL training methods (RAGEN (Wang et al., 2025b), Search-R1 (Jin et al., 2025), UserRL (Qian et al., 2025b), and InfoPO (Kong et al., 2026)). We also include vanilla PPO. For UserGym and ColBench, baseline results other than PPO are taken from Kong et al. (2026). A generic meta-agent baseline is additionally examined in our ablations.

User simulator. The default training-time LLM simulator is *CoSER* (Wang et al., 2025a), a fine-tuned *Llama-3.1-8B* model. This choice is motivated by its competitive user-simulation performance

reported by Zhou et al. (2026). During training, **EAU** may replace the simulator when justified by the training logs. For evaluation, we use GPT-4o-mini, following Kong et al. (2026).

Evaluation and model selection. Evaluation is performed after each completed training run. We report the final run and its final checkpoint for each experiment, without using final-test feedback for selection. Evaluation instances, trajectories, scores, and derived summaries are not used to propose or accept interventions, update the experience pool, or guide subsequent training runs. Training-side modifications do not alter the evaluation task set, user-simulator configuration, interaction limits, or scoring rules.

5.2 IMPLEMENTATION DETAILS

Meta-agent backbone. Unless otherwise specified, we instantiate **EAU** with Claude Code using Claude Opus-5. Table 5 examines an alternative meta-agent implementation.

Training and search budgets. The default episode limits are 16 turns on UserGym, 10 on ColBench, and 50 on τ^2 -bench, with a per-turn generation budget of 1024 tokens and per-episode rollout budgets of 8192, 8192, and 16384 tokens, respectively. For each experiment, we cap the number of inter-run iterations at 5, including proactive restarts. Remaining optimization settings are provided in Appendix C. These limits specify the rollout and search setup, rather than a matched end-to-end compute budget; the latter would also need to account for meta-agent inference and all attempted training runs.

5.3 MAIN RESULTS

As shown in Table 1, **EAU** achieves the highest overall score among the evaluated RL methods on both policy backbones. Compared with InfoPO, the strongest prior RL baseline by overall score, **EAU** improves from 0.545 to **0.627** on *Qwen3-4B* and from 0.544 to **0.615** on *Qwen2.5-7B-Instruct*, corresponding to relative gains of 15.0% and 14.7%, respectively. With a 4B policy backbone, **EAU** also achieves a higher overall score than the evaluated closed-source models, including Gemini-3-Flash (0.606) and GPT-4.1 (0.568). These results show consistent overall improvements on UserGym and ColBench across the two tested Qwen backbones, which differ in model size and generation.

Table 2: Results on τ^2 -bench (*Qwen3-4B*).

Method	Retail	Airline	Telecom	Avg.
UserRL	16.9	7.5	13.8	13.8
InfoPO	24.4	10.0	15.6	18.0
PPO	16.3	23.8	18.1	18.5
EAU	18.1	28.8	21.3	21.5

UserGym. RL training generally provides larger gains than the tested prompting methods. Relative to InfoPO, **EAU** raises the average UserGym score from 0.559 to **0.622** on *Qwen3-4B* (+11.3%) and from 0.563 to **0.620** on *Qwen2.5-7B-Instruct* (+12.3%). On *Qwen3-4B*, **EAU** outperforms the best RL baseline in six of the seven evaluated environments, with particularly large gains on Func., Pers., and Inten. Inten. and Tele. are held out during training and therefore assess cross-environment generalization. On Inten., **EAU** reaches **2.023**, exceeding InfoPO (1.862) and the evaluated closed-source models. On Tele., it scores 0.512, slightly below InfoPO (0.542). These results provide evidence of transfer to unseen environments, although the gains are not uniform across them.

ColBench. On *Qwen3-4B*, **EAU** improves over InfoPO from 0.553 to **0.685** on Pass (+23.9%) and from 0.439 to **0.600** on Succ. (+36.7%), increasing the ColBench average from 0.496 to **0.643** (+29.6%). The same pattern holds for *Qwen2.5-7B-Instruct*, where Pass improves from 0.534 to **0.659** (+23.4%) and Succ. from 0.426 to **0.533** (+25.1%). For both backbones, the relative gains are larger on the stricter Succ. metric, indicating that the improvement extends beyond the more permissive Pass criterion.

Table 3: Ablation on the user simulator. Results are on ColBench with *Qwen3-4B*.

Variant	Pass mean	Succ. mean
LLM	0.284 $\pm$ 0.046	0.200 $\pm$ 0.044
Symbolic	0.229 $\pm$ 0.004	0.177 $\pm$ 0.012
Hybrid	0.294 $\pm$ 0.019	0.213 $\pm$ 0.032

Table 4: Comparison of PPO without meta-agent, naïve meta-agent without **EAU**, and **EAU**.

Method	UserGym	ColBench	
	Avg	Pass	Succ.
<i>Qwen3-4B</i>			
PPO	0.525	0.284 $\pm$ 0.046	0.200 $\pm$ 0.044
Meta-Agent	0.574	0.317 $\pm$ 0.038	0.207 $\pm$ 0.035
Ours	0.622	0.685$\pm$0.009	0.600$\pm$0.015
<i>Qwen2.5-7B-Instruct</i>			
PPO	0.568	0.356 $\pm$ 0.012	0.240 $\pm$ 0.000
Meta-Agent	0.512	0.415 $\pm$ 0.007	0.277 $\pm$ 0.006
Ours	0.620	0.659$\pm$0.017	0.533$\pm$0.018

Table 5: Ablations on training algorithm and meta-agent backbone. Results are on τ^2 -bench.

Method	Retail	Airline	Telecom	Avg.
UserRL	16.9	7.5	13.8	13.8
+ EAU	22.5	15.0	21.9	20.8
InfoPO	24.4	10.0	15.6	18.0
+ EAU	20.0	16.3	24.4	21.0
+ EAU (Codex)	20.0	13.8	20.0	18.8

τ^2 -**bench**. Table 2 compares **EAU** with UserRL, InfoPO, and PPO on *Qwen3-4B*. **EAU** achieves the highest aggregate score, improving from 18.5 with PPO to **21.5** (+16.2%). It achieves the highest scores on Airline and Telecom, whereas InfoPO performs best on Retail.

5.4 ABLATIONS & ANALYSIS

We examine symbolic-user mixing, the EAU-specific intervention design, alternative training algorithms and meta-agent implementations, and intra-run adaptation. Our analysis focuses on capabilities that support autonomous training intervention, rather than treating every editable component as an independently beneficial module.

Ablation on symbolic user. Table 3 compares LLM, symbolic, and hybrid user simulators on ColBench. For Hybrid, we set the mixing coefficient to $\epsilon = 0.5$. The hybrid simulator achieves slightly higher mean Pass and Succ. scores than the LLM-only simulator, whereas the symbolic-only variant performs worse. One possible explanation is that mixing more constrained symbolic responses with varied LLM responses preserves interaction diversity while providing more controlled feedback.

Ablation on EAU design. Table 4 compares **EAU** with a generic meta-agent constructed by removing the symbolic user and the skill for intra-run operations over the user, environment, and agent. This baseline receives only a generic instruction to improve RL training. Compared with PPO, it produces mixed results, including a decrease in UserGym performance on *Qwen2.5-7B-Instruct*. **EAU** outperforms this baseline on both benchmarks and both policy backbones. Because the symbolic user and intervention skill are removed together, this comparison evaluates the combined design rather than their individual effects.

Analysis: Alternative training algorithms and meta-agents. Table 5 shows that **EAU** improves aggregate performance when combined with either UserRL or InfoPO on τ^2 -bench. The observed gains are therefore not restricted to the default PPO setup. Replacing the default meta-agent with Codex (*GPT-5.6-Sol*) also improves over InfoPO alone, although the gain is smaller than with the default meta-agent. These results demonstrate compatibility with multiple training and meta-agent configurations, with gains of different magnitudes across the tested configurations.

Analysis: Intra-run exploration and experience reuse. We examine how intra-run intervention supports exploration of training modifications and reuse of the resulting experience. On ColBench, **EAU** proactively restarts its first three runs after detecting severe issues (Figures 1 and 6). It then explores multiple modifications in run 4 and carries these observations into run 5, the second completed run, which surpasses the strongest prior baseline. The generic inter-run meta-agent makes smaller gains over its run sequence. This progression illustrates how intra-run feedback can inform both immediate changes and subsequent training runs.

Figure 7 further examines experience reuse on UserGym. Given the same experience from the first four runs, enabling intra-run intervention in run 5 yields a larger improvement than inter-run adaptation alone. This supports the benefit of intra-run intervention in the illustrated setting. The curves show progression by run count, rather than an efficiency comparison under matched total compute.

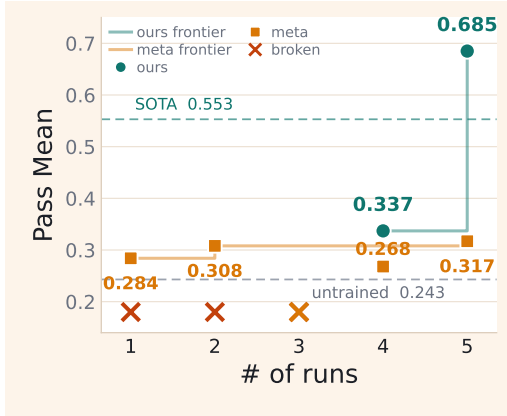


Figure 6: Best-score frontiers on ColBench for **EAU** and the generic meta-agent. Each frontier tracks the best score observed so far and remains flat after unsuccessful or lower-scoring runs.

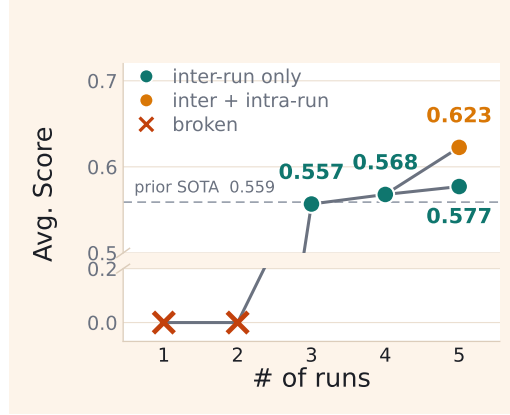


Figure 7: Run-level progression on UserGym with *Qwen3-4B*. The two conditions share experience from the first four runs; in run 5, one uses inter-run adaptation only, while the other additionally permits intra-run intervention.

6 CONCLUSION

We introduced **EAU**, a meta-agent framework that jointly evolves the environment, agent, and user during training. We show that a frontier coding agent can effectively manage and adapt these components based solely on training feedback, including trajectories and optimization logs. To provide finer control over user behavior, we introduce a symbolic user simulator that the meta-agent can modify programmatically. We further enable intra-run interventions, allowing **EAU** to explore and exploit training modifications more flexibly. Across three user-centric benchmarks, **EAU** improves upon prior state-of-the-art performance by up to 36.7%. This highlights the value of adapting the training system itself. Future work may investigate post-training open-source models to perform such co-evolution autonomously.

ETHICS STATEMENT

Our study concerns RL agents interacting with simulated users in benchmark environments; it does not recruit human participants or collect new human-participant data. **EAU** can modify training code, rewards, and user-simulator behavior, creating risks of implementation errors, misleading learning signals, and unintended task-information disclosure. The documented interface checks constrain admissible updates but do not guarantee behavioral correctness. Training-time adaptation is separated from a fixed evaluation protocol, and final-test feedback is not used to guide interventions or select reported models. These controls address experimental integrity rather than establish deployment safety. Performance in simulated benchmarks should not be interpreted as evidence of safe or reliable behavior with real users.

REPRODUCIBILITY STATEMENT

Section 4 describes the framework and meta-agent workflow. Section 5 specifies the benchmarks, model configurations, baselines, evaluation and model-selection protocol, and training and search limits. Appendix A documents the intervention interfaces, benchmark-specific support, and recorded intervention outcomes. Additional benchmark and optimization details are provided in Appendices B and C. We report standard deviations across repeated evaluations where available; these do not by themselves quantify variability across independent repetitions of the full training and adaptation process. Reproducing the reported configurations requires access to the external coding-agent and user-simulator models used in the experiments.

AI USE STATEMENT

Generative AI is a functional component of the proposed method and experimental pipeline. We use coding agents, including Claude Code and Codex, to inspect training logs and trajectories and propose or implement changes to the training system. LLMs also serve as policy models, user simulators, and evaluation components, as specified in Section 5. Beyond these experimental roles, generative AI tools assisted with manuscript revision, LaTeX preparation, and discussion of experimental interpretation. The authors retain responsibility for the research design, implementation, validation of reported results, and all final interpretations and claims.

REFERENCES

- Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2 -bench: Evaluating conversational agents in a dual-control environment, 2025. URL <https://arxiv.org/abs/2506.07982>.
- Guhong Chen, Yingcheng Shi, Yongbin Li, Binhua Li, Xander Xu, Hu Wei, Shiwen Ni, Min Yang, and Jieping Ye. Evotrainer: Co-evolving llm policies and training harnesses for autonomous agentic reinforcement learning. *arXiv preprint arXiv:2606.03108*, 2026a.
- Xiaoyin Chen, Canwen Xu, Yite Wang, Boyi Liu, Zhewei Yao, and Yuxiong He. Learning to self-evolve, 2026b. URL <https://arxiv.org/abs/2603.18620>.
- Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026.
- Fabio Ferreira, Lucca Wobbe, Arjun Krishnakumar, Frank Hutter, and Arber Zela. Can llms beat classical hyperparameter optimization algorithms? a study on autoresearch. *arXiv preprint arXiv:2603.24647*, 2026.
- Zhenyu Hou, Yujiang Li, Jie Tang, and Yuxiao Dong. Single-rollout asynchronous optimization for agentic reinforcement learning. *arXiv preprint arXiv:2607.07508*, 2026.
- Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=t9U3LW7JVX>.
- Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, Yifei Ming, Jiaqi Pan, Bhavana Dalvi Mishra, Jiaxin Huang, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. Envharness: Awakening static worlds for agent learning, 2026. URL <https://arxiv.org/abs/2608.19880>.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL <https://arxiv.org/abs/2310.06770>.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-rl: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.
- Minseon Kim, Zhengyan Shi, Emiliano Penaloza, Christopher Cui, Roger Creus Castanyer, Maryam Hashemzadeh, Isadora White, Jonathan Light, Jeonghye Kim, Matheus Pereira, Darya Moldavskaya, Chinmay Singh, Fabio Vera, Baolin Peng, Xingdi Yuan, Marc-Alexandre Côté, and Alessandro Sordoni. Frognano: Training a 4b coding agent via online task synthesis, 2026. URL <https://arxiv.org/abs/2609.07925>.
- Fanqi Kong, Jiayi Zhang, Mingyi Deng, Chenglin Wu, Yuyu Luo, and Bang Liu. Infopo: Information-driven policy optimization for user-centric agents, 2026. URL <https://arxiv.org/abs/2603.00656>.
- Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026. URL <https://arxiv.org/abs/2603.28052>.

- Siyi Liu, Chen Gao, and Yong Li. Large language model agent for hyper-parameter optimization. *arXiv preprint arXiv:2402.01881*, 2024.
- Zewen Liu, Zhan Shi, Yisi Sang, Bing He, Minhua Lin, Tianxin Wei, Dakuo Wang, Benoit Dumoulin, Wei Jin, and Hanqing Lu. Adaptive auto-harness: Sustained self-improvement for agentic system deployment on open-ended task streams, 2026. URL <https://arxiv.org/abs/2606.01770>.
- Alexander Panfilov, Peter Romov, Igor Shilov, Yves-Alexandre de Montjoye, Jonas Geiping, and Maksym Andriushchenko. Claudini: Autoresearch discovers state-of-the-art adversarial attack algorithms for llms. *arXiv preprint arXiv:2603.24511*, 2026.
- Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. Userbench: An interactive gym environment for user-centric agents, 2025a. URL <https://arxiv.org/abs/2507.22034>.
- Cheng Qian, Zuxin Liu, Akshara Prabhakar, Jielin Qiu, Zhiwei Liu, Haolin Chen, Shirley Kokane, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. Userrl: Training interactive user-centric agent via reinforcement learning, 2025b. URL <https://arxiv.org/abs/2509.19736>.
- Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL <https://arxiv.org/abs/2412.15115>.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL <https://arxiv.org/abs/1707.06347>.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL <https://arxiv.org/abs/2402.03300>.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.
- Anja Surina, Amin Mansouri, Lars Quaedylied, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning, 2025. URL <https://arxiv.org/abs/2504.05108>.
- Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, Qi Gu, Hui Su, and Xunliang Cai. Scaleenv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training, 2026. URL <https://arxiv.org/abs/2602.06820>.
- Guoqing Wang, Sunhao Dai, Guangze Ye, Zeyu Gan, Wei Yao, Yong Deng, Xiaofeng Wu, and Zhenzhe Ying. Information gain-based policy optimization: A simple and effective approach for multi-turn search agents, 2026. URL <https://arxiv.org/abs/2510.14967>.
- Xintao Wang, Heng Wang, Yifei Zhang, Xinfeng Yuan, Rui Xu, Jen tse Huang, Siyu Yuan, Haoran Guo, Jiangjie Chen, Shuchang Zhou, Wei Wang, and Yanghua Xiao. CoSER: Coordinating LLM-based persona simulation of established roles. In *Forty-second International Conference on Machine Learning*, 2025a. URL <https://openreview.net/forum?id=BOrR7YqKUt>.
- Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b.

- Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning, 2025. URL <https://arxiv.org/abs/2509.08755>.
- Sirui Xia, Yikai Zhang, Aili Chen, Siye Wu, Siyu Yuan, and Yanghua Xiao. From ai assistant to ai scientist: Autonomous discovery of llm-rl algorithms with llm agents. *arXiv preprint arXiv:2603.23951*, 2026.
- Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
- Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Coevolve: Training llm agents via agent-data mutual evolution, 2026. URL <https://arxiv.org/abs/2604.15840>.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL <https://arxiv.org/abs/2406.12045>.
- Simon Yu, Nicholas Tomlin, Marwa Abdulhai, Ximing Lu, Derek Chong, Abe Hou, Dilara Soyly, Sergey Levine, Christopher D. Manning, and Weiyang Shi. One frozen simulator is not enough: Simulator collapse in multi-agent rl, 2026. URL <https://arxiv.org/abs/2608.12253>.
- Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents, 2026a. URL <https://arxiv.org/abs/2505.22954>.
- Zhen Zhang, Kaiqiang Song, Xun Wang, Yebowen Hu, Weixiang Yan, Chenyang Zhao, Henry Peng Zou, Haoyun Deng, Sathish Reddy Indurthi, Shujian Liu, Simin Ma, Xiaoyang Wang, Xin Eric Wang, and Song Wang. Cm2: Reinforcement learning with checklist rewards for multi-turn and multi-step agentic tool use, 2026b. URL <https://arxiv.org/abs/2602.12268>.
- Xuhui Zhou, Weiwei Sun, Qianou Ma, Yiqing Xie, Jiarui Liu, Weihua Du, Sean Welleck, Yiming Yang, Graham Neubig, Sherry Tongshuang Wu, and Maarten Sap. Mind the sim2real gap in user simulation for agentic tasks, 2026. URL <https://arxiv.org/abs/2603.11245>.
- Yifei Zhou, Song Jiang, Yuandong Tian, Jason Weston, Sergey Levine, Sainbayar Sukhbaatar, and Xian Li. Sweet-rl: Training multi-turn llm agents on collaborative reasoning tasks. *arXiv preprint arXiv:2503.15478*, 2025.

A INTERVENTION INTERFACES AND RUN-LEVEL OBSERVATIONS

We distinguish the controls exposed to **EAU** from the outcomes of particular interventions. Table 6 documents the ColBench interfaces and their recorded use; Table 7 summarizes intervention attempts and implementation diagnostics; and Table 8 reports interface coverage on UserGym. Together, these records describe which controls were available, what changes were observed, and how the reported interventions were handled.

Write checks. For the documented knob interfaces, write checks enforce declared ranges. Difficulty-increasing updates additionally require the relevant recent training success rate to reach 0.80. In the ColBench records, four proposed updates to `max_steps` were rejected under this rule before the run qualified. These checks constrain admissible writes; they do not guarantee that an update improves performance or achieves its intended behavioral effect.

A.1 COLBENCH: INTERFACE USAGE

Table 6 covers two analyzed runs of 624 training steps each. We distinguish controls that were adjusted, controls that remained fixed or were not exercised, and interfaces with restricted implementation scope. An unused control is not a failed intervention, while an exposed control need not govern every behavior suggested by its name.

Table 6: ColBench interfaces and their use in the two analyzed runs. “Not exercised” refers to these runs only; “unavailable” denotes unsupported functionality in the documented implementation. A dash in the range column means that no range is specified or applicable.

Knob	Range	Recorded use / scope
<i>Environment</i>		
<code>max_steps</code>	3–60	Adjusted and reverted.
<code>min_questions</code>	0–3	Counter repaired; parameter adjusted.
<code>human_response_char_limit</code>	—	Not exercised.
<i>User (symbolic)</i>		
<code>facts_per_reply</code>	1–3	Adjusted and reverted.
<code>match_threshold</code>	[0, 1]	Multiple settings examined.
<code>volunteer_interval</code>	0–10	Adjusted and reverted.
<code>max_reveals</code>	0–20	Does not bound repeated disclosure of known facts.
<code>reveal_return_shape</code>	{T, F}	Filters shape, not return, facts.
<i>Reward</i>		
<code>per_gym_scale</code>	[0, 2]	Not exercised.
<code>turn_reward_clip</code>	[−1, 2]	Not exercised.
<code>predicate_weight</code>	—	Unavailable; no ColBench predicate library.
<i>User (mixture / persona)</i>		
<code>user_strategy_mix</code>	[0, 1]	Fixed at 0.5.
<code>user_persona</code>	—	Unavailable on ColBench; travel only.
<i>Agent</i>		
<code>entropy_coeff</code>	[0, 0.05]	Adjusted and reverted.

A.2 COLBENCH: INTERVENTION ATTEMPTS AND DIAGNOSTICS

The run summary records 15 knob writes, two measured harms, and three subsequently retracted attributions. It also records trajectory-based diagnoses of deadlock, a reward-hacking channel, and template collapse. Table 7 groups related observations rather than listing every write separately. It distinguishes the observed behavior from the reported response: a code repair, a rollback, an inconclusive perturbation, or a diagnostic finding without a documented follow-up repair.

Table 7: Intervention attempts and implementation diagnostics in the two ColBench runs. Reported score changes are local observations, not controlled estimates of each intervention’s contribution to final performance. A missing follow-up entry does not establish that a problem persisted throughout all subsequent runs.

Control / issue	Observed behavior	Reported response / status
<code>max_steps</code>	Reducing $10 \rightarrow 6$ lowered Pass by 0.246; reverting increased it by 0.177.	Reverted. The perturbation-and-reversal pattern concerns sensitivity to the interaction budget.
<code>min_questions</code>	The counter omitted turns that both asked and answered a question, deadlocking 41% of episodes.	Code repair reported. Subsequent $1 \rightarrow 2$ adjustment had no measured effect; $2 \rightarrow 3$ was harmful.
<code>facts_per_reply</code>	Increasing $1 \rightarrow 3$ produced reference-solution disclosure in 837-character replies.	Reverted. The initial reward attribution was not retained after accounting for the reported gap band.
<code>match_threshold</code>	The sequence 0.9/0.6/0.35/0.9 produced no change exceeding the reported noise floor of 0.228.	Inconclusive at the reported noise level.
<code>volunteer_interval</code>	Reducing $10 \rightarrow 5$ increased flow success while the question rate fell from 40% to 26%.	Reverted; the rise in flow success was interpreted as masking reduced questioning.
<code>max_reveals</code>	Re-disclosing an already revealed fact did not consume the budget, leaving repeated disclosure unrestricted by this control.	Interface limitation documented; no corresponding repair is specified in the run summary.
<code>reveal_return_shape</code>	The filter targeted <code>kind=="shape"</code> (8% of facts), whereas the observed leak involved <code>kind=="return"</code> (18%).	Type mismatch documented; no corrective edit is specified in the run summary.
<code>entropy_coeff</code>	Increasing $0 \rightarrow 0.004$ raised <code>actor/entropy</code> by 50% while behavioral diversity decreased.	Reverted. The observed effect lag exceeded the 10-step verification window.

Interpreting the intervention record. The `min_questions` example separates a code repair from later parameter search: repairing a counter does not imply that further increasing its threshold is beneficial. Similarly, the disclosure and volunteering examples show why a reward or success metric must be interpreted alongside the interaction behavior. Conversely, the `max_reveals` and `reveal_return_shape` findings identify limitations of the exposed controls, not successful adaptations. Trajectory inspection and knob adjustment thus address different parts of the adaptation process: diagnosing a problem and attempting a change.

Fixed mixture. The symbolic-LLM mixing coefficient remained at 0.5, while the two simulator arms differed by 0.29 in success rate. The blended metric therefore combines distinct interaction conditions. These runs do not provide evidence for gains from dynamically adapting the mixture, and differences between the simulator arms should not be conflated with effects of changing the mixing coefficient.

A.3 USERGYM: BENCHMARK-SPECIFIC INTERFACE COVERAGE

Table 8 distinguishes supported controls from unavailable ones in the documented UserGym implementation. A checkmark indicates interface support, not that the control was used or improved performance. Coverage is benchmark-specific, and the mixture-selector mapping imposes additional restrictions beyond the declared knob range.

Table 8: UserGym interface coverage in the documented implementation. ✓ indicates support and — indicates unavailable functionality. These entries describe implementation coverage rather than intervention outcomes.

Knob (range)	travel	tau	persuade	turtle	function
<i>Environment</i>					
<code>max_steps</code> (3–60)	✓	✓	✓	✓	✓
<code>search_failure_interval</code> (1–50)	✓	—	—	—	—
<code>elicitation_interval</code> (1–20)	✓	—	—	—	—
<code>noise_choice_number</code> (0–30)	✓	—	—	—	—
<code>wrong_choice_number</code> (0–30)	✓	—	—	—	—
<code>success_threshold</code>	—	—	—	✓	—
<code>max_repeated_probes</code>	—	—	—	—	✓
<i>Reward</i>					
<code>default_scale</code> [0, 2]	✓	✓	✓	✓	✓
<code>per_gym_scale</code> [0, 2]	✓	✓	✓	✓	✓
<code>turn_reward_clip</code> [−1, 2]	✓	✓	✓	✓	✓
<code>predicate_weight</code> [−0.5, 0.5]	—	—	—	—	✓ ^a
<i>User</i>					
<code>user_strategy_mix</code> [0, 1]	✓	— ^b	— ^b	— ^b	— ^b
<code>user_persona</code> (≤ 8)	✓	—	—	—	—
<i>Agent</i>					
<code>entropy_coeff</code> [0, 0.05]	Shared across gyms; change per write ≤ 0.004.				

^a `function` is the only listed gym with a predicate library (five `probe_*` predicates); `predicate_weight` is rejected elsewhere.

^b The mixture selector is keyed by `travel/colbench/tau2`. The other four UserGym keys resolve to `None`, so they cannot be split into symbolic and LLM arms through this interface; in particular, `tau` does not match `tau2`. This describes the recorded implementation, not a benchmark requirement.

B DETAILS ON EXPERIMENTS

Baselines. We compare EAU against both prompting-based and RL-based baselines. For prompting, we include ReAct (Yao et al., 2022), which interleaves reasoning with actions, and Reflexion (Shinn et al., 2023), which improves agent behavior through verbal feedback and self-reflection.

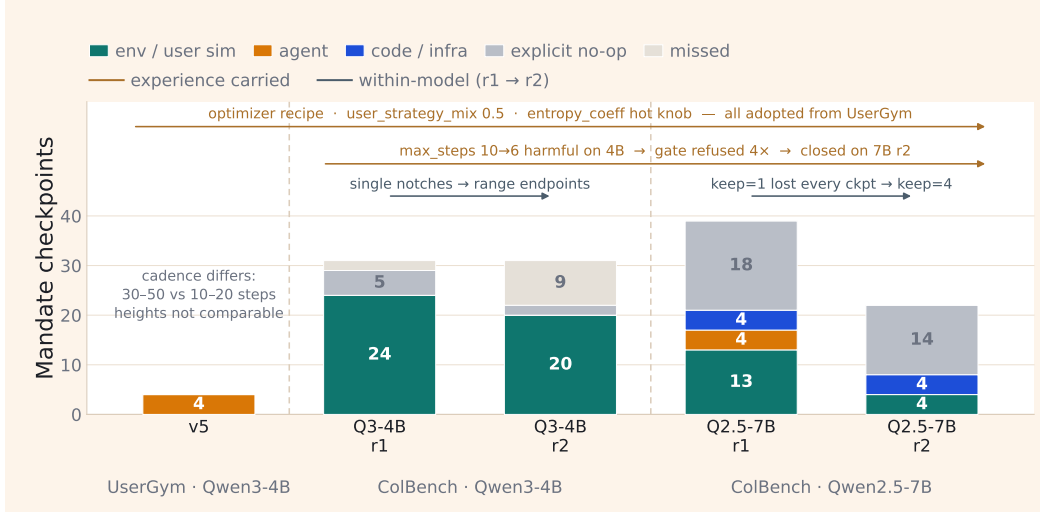


Figure 8: Experience flow from UserGym, *Qwen3-4B* to ColBench, *Qwen2.5-7B*.

For RL training, we consider RAGEN (Wang et al., 2025b), a general multi-turn agent RL framework; Search-R1 (Jin et al., 2025), which trains agents through reinforcement learning over multi-step search trajectories; UserRL (Qian et al., 2025b), which targets interactive user-centric agents; and InfoPO (Kong et al., 2026), which introduces information-driven optimization for user-agent interaction. We additionally include vanilla PPO as a standard RL baseline to isolate the gains from our co-evolution framework. For UserGym and ColBench, all baseline results except PPO are taken from Kong et al. (2026); PPO and EAU are evaluated under our implementation.

Benchmarks. We evaluate our method on three representative user-centric benchmarks, i.e., **ColBench** (Zhou et al., 2025), **UserGym** (Qian et al., 2025b), and τ^2 -**bench** (Yao et al., 2024; Barres et al., 2025). ColBench evaluates collaborative artifact creation, where the agent works with an LLM-simulated collaborator that has privileged access to a hidden reference artifact. We evaluate **EAU** on the *Backend Programming* subset, where the agent implements a Python function from an incomplete specification, and is evaluated with hidden unit tests. UserGym is a suite of eight Gymnasium environments unified under a single tool interface, spanning preference elicitation, intent clarification, entity guessing, lateral-thinking puzzles, rule induction, policy-constrained tool use, persuasion, and search. Each episode pairs a rule-based task with an LLM-simulated user who holds the ground-truth intent and reveals it only when queried. Because the initial request is systematically underspecified, the agent must repeatedly trade off asking the user against committing to an action. We report the environment reward averaged within and across gyms. Specifically, we exclude the SearchGym for evaluation due to the cost of APIs. It requires the agent to conduct online search using a charged api. Notably, this does not affect the RL training, since it’s an OOD gym solely for evaluation. τ^2 -bench casts customer service as a dual-control problem in which both the agent and the simulated user can act on a shared world state. Unlike prior benchmarks where the user is a passive information source, the user here has its own tools, so the agent must verbally guide the user through actions it cannot perform itself while complying with a domain policy. We evaluate on the retail, airline, and telecom domains, and score a task as solved only when the resulting environment state matches the annotated target and all required information has been conveyed to the user.

C ADDITIONAL IMPLEMENTATION DETAILS

We show the hyperparameters in Table 9.

D ADDITIONAL ANALYSIS

Observations from the main results. We observe that with a strong model, i.e., *Qwen3-4B*, **EAU** is able to achieve generally better performance. On benchmarks involving capabilities with

Table 9: Key hyperparameters for training on UserGym, ColBench, and τ^2 -Bench environments.

Parameter	UserGym	ColBench	τ^2 -Bench
<i>Data Configuration</i>			
Train batch size	64	64	32
Max prompt length	1152	2048	8192
Max response length	8192	8192	16384
<i>Actor Configuration</i>			
Learning rate	3×10^{-7}	3×10^{-7}	1×10^{-6}
PPO minibatch size	16	16	16
PPO microbatch size per GPU	4	4	2
KL loss coef	0.001	0.001	0.001
Entropy coefficient	0.001	0.001	0.001
<i>Rollout Configuration</i>			
Rollout engine	SGLang	SGLang	SGLang
Log prob micro batch size per GPU	4	4	2
Parallel rollouts (n)	5	5	5
Max turns	16	10	50
Response length (per turn)	1024	1024	1024
GPU memory utilization	0.5	0.5	0.5
<i>Training Configuration</i>			
GPUs per node	4	4	4
Total epochs	2	2	2

which the model is more familiar—namely coding (ColBench) rather than tool use (UserGym)—**EAU** can achieve relatively greater performance gains. This may suggest that **EAU** is more adept at eliciting better interaction behavior from the model than at overcoming its intrinsic capability limitations.