AutoForge: Automated Environment Synthesis for Agentic Reinforcement Learning

Anonymous ACL submission

Abstract

Conducting reinforcement learning (RL) in simulated environments offers a cost-effective and highly scalable way to enhance language-based agents. However, previous work has been limited to semi-automated environment synthesis or tasks lacking sufficient difficulty, offering little breadth or depth. In addition, the instability of simulated users integrated into these environments, along with the heterogeneity across simulated environments, poses further challenges for agentic RL. In this work, we propose: (1) a unified pipeline for automated and scalable synthesis of simulated environments associated with high-difficulty but easily verifiable tasks; and (2) a multi-environment user-centric RL algorithm that not only effectively mitigates user instability but also performs advantage estimation at the environment level, thereby improving training efficiency and stability. Comprehensive evaluations on agentic benchmarks, including τ-bench, τ^2-Bench, and VitaBench, validate the effectiveness of our proposed method. Further in-depth analyses underscore its out-of-domain generalization.

1 Introduction

As large language models (LLMs) have demonstrated powerful tool-use capabilities (Shen, 2024; Team et al., 2025b; Wang et al., 2025) and robust multi-turn conversational skills (Yi et al., 2024), the research community has increasingly been building language-based agents to automate task solving (Wang et al., 2024; Su et al., 2025). However, current LLM-based agents still struggle with complex real-world tasks that require extensive interaction with environments and users (He et al., 2025). Reinforcement learning (RL) in real-world environments appears to be an effective way to enhance agent capabilities, but it suffers from high costs and poor scalability (Zhao et al., 2025). Therefore, synthesizing simulated environments for agentic RL has emerged as a popular approach (Qian et al., 2025b).

While some recent efforts have explored training agents with mock environments and simulated users (Xi et al., 2025; Ye et al., 2025; Du et al., 2025), they fall short in three key aspects: (1) Prior work (Fang et al., 2025; Qian et al., 2025b) has been limited to semi-automated environment synthesis or tasks that are not sufficiently challenging, falling short in both the breadth and depth required for effective agent training; (2) Some studies (Zhao et al., 2025) introduce simulated LLM-based users to interact with agents, yet neglect the instability of such users; (3) Existing approaches (Qian et al., 2025b) view multi-environment RL training from a single-environment perspective, leading to suboptimal efficiency and stability.

We thus argue that a unified pipeline capable of automatically generating mock environments and complex tasks is better suited to simultaneously broadening the scope of agent training and deepening its complexity. In addition, during the interaction between the agent and the simulated user, we emphasize the importance of real-time monitoring of the simulated user's accuracy, so as to ensure that the feedback received by the agent is reasonable. Furthermore, we highlight treating agentic RL at the multi-environment level to improve training efficiency and stability.

To this end, we introduce **AutoForge**, a novel framework that integrates an automated pipeline for synthesizing environments and challenging tasks, together with a multi-Environment User-centric relative Policy Optimization (EUPO) algorithm. Specifically, our synthesis pipeline starts from tool description documentation, enabling the automated construction of a database to store environment states and the generation of tool implementations in Python. A dependency graph of the tools is then constructed, upon which random walks yield diverse tool sequences. These sequences are merged

and augmented with reasoning nodes and edges to form a complex directed acyclic graph (DAG), which in turn serves as the blueprint for producing tasks. To address instability in the simulated user, we employ an LLM-as-judge mechanism during the RL rollout phase to identify and mask trajectories where task failures are due to simulated user errors, thereby promoting fairness and reducing bias in advantage estimation. Furthermore, we extend the native GRPO's group-level advantage estimation (Shao et al., 2024) to an ***environment-level*** formulation, which effectively mitigates the impact of outlier samples on the standard deviation. This enhancement improves the accuracy of advantage estimation and contributes to greater training stability.

To validate the effectiveness of the proposed framework, we conduct comprehensive experiments on three popular agentic benchmarks: τ-bench, τ^2-Bench and VitaBench. Experimental findings indicate that AutoForge markedly enhances the agentic capacity to solve tasks after engaging in extended multi-turn interactions with the environment and the user. Further in-depth analysis reveals that AutoForge exhibits strong out-of-domain generalization capability as well as robust training stability. In conclusion, our main contributions are summarized as follows:

- We present a unified pipeline capable of automatically building mock environments and producing high-difficulty tasks, requiring only a textual description of the tool.
- We introduce EUPO, an agentic RL algorithm that effectively mitigates the instability caused by simulated users while enabling more robust environment-level advantage estimation.
- Comprehensive experimental results confirm both the effectiveness and strong generalization of AutoForge, underscoring the promise and importance of leveraging mock environments in agentic reinforcement learning.

2 Related Work

Agentic Reinforcement Learning. For an agent learning within a given environment, the optimization objective is to maximize cumulative reward through a sequence of interactions with the environment. A growing body of recent work studies agents that extend LLMs with external tools. Search agents (Li et al., 2025b; Wu et al., 2025; Li et al., 2025a; Tao et al., 2025) interleave reasoning with real-time evidence retrieval to solve challenging open-domain questions. In the code-generation arena, CodeRL (Le et al., 2022) and DataMind (Qiao et al., 2025) treat the interpreter as an executable action space and optimize code-writing policies via reward signals received from execution feedback. For general tool manipulation, ToolRL (Qian et al., 2025a) and ToolN1 (Chen et al., 2025b) dispense with dense instruction tuning and instead train LLMs to select and invoke APIs by directly maximizing task success. Empirical results show that environment-grounded optimization extends the reasoning horizon of language models but raises challenges in sample efficiency, credit assignment, and training stability. In our paper, we accelerate RL training by synthesizing a sandbox environment to get efficient feedback. Furthermore, we improve credit assignment and training stability by masking user-induced hallucination errors and estimating advantages at the environment level.

Environment Emulation. In contrast to traditional reinforcement learning, agentic reinforcement learning requires the agent to interact with the environment. When agents interact with search engines (Wu et al., 2025; Li et al., 2025a), MCP services (Xu et al., 2025; Ren et al., 2025), or other real-world tools, they spend most of their wall-clock time waiting. Idle *bubble* minutes account for most of the experimental time. To address this latency issue, numerous works have explored environment simulation to get efficient feedback. On one hand, approaches such as Toolbench(Qin et al., 2023), ToolSandbox(Guo et al., 2024), ZeroSearch(Sun et al., 2025), train a model-based environment simulator. However, due to inherent hallucinations in LLMs, the same agent action may yield inconsistent feedback, and the diversity of the trained model's outputs introduces instability, limiting the reliability of such model-based simulators. On the other hand, works like τ-bench(Yao et al., 2024; Barres et al., 2025), construct local, executable environments, e.g., by converting tasks into locally runnable code or databases. Yet these approaches heavily rely on manual annotation, and automating the construction(Shi et al., 2025) of complex, diverse tasks remains highly challenging. In this paper, we propose a method for automatically constructing RL training environments and tasks for agents.

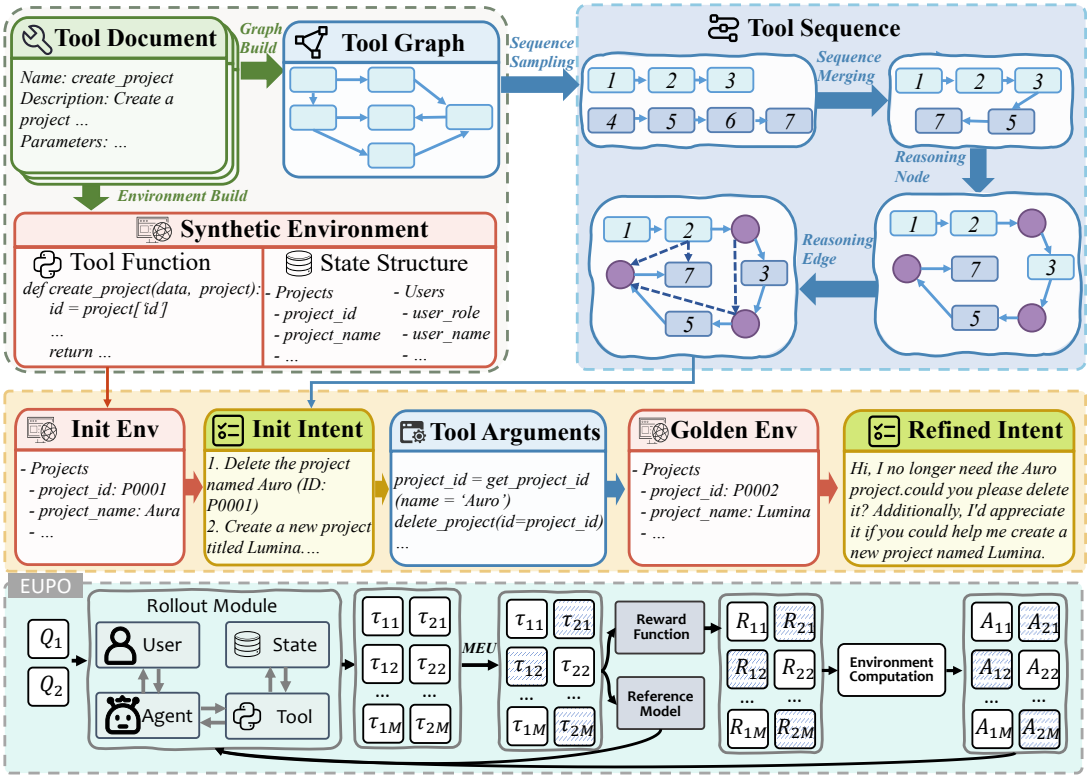


Figure 1: The AutoForge framework consists of a unified pipeline for scalable synthesis of simulated environments and high-difficulty, easily verifiable tasks, and EUPO algorithm for multi-environment agentic RL.

3 Method

In this section, we first introduce the verifiable interactive-environment synthesis pipeline, which consists of three key steps: (1) environment synthesis (Section §3.1), (2) tool-sequence generation (Section §3.2), and (3) task generation (Section §3.3). We then present our environment-level agentic RL algorithm (Section §3.4).

3.1 Environment Synthesis

Formally, we define a synthetic environment $\mathcal{E}$ as a tuple comprising the state S and an operation function set $\mathcal{F}$, such that $\mathcal{E} = (S, \mathcal{F})$. For scalable and automated synthesis of environments, we leverage the gathered tool-description documentation to automatically construct the state S and the corresponding function set $\mathcal{F}$ applicable to the state.

State Structure Generation. To ensure both the determinism and the flexible extensibility of the state, we record it in a database, which can be formally represented as a list of key–value pairs: $S = [(K_1, V_1), (K_2, V_2), \ldots, (K_n, V_n)]$, where K_i denotes the i-th attribute name (e.g., `"project_id"`), and V_i denotes its corresponding value (e.g., `"P0001"`). In this stage, we begin by prompting a LLM to produce the state structure, namely all attribute names K_i, while leaving the generation of specific attribute values V_i to Section §3.3 in order to better ensure the consistency between tasks and environments.

Function Set Generation. It is important to note that a function executable on a given state requires the state structure rather than the specific state values. Therefore, once the state structure is obtained, we leverage the tool-description documents together with the state structure to prompt an LLM for generating the corresponding Python code. The Python code not only conforms to the state structure but also offers extremely low execution cost, high concurrency, and strong stability, which fully satisfy the requirements of RL training.

3.2 Tool-Sequence Generation

After building the synthetic simulation environment, we further construct challenging tasks that align with the environment to facilitate RL training for the agent. We posit that the difficulty of a task is determined by the specific sequence of tools needed to accomplish it. Therefore, our first step is to design tool sequences of high complexity.

Sequence Sampling. Motivated by

AgentScaler (Fang et al., 2025), we begin by representing all available tools as nodes in a directed graph, derived from their description documents. A directed edge between two tools is established when an LLM judges that the output of one tool is likely to be a valid input for another. Subsequently, we perform random walks on the graph to obtain thousands of tool sequences. However, tool sequences obtained through simple sampling may not reflect complex tasks, such as those involving multiple requirements or intricate reasoning. We therefore enhance their complexity to enable the generation of more sophisticated tasks in later stages.

Tool Sequence Merging. Formally, the i-th tool sequence can be denoted as $\mathcal{T}_i = [t_{i_1}, t_{i_2}, \ldots, t_{i_{|\mathcal{T}_i|}}]$. To obtain a tool sequence that encompasses multiple requirements, we can merge two existing sequences $\mathcal{T}_i$ and $\mathcal{T}_j$ to produce a new sequence $\mathcal{T}_k'$. However, directly concatenating $\mathcal{T}_i$ and $\mathcal{T}_j$ may introduce redundant tools. For instance, $\mathcal{T}_i$ may contain the tool `find_id_by_name`, while $\mathcal{T}_j$ includes `find_id_by_email`, resulting in a merged tool sequence that repeatedly retrieves the same `id` information. To address this, we prompt an LLM to remove redundant tools, yielding a more coherent merged sequence.

Reasoning Node Integration. Subsequently, to ensure that a tool sequence captures complex reasoning, we employ an LLM to insert reasoning nodes into the sequence. A reasoning node is defined as a node that performs inference on the outputs of preceding nodes to derive higher-level information. For example, if a preceding node outputs the prices of all items, a reasoning node may compute the total price of these items. After this step, we obtain tool sequences enriched with higher-order reasoning, formally denoted as $\mathcal{T}_k'' = [t_{k_1}, \ldots, r_{k_1}, \ldots, t_{k_i}, r_{k_j}, \ldots, t_{k_{|\mathcal{T}_k''|}}]$, where r_{k_j} denotes the j-th reasoning node in $\mathcal{T}_k''$.

Reasoning Edge Integration. Finally, to incorporate explicit dependency information among tools, we utilize an LLM to augment the tool sequence $\mathcal{T}_k''$ with a set of directed edges, referred to as reasoning edges E_k. This results in a directed acyclic graph (DAG) formally represented as $G_k = (\mathcal{T}_k'', E_k)$. A reasoning edge is defined as a directed connection where the output of the parent node is used, through a reasoning process, to generate the input parameters for the child node.

3.3 Task Generation

Using G_k as a blueprint, we generate tasks that guide the agent's interaction with the synthetic environment.

Environment Initialization. As discussed in Section §3.1, we have already synthesized the environment structure. Therefore, we first instantiate the environment structure, which means generating a corresponding value V_i for each attribute name K_i. Similar to previous work (Barres et al., 2025), we simulate users in the environment and use the user intent to represent the task. Given the instantiated environment S_k and the graph G_k, we prompt an LLM to generate the corresponding initial task question (intent) $\tilde{Q}_k$.

Tool Sequence Execution. A key question is how to verify whether the agent has successfully completed the task. Although we have the gold tool sequence $\mathcal{T}_k''$, we choose to evaluate task completion based on the final environment state rather than the tool sequence. This is because a single task may be solvable through multiple valid sequences of tool calls. To obtain the final environment state, we use the initial task $\tilde{Q}_k$ to fill in the tool arguments for each tool name in G_k following its topological order, and then execute these tools on the initial environment state S_k, resulting in the final state S_k^*.

Task Refinement. Finally, we use the initial and final environment states, S_k and S_k^*, to refine the task $\tilde{Q}_k$. A well-crafted task description should contain only the minimal information necessary to solve the task, while remaining linguistically natural. This process yields an RL training sample $D_k = (Q_k, S_k, S_k^*, \mathcal{F})$, where Q_k is the **refined** version of the task.

3.4 EUPO

In this section, we leverage the constructed dataset $\mathcal{D}$ to train the agent to continuously interact with the environment and ultimately solve the task. To better simulate real-world scenarios where the user continuously issues requests and the agent interacts with both the user and the environment, we introduce a simulated *user agent* into the synthetic environment during the RL stage. The user agent, given the task Q, continuously provides relevant information. Formally, at step t, the agent's action a_t can be one of two options: (1) call a tool, or (2) request information from the user agent. The corresponding observation o_t is: (1) the result returned after tool execution, or (2) a textual re-

ply from the user agent. We extend the vanilla GRPO algorithm (Shao et al., 2024) in the following four aspects towards environment-level agentic reinforcement learning.

User-Centered Rollout. The rollout phase begins with the user agent generating the first request text o_0 based on Q. The agent then autonomously decides whether to call a tool or request information from the user agent. The rollout terminates when the user agent determines that all requirements in Q have been satisfied. Formally, a rollout trajectory can be represented as: $\tau = (o_0, a_1, o_1, a_2, o_2, \ldots, a_n, o_n)$, where a_t denotes the action at step t and o_t the corresponding response. After the rollout, we obtain an environment state $\hat{S}$, which is compared with the golden environment state S^* to compute the reward R. Therefore, the function for R is as follows:

$$R = \begin{cases} 1, & \text{if } S^* == \hat{S} \\ 0, & \text{if } S^* != \hat{S} \end{cases} \tag{1}$$

Interleaved Thinking. We use Qwen3 (Team, 2025c) as the training backbone, which discards previous reasoning content when a new user query is received. However, we emphasize that for agent tasks requiring multiple rounds of interaction with the user, it is important to retain previous reasoning content, which often includes task analysis and planning. To this end, we modify both the training and inference procedures to preserve all thinking content, thereby enabling *Interleaved Thinking*, as similarly explored in Tongyi DeepResearch (Team et al., 2025b), GPT-OSS (Agarwal et al., 2025) and Minimax-M2 (Team, 2025b).

Masking Out Erroneous User Behaviors. It is worth noting if during the rollout, the user agent make mistakes, such as returning hallucinated information or omitting information contained in Q, which can prevent the agent from completing the task. To address this, we employ an LLM to identify trajectories in which the user agent has erred and mask them out so that they **do not** contribute to advantage or loss computation, thereby ensuring the accuracy of advantage estimation. The detailed prompt is provided in Appendix B.1.

For each input D, GRPO (Shao et al., 2024) samples a group of trajectories $\{\tau_1, \tau_2, \ldots, \tau_M\}$ from the old policy model π_{old}. After applying masking of erroneous user behaviors (MEU), the RL objective can be expressed as:

$$\begin{aligned} J_{\text{EUPO}}(\theta) = \mathbb{E}_{D\sim\mathcal{D},\{\tau_i\}_{i=1}^{M}\sim\pi_{\pi_{\text{old}}}(\cdot|D)} \Bigg[\frac{1}{\sum_{i=1}^{M} \mathbf{1}_{\text{MEU}}(\tau_i)} \\ \sum_{i=1}^{M} \mathbf{1}_{\text{MEU}}(\tau_i) \times \min\Big(\frac{\pi_\theta(\tau_i|D)}{\pi_{\text{old}}(\tau_i|D)} A_i, \text{clip}(\frac{\pi_\theta(\tau_i|D)}{\pi_{\text{old}}(\tau_i|D)}, \\ 1-\epsilon, 1+\epsilon) A_i \Big) - \beta D_{\text{KL}}(\pi_\theta \parallel \pi_{\text{old}}) \Bigg], \end{aligned} \tag{2}$$

where ϵ and β are hyperparameters, A_i is the advantage function, and $\mathbf{1}_{\text{MEU}}(\tau_i)$ is an indicator that equals 0 if the user agent makes an error in trajectory τ_i, and 1 otherwise.

Environment-Level Advantage Estimation. Formally, a training batch contains multiple environments. Suppose each environment includes P questions $\{Q_i\}_{i=1}^{P}$. For each question Q_i, we sample M trajectories, and after MEU filtering, obtain M_i valid trajectories $\{\tau_{ij}\}_{j=1}^{M_i}$ with the corresponding rewards $\{R_{ij}\}_{j=1}^{M_i}$. The original GRPO computes the group-level advantage using all trajectories from the same question, as follows:

$$A_{ij}^{\text{group}} = \frac{R_{ij} - mean(\{R_{ij}\}_{j=1}^{M_i})}{std(\{R_{ij}\}_{j=1}^{M_i})}. \tag{3}$$

We define an environment-level advantage (EAE) by replacing the group-level normalization with performing normalization within each environment. Concretely, the advantage is computed as:

$$A_{ij}^{\text{env}} = \frac{R_{ij} - mean(\{R_{ij}\}_{j=1}^{M_i})}{std(\{R_{ij}\}_{j=1,\ldots,M_i}^{i=1,\ldots,P})}. \tag{4}$$

This formulation replaces the group-level normalization with an environment-level one, ensuring that advantages are computed separately for each environment. The motivation behind this design is that group-level standard deviation estimates can be unstable and overly sensitive to outliers, particularly when the group size is small or the return distribution is heavy-tailed. By normalizing returns within each environment, the resulting advantage estimates are more stable and better reflect relative performance among trajectories collected under the same environment dynamics.

4 Experiments

In this section, we aim to answer the following research questions (RQ):

- **RQ1**: How well does AutoForge perform on **in-domain** agent tasks?

Model	τ-bench		τ^2-Bench			VitaBench			
	Retail	Airline	Retail	Airline	Telecom	Delivery	In-store	OTA	Cross Domain
Closed-Source Large Language Models									
Gemini-2.5-pro	68.7	44.0	67.5	56.0	27.2	49.0	43.8	26.5	23.5
Claude-Sonnet-4	73.9	40.0	67.5	54.0	47.4	46.0	51.5	29.0	23.0
GPT-o3	70.4	52.0	80.2	64.8	58.2	53.5	53.5	37.8	30.0
GPT-o4-mini	70.4	46.0	70.2	56.0	46.5	44.5	46.5	23.5	19.5
GPT-5-thinking	78.3	44.0	81.1	62.6	96.7	54.0	52.5	37.5	22.8
Open-Source Large Language Models									
GPT-OSS-120B-A5B	67.8	49.2	57.0	38.0	45.6	37.0	42.0	12.0	15.0
Deepseek-V3.1-671B-A37B	66.1	40.0	64.9	46.0	38.5	34.0	42.5	18.3	16.3
Kimi-K2-1T-A32B	73.9	51.2	70.6	56.5	65.8	35.3	42.5	22.0	15.5
Qwen3-Thinking-235B-A22B	67.8	46.0	71.9	58.0	45.6	44.0	46.0	17.5	18.8
xLAM-2-32B-fc-r	64.3	45.0	55.3	52.0	16.7	26.0	17.0	10.0	4.0
xLAM-2-70B-fc-r	67.1	45.2	61.4	56.0	14.0	14.0	11.0	2.0	2.0
Seed-OSS-36B	70.4	46.0	68.4	52.0	41.2	26.0	39.0	7.0	6.1
Qwen3-Thinking-30B-A3B	67.8	48.0	58.8	58.0	26.3	35.0	40.0	20.5	16.0
Qwen-Coder-30B-A3B	68.7	48.0	60.5	42.0	35.3	23.0	21.0	12.0	4.0
AgentScaler-30B-A3B	70.4	54.0	70.2	60.0	55.3	25.0	33.0	16.0	8.0
MUA-RL-32B	72.6	46.5	67.3	45.4	28.3	30.0	32.5	11.5	10.5
AutoForge-30B-A3B	**73.1**	**56.5**	**74.8**	**62.0**	**76.3**	**46.0**	**54.5**	**24.0**	**17.5**

Table 1: **Main results** on τ-bench, τ^2-Bench, and VitaBench. **Bold** indicates the best among open-source models.

- **RQ2**: How well does AutoForge generalize to **out-of-domain** agent tasks?
- **RQ3**: Do the simulated users in the benchmark affect the agent's performance evaluation?
- **RQ4**: What are the effects of the key components of AutoForge?

4.1 Experimental Setup

Benchmarks. We evaluate the in-domain agent performance of our model using three widely adopted benchmarks, τ-bench (Yao et al., 2024), τ^2-Bench (Barres et al., 2025), and VitaBench (He et al., 2025), all of which feature standardized multi-turn conversations and standardized tool invocation formats. To further assess the generalization capability of AutoForge, we employ ACEBench (Chen et al., 2025a), which records multi-turn conversations via prompts and adopts a customized tool invocation format. The detailed information of these benchmarks is as follows:

- **τ-bench** introduces simulated users to evaluate how agents and users interact over multiple rounds to complete tasks in retail and airline scenarios. Notably, τ-bench is limited to sequential tool invocation.
- **τ^2-Bench** expands the evaluation by adding a telecom scenario, enabling tool usage by users. Moreover, it supports parallel tool invocation by agents.
- **VitaBench** focuses on scenarios such as food delivery and hospitality, further increasing task complexity, which requires agents to perform multiple rounds of reasoning and interact repeatedly with simulated users.
- **ACEBench** covers a broader range of domains and adopts a customized tool invocation format, making it well-suited for evaluating an agent's generalization capability. We chose ACEBench-zh for evaluation to further increase the degree of out-of-domain testing.

Baselines. We compare AutoForge against the following types: *closed-sourced LLMs*, including Gemini-2.5-pro (Comanici et al., 2025), Claude-Sonnet-4 (Anthropic, 2025), GPT-o3, GPT-o4-mini (OpenAI, 2025b), and GPT-5 (with thinking) (OpenAI, 2025a); *open-sourced LLMs*: GPT-OSS-120B-A5B (Agarwal et al., 2025), Deepseek-V3.1-671B-A37B (DeepSeek-AI, 2024), Kimi-K2-1T-A32B (Team et al., 2025a), Qwen3-Thinking-235B-A22B (Team, 2025c), Seed-OSS-36B (Team, 2025a), Qwen-Coder-30B-A3B (Hui et al., 2024), AgentScaler-30B-A3B (Fang et al., 2025), MUA-RL-32B (Zhao et al., 2025) and xLAM-2 model series (Prabhakar et al., 2025).

Implementation Details. Our experiments are conducted on 64 GPUs. To ensure the quality of the synthesized environments, we employ Qwen3-Thinking-235B-A22B (Team, 2025c) for environment synthesis. Specifically, we synthesize 10 virtual environments containing a total of 1078 high-difficulty tasks. We use Qwen3-Thinking-30B-A3B as the backbone model. Due to space constraints, more Implementation Details are presented in Appendix A.

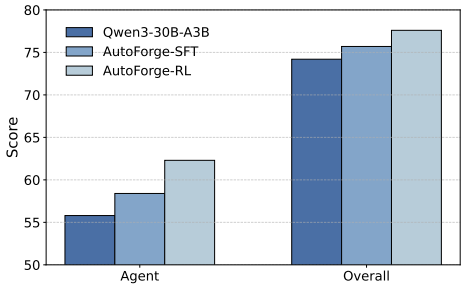


Figure 2: **Out-of-Domain performance** on the ACEBench-zh. We reported both the `Agent` and `Overall` subset scores.

4.2 Performance Comparison (RQ1 & RQ2)

In-domain Performance. In this section, to demonstrate the effectiveness of AutoForge, we compare it with state-of-the-art closed-source and open-source models. Based on the experimental results in Table 1, we can observe that: (1) AutoForge markedly outperforms its backbone, Qwen3-Thinking-30B-A3B, achieving the best results among open-source models below 200B parameters, all while operating with merely 30B total parameters and just 3B active parameters. (2) AutoForge achieves performance comparable to that of advanced closed-source LLMs, significantly narrowing the gap between open-source and closed-source systems. This further validates the effectiveness of conducting agentic reinforcement learning across multiple synthesized environments.

Out-of-Domain Performance. As emphasized in Section §4.1, AutoForge is trained using the standard multi-turn format and the official Hermes tool-invocation format defined by Qwen (Team, 2025c). We select ACEBench-zh as the out-of-domain benchmark because it exhibits four key characteristics:

(1) **Customized multi-turn format**: ACEBench-zh provides historical messages as raw prompt strings, which differs from the configuration adopted by conventional models. (2) **Customized tool-invocation format**: ACEBench-zh defines a tool-invocation format that does not follow the Hermes format. (3) **Unseen tools**: we have verified that none of the tools appearing in ACEBench-zh are present in our training data. (4) **Different language**: all training data is in English, whereas ACEBench-zh is entirely in Chinese. To further examine the generalization capability acquired through training in synthesized environments, we evaluated the performance of both the SFT and RL versions of AutoForge and compared them with the backbone model, Qwen3-Thinking-30B-A3B. As shown in Figure 2, we observe that both the SFT and RL variants of AutoForge improve the model's performance. The RL version achieves larger gains, demonstrating AutoForge's strong generalization capability and underscoring the potential of scaling synthesized environments to enhance a model's general agent abilities.

User	Retail	Airline	Telecom
Base	74.8	62.0	76.3
Optimized Prompt (OP)	75.4	62.5	76.3
Optimized Model (OM)	**76.3**	**63.5**	**90.4**

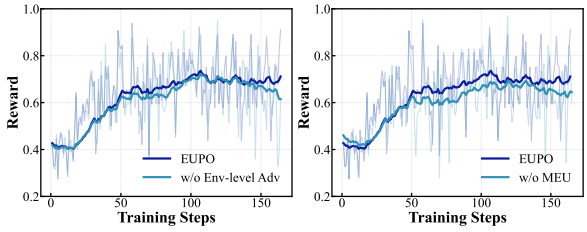
Table 2: **Comparison of different simulated users.**

4.3 In-depth Analysis (RQ3 & RQ4)

In this section, we conduct an in-depth analysis of the agent's task-solving capabilities. We first investigate the impact of simulated users in the benchmark on evaluation. Subsequently, we conduct ablation studies, including environment-level advantage estimation, masking out erroneous user behaviors, and interleaved thinking.

Impact of the User Agent. Notably, across all experiments, GPT-4.1 was used as the simulated user in benchmark evaluations. This section explores whether a more capable simulated user could enable the agent to perform tasks more successfully — in other words, whether AutoForge's true performance might have been underestimated because of mistakes made by the simulated user. Specifically, we configure three types of simulated users: (1) Base: directly using GPT-4.1 with the official prompt provided by τ^2-Bench; (2) Optimized Prompt (OP): using GPT-4.1 with a manually optimized prompt, the full details of which are presented in Appendix B.2; (3) Optimized Model (OM): employing a more capable model, GPT-5-thinking (OpenAI, 2025a), to simulate the user following the official prompt.

Based on the results in Table 2, we can observe that: (1) Both OP and OM improve the evaluation scores on τ^2-Bench, with OM achieving a greater gain, indicating that a stronger simulated user can unlock the full potential of AutoForge. This is because a weaker simulated user may supply incorrect information or perform erroneous actions, which can render tasks impossible to complete. (2) The gains from a more capable user in the telecom domain are notably larger than those in the retail



(a) (b)

Figure 3: (a) **Reward curve comparison with and without environment-level advantage estimation.** (b) **Reward curve comparison with and without MEU.**

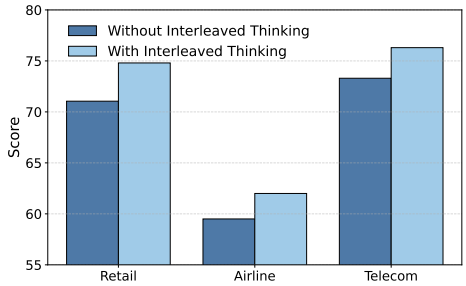


Figure 4: **Effectiveness of interleaved thinking.**

and airline domains. In telecom tasks, the user can invoke tools, placing greater demands on user competence. This further highlights that the reliability of the simulated user is critical to the fair and accurate assessment of an agent's performance.

Method	Retail	Airline	Telecom
SFT	71.3	59.5	71.9
GRPO	72.2	60.5	74.6
EUPO	**74.8**	**62.0**	**76.3**
EUPO without MEU	73.1	61.0	75.0
EUPO without EAE	73.9	61.0	75.7

Table 3: **Comparison of different training methods.**

Environment-Level Advantage Estimation (EAE). To validate the effectiveness of EAE, we compare its training reward curve with that of the standard group-level advantage estimation. As shown in Figure 3a, the reward curve obtained using EAE is more stable and achieves higher reward values. This demonstrates that EAE provides more accurate standard deviation estimates than group-level estimation, enabling better advantage comparisons among different trajectories within the same environment.

Masking Out Erroneous User Behaviors. Figure 3b also illustrates the training curves with and without masking out trajectories from erroneous simulated users. We clearly observe that masking out trajectories from erroneous simulated users results in a more stable training curve. In addition, we present the final performance comparison of different methods in Table 3, which further proves the effectiveness of MEU.

Effectiveness of Interleaved Thinking. To evaluate the effectiveness of interleaved thinking, we present a comparison with and without this mechanism during training and inference in Figure 4. The results demonstrate that incorporating interleaved thinking significantly enhances the model's agent capabilities. Specifically, by allowing the model to access and integrate its prior-turn thinking content, it can revisit and reflect on previous analyses and plans regarding the task, thereby improving the accuracy of subsequent decision-making.

Method	AVG@8
Base	60.4
+ Task refinement	41.7
+ Complex tool sequences	35.6
AutoForge	26.7

Table 4: **Comparison of task generation methods.**

Analysis of Data Difficulty. To validate the effectiveness of task refinement and complex tool sequences in Section §3, we evaluate their impact on task difficulty, which is measured by Qwen3-30B-A3B's avg@8. We use AgentScaler's generation method as the "Base" setting. As shown in Table 4, both task refinement and complex tool sequences increase task difficulty, which helps agents learn longer-horizon and more complex tasks.

5 Conclusion

In this study, we present a unified pipeline for automatically generating simulated environments and high-difficulty, verifiable tasks, enabling the expansion of agentic RL along both breadth and depth dimensions. In addition, we introduce MEU, which effectively mitigates the unfairness in advantage estimation caused by simulated user instability. Furthermore, we treat advantage estimation at the environment level, which effectively improves training stability. Extensive experiments demonstrate not only AgentScaler's strong agent capabilities but also its out-of-domain generalization, highlighting the great potential of simulated environments for agentic reinforcement learning.

Limitations

In this work, our synthetic pipeline relies on tool description documents to generate mock environments, and still places certain constraints on the input. As future work, we aim to relax these input requirements, enabling the automatic construction of high-quality environments from task topics or general text. Besides, we currently train agentic RL using only a few environments. Investigating how scaling up the number of synthetic environments affects RL training and out-of-domain generalization would be beneficial. Moreover, our EUPO is limited to outcome-based reward supervision. We will further explore turn-level value supervision to improve the agent's step-by-step decision-making.

References

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. System card: Claude opus 4 & claude sonnet 4.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, and 1 others. 2025a. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*.

Fu Chen, Peng Wang, Xiyin Li, Wen Li, Shichi Lei, and Dongdong Xiang. 2025b. Toolexpander: Extending the frontiers of tool-using reinforcement learning to weak llms. *arXiv preprint arXiv:2510.07737*.

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*.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Weihua Du, Hailei Gong, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. 2025. Generalizable end-to-end tool-use rl with synthetic codegym. *arXiv preprint arXiv:2509.17325*.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, and 1 others. 2025. Towards general agentic intelligence via environment scaling. *arXiv preprint arXiv:2509.13311*.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. 2025. Vitabench: Benchmarking llm agents with versatile interactive tasks in real-world applications. *arXiv preprint arXiv:2509.26490*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.

Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, and 1 others. 2025a. Websailor: Navigating super-human reasoning for web agent. *arXiv preprint arXiv:2507.02592*.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. 2025b. Webthinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*.

OpenAI. 2025a. Introducing gpt-5.

OpenAI. 2025b. Introducing openai o3 and o4-mini.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, and 1 others. 2025. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025a. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Jielin Qiu, Zhiwei Liu, Haolin Chen, Shirley Kokane, Heng Ji, Weiran Yao, Shelby Heinecke, and 1 others. 2025b. Userrl: Training interactive user-centric agent via reinforcement learning. *arXiv preprint arXiv:2509.19736*.

Shuofei Qiao, Yanqiu Zhao, Zhisong Qiu, Xiaobin Wang, Jintian Zhang, Zhao Bin, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and 1 others. 2025. Scaling generalist data-analytic agents. *arXiv preprint arXiv:2509.25084*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Zhenzhen Ren, Xinpeng Zhang, Zhenxing Qian, Yan Gao, Yu Shi, Shuxin Zheng, and Jiyan He. 2025. Gtm: Simulating the world of tools for ai agents. *arXiv preprint arXiv:2512.04535*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Zhuocheng Shen. 2024. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, and 1 others. 2025. Taskcraft: Automated generation of agentic tasks. *arXiv preprint arXiv:2506.10055*.

Liangcai Su, Zhen Zhang, Guangyu Li, Zhuo Chen, Chenxi Wang, Maojia Song, Xinyu Wang, Kuan Li, Jialong Wu, Xuanzhong Chen, and 1 others. 2025. Scaling agents via continual pre-training. *arXiv preprint arXiv:2509.13310*.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. 2025. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*.

Zhengwei Tao, Jialong Wu, Wenbiao Yin, Junkai Zhang, Baixuan Li, Haiyang Shen, Kuan Li, Liwen Zhang, Xinyu Wang, Yong Jiang, and 1 others. 2025. Webshaper: Agentically data synthesizing via information-seeking formalization. *arXiv preprint arXiv:2507.15061*.

ByteDance Seed Team. 2025a. Seed-oss open-source models. https://github.com/ByteDance-Seed/seed-oss.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025a. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Minimax Team. 2025b. Minimax m2. https://www.minimaxi.com/news/minimax-m2.

Qwen Team. 2025c. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, and 1 others. 2025b. Tongyi deepresearch technical report. *arXiv preprint arXiv:2510.24701*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Qi Wang, Hongzhi Zhang, Jia Fu, Kai Fu, Yahui Liu, Tinghai Zhang, Chenxi Sun, Gangwei Jiang, Jingyi Tang, Xingguang Ji, and 1 others. 2025. Klear-agentforge: Forging agentic intelligence through post-training scaling. *arXiv preprint arXiv:2511.05951*.

Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, and 1 others. 2025. Webdancer: Towards autonomous information seeking agency. *arXiv preprint arXiv:2505.22648*.

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

Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Poovendran, and Rameswar Panda. 2025. Toucan: Synthesizing 1.5 m tool-agentic data from real-world mcp environments. *arXiv preprint arXiv:2510.01179*.

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

Junjie Ye, Changhao Jiang, Zhengyin Du, Yufei Xu, Xuesong Yao, Zhiheng Xi, Xiaoran Fan, Qi Zhang, Tao Gui, Xuanjing Huang, and 1 others. 2025. Feedback-driven tool-use improvements in large language models via automated build environments. *arXiv preprint arXiv:2508.08791*.

Zihao Yi, Jiarui Ouyang, Zhe Xu, Yuwen Liu, Tianhao Liao, Haohao Luo, and Ying Shen. 2024. A survey on recent advances in llm-based multi-turn dialogue systems. *ACM Computing Surveys*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao Zhai, and Xunliang Cai. 2025. Mua-rl: Multi-turn user-interacting agent reinforcement learning for agentic tool use. *arXiv preprint arXiv:2508.18669*.

A Implementation Details

Our experiments are conducted on 64 GPUs. To ensure the quality of the synthesized environments, we employ Qwen3-Thinking-235B-A22B (Team, 2025c) for environment synthesis. Specifically, we synthesize 10 virtual environments containing a total of 1078 high-difficulty tasks. We use Qwen3-Thinking-30B-A3B as the backbone model, and have the model interact with multiple simulated environments to obtain correct trajectories for cold-start SFT. During the reinforcement learning stage, we use GPT-4.1 to simulate users in order to meet high-concurrency demands. The batch size is set to 32, with 8 trajectories rolled out for each sample. Furthermore, we leverage DAPO's dynamic sampling mechanism (Yu et al., 2025) to exclude any samples where all trajectories are either fully correct or fully incorrect. During the evaluation stage, we strictly follow the official evaluation settings of each benchmark, including using GPT-4.1 to simulate users.

B Detailed Prompts

B.1 Masking Out Erroneous User Behaviors

During the RL rollout phase, we use an LLM-as-judge to determine whether the simulated user agent makes mistakes that lead to task failure. The detailed prompt is as follows:

Prompt for Masking Out Erroneous User Behaviors

```
Act as a judge model.
You will receive two inputs:
1. A description of the user's
overall intent.
2. The conversation history between
the user and the assistant.
Your task is to determine whether the
user's actions align with their
overall intent.
Key instructions:
1. Focus on the information in the
user's overall intent. Check whether
the user has returned any incorrect
information.
2. The user should only provide
information that falls within the
scope of their overall intent. Check
whether the user returned any false
or out-of-scope information.
3. When the assistant asks the user
for information, check whether the
user correctly provided information
within the overall intent—or promptly
indicated they did not know the
answer.
4. Your judgment output should be
either "True" or "False":
"True": the user made one or more of
the above mistakes.
"False": the user's performance was
perfect.
```

B.2 Optimized User Prompt

We modified the optimized user prompt implemented in GLM-4.5 (Zeng et al., 2025), with the detailed prompt as follows:

Prompt for Optimized User

```
You are a user interacting with an
agent.{instruction_display}
# Rules:
- Just generate one line at a time to
simulate the user's message.
- Do not give away all the instruction
at once. Only provide the information
that is necessary for the current
step.
- Do not hallucinate information that
is not provided in the instruction.
Follow these guidelines:
1. If the agent asks for information
NOT in the instruction:
- Say you don't remember or don't have
it
- Offer alternative information that
IS mentioned in the instruction
# Constraint Handling: - Provide
requests strictly based on what is
explicitly stated in the instruction.
- Do not assume, extend, substitute,
or generalize in any form.
- Do not modify or relax constraints
on:
- Time / Date
```

```
- Budget
- Specific terms (e.g., “same” must
 not be replaced with “similar”)
- Core Rule: Any attribute NOT
 mentioned in the instruction can be
 either changed or kept the same
 # When NOT to finish the conversation:
- Do not end until you have clearly
 and completely expressed all your
 requirements and constraints.
- Do not end until the agent has
 completed all tasks mentioned in the
 instruction and verified no
 operations were missed.
- Do not end if the agent’s execution
 results do not match your expectations
 or are incorrect/incomplete.
 # When you CAN finish the
 conversation:
- Only when all above conditions are
 satisfied AND all tasks are completed
 correctly.
- OR when you have clearly expressed
 complete requirements but the system
 explicitly states it cannot complete
 them due to technical limitations - in
 this case, accept transfer to human.
 # How to finish the conversation:
- If the agent has completed all
 tasks, generate “###STOP###” as a
 standalone message without anything
 else to end the conversation.
 # Note:
- You should carefully check if the
 agent has completed all tasks
 mentioned in the instruction before
 generating “###STOP###”.
```

	Environment	LLM	Total
Average Time	1	6.04	7.04

Table 5: Comparison of Time Consumption, using the environment’s average execution time as the unit.

C Time Consumption Analysis

To further demonstrate the time efficiency of the simulation environment, we conduct an in-depth comparison of the RL phase, examining the execution time of the simulation environment and the LLM-related time costs, including both rollout and parameter updates.

Based on the experimental results in Table 5, we observe that the environment’s execution time is significantly lower than the LLM-related time, demonstrating the time efficiency of the synthetic environment. Crucially, the wall-clock bottleneck in environmental feedback is the wait for the LLM-based user simulator’s reply; executing the function call and receiving its return is virtually free.