

EnvScaler: Scaling Tool-Interactive Environments for LLM Agent via Programmatic Synthesis

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Zhicheng Dou*, Ji-Rong Wen

Gaoling School of Artificial Intelligence, Renmin University of China.

{songxiaoshuai,dou}@ruc.edu.cn

GitHub: <https://github.com/RUC-NLPIR/EnvScaler>

Abstract

Large language models (LLMs) are expected to be trained to act as agents in various real-world environments, but this process relies on rich and varied tool-interaction sandboxes. However, access to real systems is often restricted; LLM-simulated environments are prone to hallucinations and inconsistencies; and manually built sandboxes are hard to scale. In this paper, we propose EnvScaler, an automated framework for scalable tool-interaction environments via programmatic synthesis. EnvScaler comprises two components. First, SkelBuilder constructs diverse environment skeletons through topic mining, logic modeling, and quality evaluation. Then, ScenGenerator generates multiple task scenarios and rule-based trajectory validation functions for each environment. With EnvScaler, we synthesize 191 environments and about 7K scenarios, and apply them to Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL) for Qwen3 series models. Results on three benchmarks show that EnvScaler significantly improves LLMs' ability to solve tasks in complex environments involving multi-turn, multi-tool interactions.

1 Introduction

Large language models (LLMs) are increasingly expected to serve as agents in a wide range of real-world applications, such as modifying orders in e-commerce backends, rescheduling flights via ticketing platforms, or managing documents in a file system (Luo et al., 2025; Yao et al., 2025; Qian et al., 2025). In these applications, the agent operates within a specific environment (Env), interacting with the user to gather information and invoking tools to query or update the Env's state, as illustrated in Figure 1. This challenges LLMs to combine dialogue and tool use, adapt actions based on Env feedback, and solve tasks while respecting Env rules over long-horizon trajectories.

*Corresponding Author

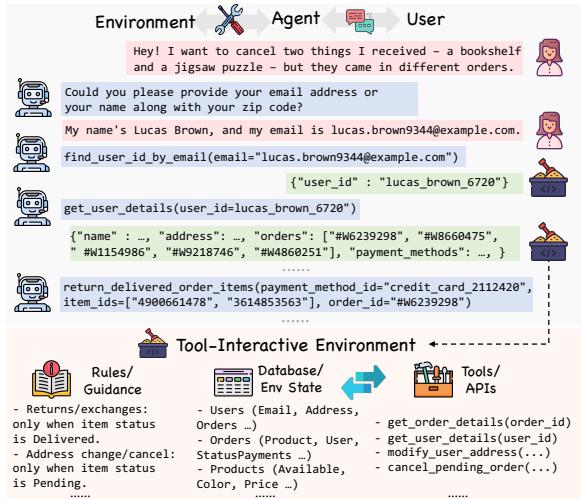


Figure 1: An illustration of tool-interactive environments. The environment (1) defines rules and provides tool interfaces to the agent; (2) executes the agent's tool calls to update its state and return results.

To develop such capable LLM agents, scaling up rich and diverse tool-interactive environments is essential. Whether by collecting trajectories followed by imitation learning, or by autonomous exploration and reinforcement learning (RL) within Envs, we hope that exposure to a sufficiently broad range of environments during training will enable LLMs to generalize effectively to unseen environments and scenarios at test time (Huang et al., 2025; Liu et al., 2025a; Andrews et al., 2025). However, as compared in Table 1, real-world environments often have restricted access; LLM-simulated environments also suffer from hallucinations and inconsistencies. Recently, a series of studies (Patil et al., 2025; Yao et al., 2025; Lu et al., 2025) build stateful, tool-interactive sandboxes through executable programs, offering advantages in controllability and stability. Nonetheless, these environments are manually crafted for evaluation purposes, with limited coverage and scalability. Therefore, a key challenge lies in automating the synthesis and scaling of sandbox environments to support training. It re-

Env Type	Scalable	Consistent	Controllable	Stable	Explainable
Real-World	✗	✓	✗	✓	✓
LLM-Simulated	✓	✗	✓	✗	✗
Programmatic	✓	✓	✓	✓	✓

Table 1: Key property comparison of three Env types for LLM training. Scalable: ease of large-scale expansion; Consistent: logical coherence between multiple calls; Controllable: flexibility in modifying Env logic; Stable: reproducible over time; Explainable: transparency of Env logic. Symbols denote: ✓ full support, ✗ not supported, ↗ partial or conditional support.

quires creating diverse, high-quality environments with states, tools, and interaction logic, and designing tasks that align with each environment.

Several studies have made progress in tackling this challenge, with LLMs used as programmers of environment logic rather than direct simulators. One approach (Ye et al., 2025; Sullivan et al., 2025) focuses solely on tool-layer modeling. It does not model the sandbox’s state, nor consider the interaction logic between tools and the database. Another approach (Tang et al., 2024; Piriyakulkij et al., 2025) seeks to programmatically reconstruct environments from existing observations (e.g., trajectories), but inevitably depends on access to pre-existing environments. Besides, AgentScaler (Fang et al., 2025) and AutoForge (Cai et al., 2025) rely on pre-collected toolsets or tool documentation, and lack an automated mechanism for assessing environment quality. Due to these limitations, a notable gap remains in automatically synthesizing and scaling tool-interactive environments without relying on environmental priors or toolsets. To bridge this gap, we propose **EnvScaler**, an automated, scalable framework for synthesizing diverse, executable, tool-interactive environments to train LLM agents, as shown in Figure 2.

We first introduce **SkelBuilder** to automate the construction of environmental skeletons, covering topic mining, logic modeling, and assessment. It comprises three modules: (1) Task-driven environment discovery: mines diverse environment themes from existing open-source task sets. (2) Executable environment construction: starting from an environment description, it plans states and tools, and programmatically implements them into a complete, runnable environment. (3) Quality inspection: a testing agent sends tool requests, while a checking agent assesses whether executions meet expectations. This process iterates over multiple rounds,

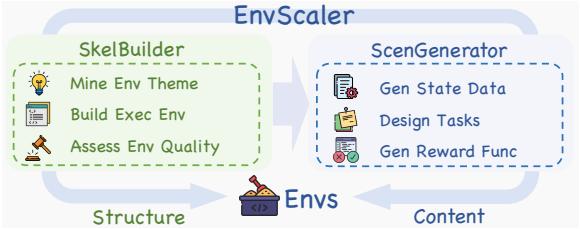


Figure 2: The overview of EnvScaler.

with the pass rate indicating environment quality.

To further synthesize multiple task scenarios for each environment, we propose **ScenGenerator**. To ensure task relevance and solvability within a given environment and scenario, ScenGenerator first synthesizes the environment’s initial database/state, and then derives challenging tasks from the current state. To achieve rule-based trajectory verification, ScenGenerator generates a set of terminal-state validation functions for each task. After the trajectory ends, these functions check whether the final environment state meets the expected conditions, using the functions’ pass rate as the reward score.

To validate the effectiveness of EnvScaler, we synthesized 191 environments and about 7K scenarios, applying them to SFT and RL for the Qwen3 series models. Evaluation on multiple tool-use benchmarks (Patil et al., 2025; Yao et al., 2025; Chen et al., 2025) shows that EnvScaler significantly enhances LLMs’ ability to solve tasks in complex environments involving multi-turn, multi-tool interactions. Further analysis of environment coverage, scale, and training strategies provides insights into how synthetic environments promote tool learning and generalization for LLM agents.

In summary, we propose EnvScaler for scalable tool-interactive environment synthesis. Our contributions are threefold: (1) We propose SkelBuilder, an automated framework for synthesizing diverse, executable environment skeletons. (2) We propose ScenGenerator, a scenario generation pipeline that produces state data, challenging tasks, and rule-based trajectory verification for each environment. (3) Experiments on three benchmarks verify the effectiveness of EnvScaler in improving LLMs’ ability to solve tasks in complex environments involving multi-turn, multi-tool interactions.

2 Related Work

2.1 Tool Use of LLMs

Many studies aim to improve LLMs’ ability to solve tasks with tools (Qu et al., 2025; Luo et al.,

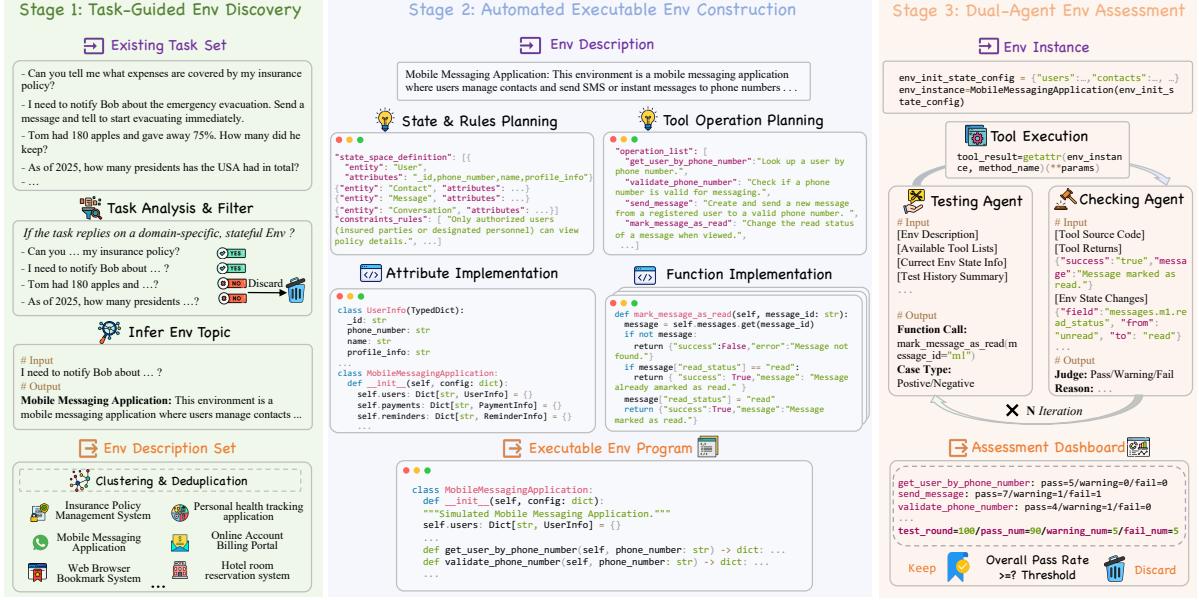


Figure 3: The overall framework of SkelBuilder.

2025). In this paper, we focus on general tool use across various domain-specific environments (Patil et al., 2025; Yao et al., 2025; Chen et al., 2025), rather than tool-integrated reasoning and web information access centered on Python or search tools (Dong et al., 2025; Li et al., 2025a). Some work have explored the training data and RL strategies from different perspectives (Prabhakar et al., 2025; Liu et al., 2025b; Xu et al., 2025; Zhang et al., 2025; Zhao et al., 2025). However, they mainly focus on synthetic static trajectories and cannot support LLMs’ self-exploration. For trajectory evaluation, they primarily rely on surface matching, checking whether generated tool names and parameters match references, which is neither sufficient to determine whether the task is truly completed nor able to accommodate multiple equivalent solution paths. In contrast, we synthesize executable environments and tasks, along with rule-based evaluation grounded in environments’ state, thereby supporting LLMs’ training across varied scenarios.

2.2 Scaling Environments for LLM Agent

Environments provide agents with action feedback and rewards for interaction and policy optimization. We focus on tool-interactive environments, where LLM agents can use tools to query environmental information or change the state of the environment. One line of work (Guo et al., 2024, 2025; Castellani et al., 2025; Li et al., 2025b) leverages LLMs’ reasoning and world knowledge to simulate environments. Although there is no need to build

real environments, it is prone to hallucinations and inconsistencies, and lacks transparency and persistent state management. Another line of work (Tang et al., 2024; Ye et al., 2025; Fang et al., 2025; Cai et al., 2025) builds sandbox environments through programming. However, they either only model isolated, stateless functions, or rely on environmental priors (e.g., trajectories, toolsets) and lack automatic assessment, which limits scalability and coverage. Therefore, we propose EnvScaler to enable automatic, scalable environment and scenario synthesis for agent training.

3 Automated Env Skeleton Synthesis

Overview. The goal of SkelBuilder is to construct environments $\{E\}$, where each can be abstracted as a set of three elements: $E = \{F_{\text{exec}}, E_{\text{doc}}, \Sigma_{\text{tool}}\}$.

- Executable program files F_{exec} : Complete logic implementation of E ’s states, tools, and rules.
 - Documentation E_{doc} : Provides the agent with introductions or rules about E .
 - Tool interface set Σ_{tool} : Names, parameters, and descriptions of all tools exposed to the agent, serving as the entry for agent–Env interaction.
- As shown in Figure 3, SkelBuilder enables an automated workflow from text resource mining to environment modeling and evaluation.

3.1 Task-Guided Env Discovery

The first step in scaling environments is to collect diverse environment themes. Unlike manual pre-

setting or derivation from API collections (Fang et al., 2025), SkelBuilder mines them from existing text resources. Considering that studies around SFT have gathered a large and diverse set of tasks that may implicitly contain latent environmental contexts, this inspired us to derive themes through reverse inference from the existing tasks.

Given a task set $T_{\text{exist}} = \{t_1, \dots, t_n\}$, an LLM M first performs binary filtering to retain tasks situated within a domain-specific, stateful environment. For each retained task, M infers the corresponding environment description:

$$\{E'_{\text{des}}\} = \{M(P_{\text{infer}}^{\text{env}} || t) \mid t \in T_{\text{exist}}, M(P_{\text{filter}}^{\text{task}} || t)\}, \quad (1)$$

where $P_{\text{filter}}^{\text{task}}$ and $P_{\text{infer}}^{\text{env}}$ denote prompts for task filtering and environment inference¹. The inferred environments are then aggregated and deduplicated by embedding each description and retaining one record from groups of highly similar descriptions, yielding the final diverse, non-redundant set $\{E_{\text{des}}\} = \text{Dedup}(\{E'_{\text{des}}\}, \text{sim})$.

3.2 Automated Executable Env Construction

To transform the environment description into a programmatically modeled environment, we design a three-stage pipeline.

Logic Planning. An LLM enriches the environment description E_{des} , inferring the Env state definition E_{state} , domain rules E_{rule} , and the list of tool operations $\{E_{\text{tool}_i}\}$. These elements serve as a structured blueprint, with E_{rule} concatenated with E_{des} to form the environment documentation E_{doc} :

$$E_{\text{state}}, E_{\text{rule}} = M(P_{\text{plan}}^{\text{state}} || E_{\text{des}}), \\ \{E_{\text{tool}_i}\} = M(P_{\text{plan}}^{\text{tool}} || E_{\text{des}} || E_{\text{state}} || E_{\text{rule}}). \quad (2)$$

Program Modeling. The LLM first converts the planned state space into class attribute definitions F_{attr} . Then, for each tool operation, given the environment rules and class attributes, it generates the corresponding class-method implementation F_{meth_i} , ensuring consistency with rules and proper state transitions:

$$F_{\text{attr}} = M(P_{\text{exec}}^{\text{state}} || E_{\text{state}}), \\ F_{\text{meth}_i} = M(P_{\text{exec}}^{\text{tool}} || E_{\text{rule}} || F_{\text{attr}} || E_{\text{tool}_i}). \quad (3)$$

Program Assembly. The generated code fragments are automatically merged into a complete Python class file F_{exec} , implementing all sandbox

¹We denote the prompt for LLMs by P and use the notation henceforth. The content of P is shown in Appendix A.

logic, where attributes represent environment states and methods represent supported tool operations:

$$F_{\text{exec}} = \text{Merge}(F_{\text{attr}}, \{F_{\text{meth}_i}\}_{i=1}^m). \quad (4)$$

Syntax validity is verified via the abstract syntax tree (AST), with invalid files discarded. AST combined with regex extraction yields all method signatures, forming the tool interface set Σ_{tool}^2 .

3.3 Dual-Agent Env Assessment

Unlike static evaluation via direct LLM scoring, we propose a dual-agent loop to assess the actual tool-execution performance of the environment.

Frontend Testing Agent. After instantiating the environment class, the testing LLM agent M_{test} has no access to its internal implementation. In round j , it receives the current environment state S_j , and randomly generates a call request, which may be a positive or a negative test case (e.g., invoking a file-deletion tool to delete a non-existent file):

$$\text{call}_j = M_{\text{test}}(\Sigma_{\text{tool}}, S_j). \quad (5)$$

Backend Checking Agent. The environment first executes the tool invocation. The checking agent M_{check} , inspects the tool’s source code F_{meth} , the returned result R_j , and state changes before and after execution to judge whether the behavior matches expectations:

$$R_j, S_{j+1} = \text{Exec}(F_{\text{meth}}, \text{call}_j, S_j), \\ \text{judge}_j = M_{\text{check}}(\text{call}_j, R_j, \Delta S_{j \rightarrow j+1}). \quad (6)$$

The testing and checking agents form a closed loop, iterating for N rounds to cover diverse situations. The average judging pass rate serves as the quantitative metric for the environment’s quality:

$$\text{score}_{\text{env}} = \frac{1}{N} \sum_{j=1}^N \text{judge}_j, \quad (7)$$

where environments with $\text{score}_{\text{env}}$ below a predefined threshold are discarded.

3.4 Practical Analysis

In practice, we select API-Bank (Li et al., 2023) and ToolACE (Liu et al., 2025b) as initial task sources, which have high task retention rates. GPT-4.1 and Qwen3-235B-Instruct-2507 are used for environment discovery and programming, while GPT-4.1-mini and Qwen3-30B-A3B-Instruct-2507

²We provide the example of F_{exec} , Σ_{tool} in Appendix A.5.

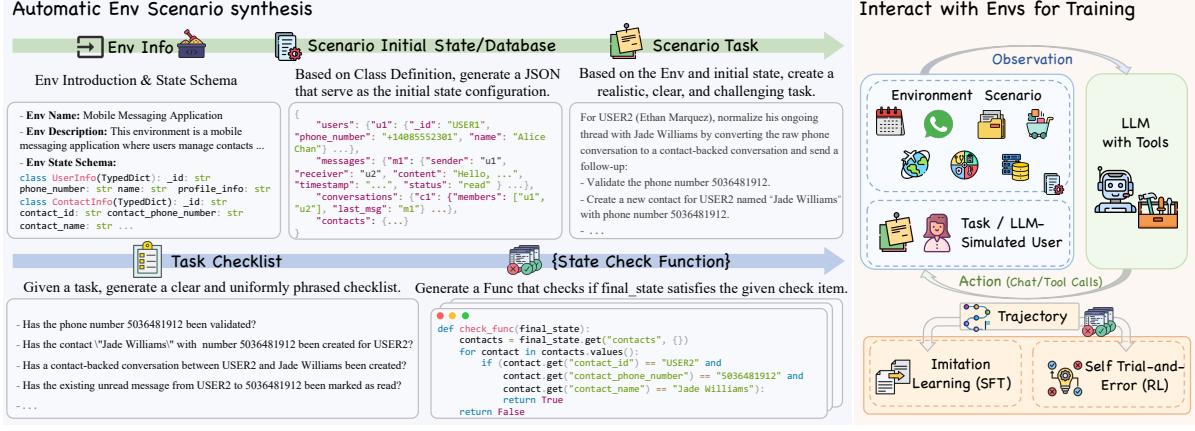


Figure 4: The overall framework of ScenGenerator.

Item	Avg.	Med.
# Constraint Rules Per Env	4.58	5
# State Category Per Env		
Level 1 (e.g., user, message, item)	3.74	4
Level 2 per Level 1 (e.g., u_id, u_phone)	5.72	5
Total	21.38	21
# Tools Per Env		
Env Information Query (e.g., list_users)	10.44	10
Env State Change (e.g., send_message)	8.14	8
Total	18.58	18

Table 2: Statistics across 191 synthetic environments.

are used for environment assessment. The number of test rounds N is set to 100, with a filtering threshold of 0.85. In total, we obtain 191 environments. Table 2 and Figure 8 presents their statistics, with an average of 18.58 tools and 21.38 state categories per environment, reflecting the complexity of the synthesized environments.

4 Automated Env Scenario Synthesis

Overview. The skeleton of the environment alone is insufficient to support agent interactions. An environment also requires initial state data, tasks, and an evaluation mechanism for assessing agents’ action trajectories. As shown in Figure 4, we propose ScenGenerator to automatically construct multiple task scenarios for each environment.

4.1 Env Initial State & Task Generation

In stateful environments, tasks are closely tied to state data. For example, an agent cannot cancel an order that does not exist in the environment’s database. To ensure that the generated task is both solvable and consistent with the given environment and scenario, ScenGenerator first uses an LLM to

generate the environment’s initial state data S_{init} , and then derives challenging tasks based on the environment’s initial state:

$$S_{\text{init}} = M(P_{\text{init}}^{\text{gen}} || F_{\text{exec}} || E_{\text{state}}). \quad (8)$$

A common task synthesis approach (Fang et al., 2025; Cai et al., 2025) is to walk through tool invocation sequences and infer the corresponding task by reversing the sequence. However, this often yields low-quality tasks, and the existence of multiple valid task solution paths also make the sequence unsuitable as a unique “ground-truth” reference. Therefore, we directly prompt the LLM to generate a challenging, scenario-specific task from the initial state S_0 , tool set E_{tool} , and rules E_{rule} :

$$\text{task} = M(P_{\text{task}}^{\text{gen}} || S_{\text{init}} || E_{\text{tool}} || E_{\text{rule}}). \quad (9)$$

4.2 Validation Function Generation

To achieve rule-based trajectory verification, we first use an LLM to decompose the task into a checklist of verifiable conditions. Then, for each checkpoint, we call the LLM to generate a terminal-state validation function. This function takes the environment’s final state S_{final} after the trajectory as input and returns “True” or “False” to indicate whether the condition is satisfied. Finally, the proportion of passed functions is used as the trajectory’s reward score. Formally, we have:

$$\begin{aligned} \{c_k\}_{k=1}^K &= M(P_{\text{list}}^{\text{check}} || \text{task}), \\ f_{c_k} &= M(P_{\text{func}}^{\text{check}} || c_k), \\ \text{reward} &= \frac{1}{K} \sum_{k=1}^K \mathbf{1}[f_{c_k}(S_{\text{final}}) = \text{True}]. \end{aligned} \quad (10)$$

Compared with a single boolean judgment, decomposing the task into multiple validation functions

	Qwen3 -4B(NT)	Qwen3 -4B	Qwen3 -8B	Qwen3-8B (max@8)	Qwen3 -30B
Non-Conversation					
Score	37.53	53.05	57.78	69.38	67.07
Avg Step	21.36	12.96	13.66	-	16.78
Conversation					
Score	46.13	55.56	58.35	74.48	63.08
Avg Step	27.20	25.48	25.90	-	25.38

Table 3: Average scores and step counts of LLMs on a random sample of 50 scenarios. Qwen3-30B is Qwen3-30B-A3B-Thinking-2307. NT: Non-Think.

not only captures partial completion, but also is easier for the LLM to generate. Moreover, unlike superficial evaluation that merely checks whether the tool invocation sequence matches a reference sequence, evaluation based on the environment’s final state is agnostic to the solution process and can accommodate multiple valid solution paths.

4.3 Interact with Envs for Training

The agent–environment interaction can be modeled as a Partially Observable Markov Decision Process (POMDP). At time step t , the agent π_θ cannot directly access the environment state data S_t ; instead, it makes decisions based on the observation–action history H_t and the current observation o_t :

$$a_t = \pi_\theta(H_t, o_t) \quad o_{t+1}, S_{t+1} = E(a_t, S_t). \quad (11)$$

We consider two interaction settings:³

(1) **Non-Conversation (Non-Conv.)**: The environment directly provides the complete task information to the agent, with the initial observation $o_0 = \{E_{\text{doc}}, \Sigma_{\text{tool}}, \text{task}\}$. At each step, the agent invokes a tool, and the execution result as the observation, until the agent believes the task is completed or the maximum step limit is reached.

(2) **Conversation (Conv.)**: The environment additionally includes an LLM-simulated user π_{user} . Complete task information is not revealed upfront but must be progressively acquired via agent-user dialogue. Compared with Non-Conv, it expands the agent’s action space to include user interaction. The trajectory ends when the user considers the task completed or the maximum steps are reached.

For SFT, trajectories from the teacher LLM can be directly used as learning targets for the student LLM. For RL, trajectories are converted into rewards using the validation functions, which are then applied for policy optimization.

³The trajectory examples are shown in Appendix A.7.

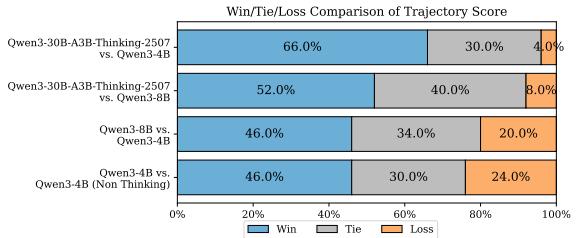


Figure 5: Pairwise comparison of different LLMs on a random sample of 50 scenarios under Non-Conv setting.

4.4 Practical Analysis

With ScenGenerator, we construct around 7K task scenarios for 191 environments and randomly sample 50 scenarios for a pilot study. As shown in Table 3, Qwen3-8B (Thinking) scores 57.78, while Qwen3-4B (Non-Think) scores only 37.53. Besides, the average trajectory is about 15 steps under the Non-Conv setting, increasing to over 25 steps under Conv. This shows that the synthetic tasks have substantial difficulty and can incentivize LLMs to produce long action trajectories. Moreover, for the same model, multiple sampling and selecting the highest-scoring trajectory significantly boost the score, suggesting that synthetic tasks offer considerable room for self-exploration and optimization. Figure 5 shows that stronger LLMs consistently achieve higher win rates, confirming the validity of the state-check functions in distinguishing and quantifying model performance.

5 Experiments

5.1 Experiment Setup

Training. We conduct SFT and RL on Qwen3 series models (Thinking Mode) (Yang et al., 2025). A total of 140 environments are used for SFT, and the remaining 51 environments are used for RL. For SFT, we employ Qwen3-30B-A3B-Thinking-2507 as the teacher model, interacting with the environments under two interaction settings, yielding about 9K trajectories in total. For RL, we apply the Reinforce++ algorithm (Hu et al., 2025) under Non-Conv Setting. Detailed implementation are provided in Appendix B.2.

Evaluation. We use three widely used multi-turn tool-use benchmarks: BFCL-v3 Multi-Turn (abbreviated as BFCL-MT) (Patil et al., 2025), Tau-Bench (Yao et al., 2025), and ACEBenchmark-Agent (Chen et al., 2025). Each benchmark consists of several domain-specific environments equipped with tools, requiring the LLM to interact

Model	BFCL-v3 Multi-Turn					Tau-Bench			ACEBench-Agent		
	Base	Miss-Func	Miss-Param	Long-Context	Overall	Retail	Airline	Overall	Multi-Step	Multi-Turn	Overall
Advanced Models											
GPT-4.1	46.00	37.50	32.00	44.50	40.00	66.95	48.0	57.48	95.00	60.00	77.50
Qwen3-235B-Thinking-2507	60.00	35.00	34.00	54.00	45.75	67.8	46.00	56.90	85.00	63.33	74.17
Qwen3-235B-Instruct-2507	58.00	33.00	27.50	50.50	42.25	71.30	44.00	57.65	80.00	63.33	71.67
Kimi-K2-Instruct-0905	57.50	35.00	42.00	49.00	45.88	69.87	54.00	61.94	85.00	73.33	79.17
Qwen3-1.7B (Thinking)	13.50	6.00	12.50	7.00	9.75	8.99	16.00	12.50	35.0	28.89	31.95
+ EnvScaler (w/ SFT)	24.50	11.50	20.00	16.50	18.13_{+8.38}	20.87	14.00	17.44_{+4.94}	55.00	32.22	43.61_{+11.66}
+ EnvScaler (w/ SFT&RL)	31.50	17.00	20.50	23.00	23.00_{+13.25}	18.55	14.00	16.28_{+3.78}	66.67	33.33	50.00_{+18.05}
Qwen3-4B (Thinking)	32.00	20.00	24.00	25.50	25.38	40.87	26.00	33.44	58.33	52.22	55.28
+ EnvScaler (w/ SFT)	47.00	24.00	31.50	37.00	34.88_{+9.50}	44.35	32.00	38.20_{+4.76}	73.33	60.00	66.67_{+11.39}
+ EnvScaler (w/ SFT&RL)	51.00	34.00	28.00	39.00	38.00_{+12.62}	48.12	34.00	41.06_{+7.62}	80.00	61.11	70.55_{+15.27}
Qwen3-8B (Thinking)	32.00	33.50	22.00	28.00	28.88	46.38	30.00	38.19	63.33	56.67	60.00
+ EnvScaler (w/ SFT)	47.00	33.00	29.50	38.50	37.00_{+8.12}	48.70	34.00	41.35_{+3.16}	83.33	60.00	71.67_{+11.67}
+ EnvScaler (w/ SFT&RL)	55.50	36.00	35.00	41.00	41.88_{+13.00}	53.62	36.00	44.81_{+6.62}	85.00	60.00	72.50_{+12.50}

Table 4: Performance comparison of models on three benchmarks. We bold the Overall results for each benchmark.

with users and invoke tools to solve tasks. Notably, ACEBench uses the [func_name(param)] prompt format by default; we modify the official code to support LLMs’ native function-calling interface to ensure consistency. Detailed introductions are provided in Appendix B.1 and Table 7.

5.2 Main Results

As shown in Table 4, using EnvScaler for SFT leads to significant improvements across all benchmarks, and incorporating RL further boosts performance⁴. Next, we analyze the results from three aspects:

(1) **Training Strategy.** SFT significantly boosts LLMs’ performance. Averaged across three models, BFCL-MT improves by 8.67 points, Tau-Bench by 4.29 points, and ACEBench-Agent by 11.57 points. This can be attributed to the diverse, multi-tool environments and complex tasks synthesized by EnvScaler, which help LLMs strengthen domain adaptability, multi-turn interaction, and multi-tool usage in supervised training. Incorporating RL yields further improvements (the RL curve is shown in Figure 6). For example, Qwen3-8B achieves 4.88 and 3.46 point gains on BFCL-MT and Tau-Bench, respectively. This demonstrates that leveraging synthetic environments and state-checking reward signals can guide LLMs to autonomously learn more effective strategies.

(2) **Model Size.** SFT delivers stable improvements across models, whereas RL performance is more model-dependent. Qwen3-8B achieves

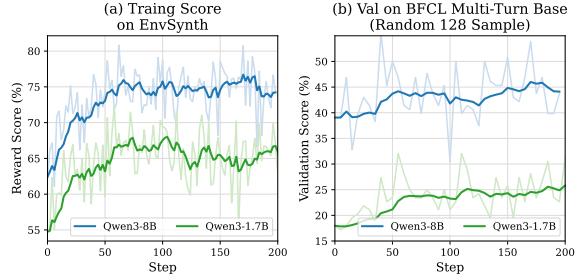


Figure 6: The RL training and validation curve of Qwen3 in synthetic environments after SFT.

gains across all datasets, whereas Qwen3-1.7B shows notable improvements on BFCL-MT and ACEBench-Agent but a slight drop on Tau-Bench. The main reason is that large-scale models possess stronger exploration capabilities during RL, enabling them to extract effective strategies. In contrast, small-scale models, with weaker foundational abilities, are more susceptible to noisy reward signals and produce lower-quality strategies that struggle to generalize to unseen environments.

(3) **Different Benchmarks.** Comparatively, EnvScaler shows more substantial gains on BFCL-MT and ACEBench-Agent, while improvements on Tau-Bench are relatively limited. Qwen3-1.7B even slightly declines on the most challenging Tau-Bench Airline task. This is mainly because BFCL-MT and ACEBench-Agent underscore the evaluation of LLM’s ability in multi-turn, multi-tool collaboration and domain adaptability, while EnvScaler can substantially enhance these capabilities. In contrast, Tau-Bench focuses on deep reasoning

⁴The experiments on EnvScaler in non-thinking mode, as well as direct RL, are presented in Appendix C.

	Base	Miss-Func	Miss-Parm	Long-Cont.	Overall all
Qwen3-4B	32.00	20.00	24.00	25.50	25.38
+ SFT (Full)	47.00	24.00	31.50	37.00	34.88
+ SFT (top 50%)	40.50	28.00	27.50	32.00	32.00
+ SFT (bottom 50%)	43.00	27.50	27.50	32.00	32.50
+ SFT (random 50%)	42.00	26.00	26.00	31.00	31.25

Table 5: Qwen3-4B’s Performance on BFCL-MT with training on different environment-similarity subsets.

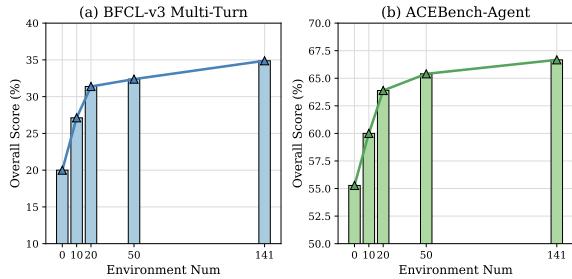


Figure 7: The change of Qwen3-4B’s performance with the scaling of the number of environments for SFT.

under complex environment’s rules, making such tasks harder for LLMs to learn and generalize.

5.3 Train-Test Env Similarity Analysis

To investigate how the similarity between synthetic and test environments affects test performance, we compute the similarity between each synthetic environment and those in BFCL-MT, based on textual embeddings of the environment topic and toolset descriptions. According to the similarity scores, environments are divided into three groups: (1) the 50% most similar; (2) the 50% least similar; and (3) a random 50% selection. As shown in Table 5, training on either the most similar or least similar subsets yields only minor differences in overall metrics, and both significantly outperform the baseline. This suggests that the performance gains do not primarily depend on direct similarity between training and test environments, but on the problem-solving patterns and transferable tool-use skills learned from different environments.

5.4 Effect of Scaling Environment

To analyze the effect of scaling the number of training environments on model performance, we sample subsets with varying environment counts from the full SFT dataset and conduct SFT on each subset. As shown in Figure 7, Qwen3-4B’s scores on both the BFCL-MT and ACEBench-Agent exhibit a steady upward trend as the number of training

	Base	Miss-Func	Miss-Parm	Long-Cont.	Overall all
Qwen3-8B	32.00	33.50	22.00	28.00	28.88
+ SFT (Non-Conv)	49.50	32.50	23.50	37.50	35.75
+ SFT (Conv)	45.50	32.00	30.50	34.00	35.50
+ SFT (Full)	47.00	33.00	29.50	38.50	37.00

Table 6: Qwen3-8B’s Performance on BFCL-MT with training on different subsets of interaction patterns.

environments increases. The most pronounced improvement occurs when scaling from 0 to 20 environments; although the rate of gain slows beyond this point, performance continues to rise overall. These results indicate that scaling up the number and diversity of training environments effectively enhances the model’s adaptability across contexts and improves its task-solving performance.

5.5 Effect of Interaction Patterns

To investigate the impact of interaction patterns on model training, we conduct SFT under three data settings: Non-Conv, Conv, and their combination (Full). Table 6 reports the BFCL-MT results for Qwen3-8B. The Base and Long-Context subsets have complete task information, whereas the Miss-Func and Miss-Parm subsets lack necessary tools or parameters, requiring the agent to interact with the user to obtain them. It can be observed that SFT (Non-Conv) performs better on Base and Long-Context, while SFT (Conv) achieves higher scores on Miss-Parm. Performance on Miss-Func does not improve, primarily because the training data lacks samples with missing tool types. The SFT (Full) setting yields the highest overall score, highlighting the need to learn both interaction patterns for adaptability in scenarios with either sufficient or incomplete information.

6 Conclusion

In this paper, we propose EnvScaler, a scalable framework for synthesizing diverse, executable tool-interactive environments to train LLM agents. EnvScaler first constructs environment skeletons through topic mining, programmatic modeling, and dual-agent assessment (SkelBuilder), then synthesizes scenarios by generating initial states, challenging tasks, and rule-based trajectory verification (ScenGenerator). Experiments on three benchmarks demonstrate that EnvScaler significantly enhances LLMs’ ability to solve tasks through multi-turn, multi-tool interactions across environments.

Limitations

In this paper, we propose EnvScaler for automated synthesis of tool-interactive environments. However, there are still some limitations as follows: (1) Although LLMs are not directly used as environment simulators, the entire construction process relies on LLM synthesis, which may introduce biases compared to real systems (e.g., business logic, state definitions, and task design). (2) In terms of domain coverage, EnvScaler mainly targets domain-specific, stateful environments, with limited support for open environments such as web search or information access. (3) In terms of feature modeling, EnvScaler focuses on tool-state interactions but lacks explicit simulation of real-system characteristics such as interface latency, network fluctuations, and error patterns. (4) In terms of modality, EnvScaler only supports text-based tool inputs and outputs, without incorporating multimodal tools involving images, audio, or other modalities.

References

- Pierre Andrews, Amine Benhalloum, Gerard Moreno-Torres Bertran, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Romain Froger, Emilien Garreau, Jean-Baptiste Gaya, and 1 others. 2025. Are: Scaling up agent environments and evaluations. *arXiv preprint arXiv:2509.17158*.
- Shihao Cai, Runnan Fang, Jialong Wu, Baixuan Li, Xinyu Wang, Yong Jiang, Liangcai Su, Liwen Zhang, Wenbiao Yin, Zhen Zhang, Fuli Feng, Pengjun Xie, and Xiaobin Wang. 2025. *Autoforge: Automated environment synthesis for agentic reinforcement learning*. Preprint, arXiv:2512.22857.
- Tommaso Castellani, Naimeng Ye, Daksh Mittal, Thomson Yen, and Hongseok Namkoong. 2025. *Synth-tools: A framework for scaling synthetic tools for agent development*. Preprint, arXiv:2511.09572.
- Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzhi, and Wu Liu. 2025. *ACEBench: A comprehensive evaluation of LLM tool usage*. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12970–12998, Suzhou, China. Association for Computational Linguistics.
- Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*.
- 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*.
- Shangmin Guo, Omar Darwiche Domingues, Raphaël Avalos, Aaron Courville, and Florian Strub. 2025. *World modelling improves language model agents*. Preprint, arXiv:2506.02918.
- 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*. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156, Bangkok, Thailand. Association for Computational Linguistics.
- Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. 2025. *Reinforce++: Stabilizing critic-free policy optimization with global advantage normalization*. Preprint, arXiv:2501.03262.
- Yuchen Huang, Sijia Li, Zhiyuan Fan, Minghao LIU, Wei Liu, and Yi R. Fung. 2025. *Scaling environments for LLM agents: Fundamentals, approaches, and future directions*. In *Workshop on Scaling Environments for Agents*.
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. *API-bank: A comprehensive benchmark for tool-augmented LLMs*. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore. Association for Computational Linguistics.
- Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yongkang Wu, Ji-Rong Wen, Yutao Zhu, and Zhicheng Dou. 2025a. *Webthinker: Empowering large reasoning models with deep research capability*. arXiv preprint arXiv:2504.21776.
- Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. 2025b. *Simulating environments with reasoning models for agent training*. Preprint, arXiv:2511.01824.
- Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025a. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.
- Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, and 8 others. 2025b. *ToolACE: Winning the points of LLM function calling*. In *The Thirteenth International Conference on Learning Representations*.

- Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, and 1 others. 2024. Apigen: automated pipeline for generating verifiable and diverse function-calling datasets. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, pages 54463–54482.
- Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. **ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities**. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183, Albuquerque, New Mexico. Association for Computational Linguistics.
- Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, and 1 others. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.
- Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. **The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models**. In *Forty-second International Conference on Machine Learning*.
- Wasu Top Piriyakulkij, Yichao Liang, Hao Tang, Adrian Weller, Marta Kryven, and Kevin Ellis. 2025. Poe-world: Compositional world modeling with products of programmatic experts. *arXiv preprint arXiv:2505.10819*.
- Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgarkar, 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, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, and Huan Wang. 2025. **Userbench: An interactive gym environment for user-centric agents**. In *Workshop on Scaling Environments for Agents*.
- Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343.
- Michael Sullivan, Mareike Hartmann, and Alexander Koller. 2025. **Procedural environment generation for tool-use agents**. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18555–18573, Suzhou, China. Association for Computational Linguistics.
- Hao Tang, Darren Key, and Kevin Ellis. 2024. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. *Advances in Neural Information Processing Systems*, 37:70148–70212.
- Weixun Wang, Shaopan Xiong, Gengru Chen, Wei Gao, Sheng Guo, Yancheng He, Ju Huang, Jiaheng Liu, Zhendong Li, Xiaoyang Li, and 1 others. 2025. Reinforcement learning optimization for large-scale learning: An efficient and user-friendly scaling library. *arXiv preprint arXiv:2506.06122*.
- Zengzhuang Xu, Bingguang Hao, Zechuan Wang, Yun-tao Wen, Xinyi Xu, Yang Liu, Long Chen, Dong Wang, Maolin Wang, Tong Zhao, Yicheng Chen, Cunyin Peng, Jinjie Gu, Leilei Gan, Xiangyu Zhao, Chenyi Zhuang, and Shi Gu. 2025. **Funreason-mt technical report: Advanced data synthesis solution for real-world multi-turn tool-use**. *Preprint, arXiv:2510.24645*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengan Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. **{\\$tau\\$}-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains**. In *The Thirteenth International Conference on Learning Representations*.
- 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*.
- Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhidong Yu, and Guilin Liu. 2025. **Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning**. *Preprint, arXiv:2505.00024*.
- 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**. *Preprint, arXiv:2508.18669*.
- Yaowei Zheng, Richong Zhang, Junhao Zhang, YeYanhan YeYanhan, and Zheyuan Luo. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410.

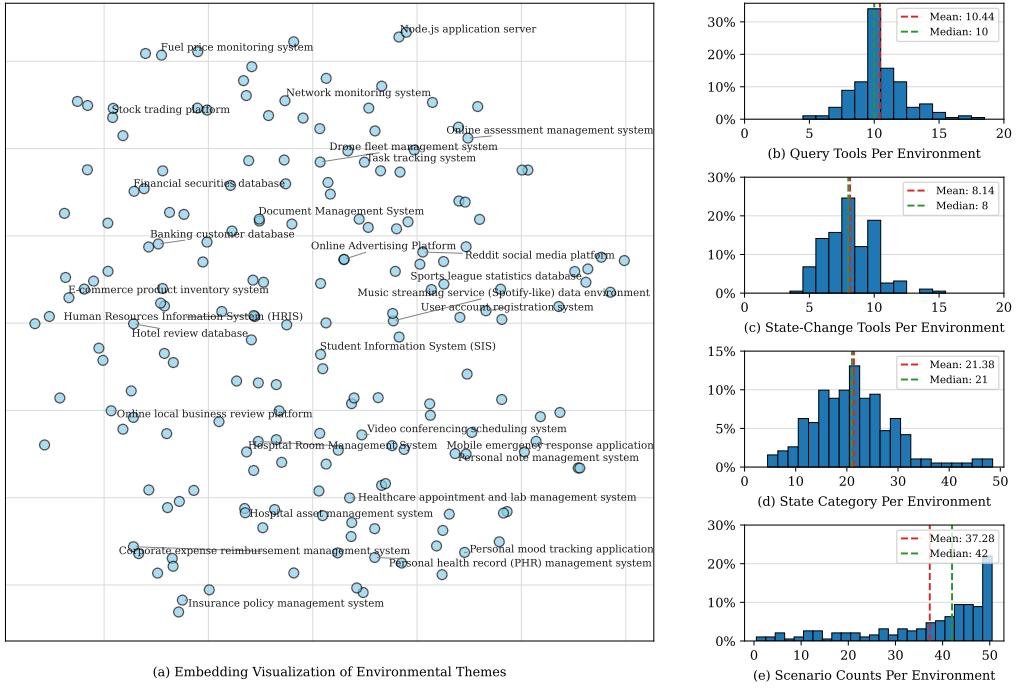


Figure 8: Diversity and statistical distributions of 191 synthesized environments.

A Details of EnvScaler

A.1 Details of Prompts for SkelBuilder

- Figures 10 and 11 show the prompts for task filtering and environment-theme inference in the Task-driven Environment Discovery stage.
- Figures 12 and 13 present the prompts used to plan environment states, rules, and operations during the environment-modeling stage, while Figures 14 and 15 show the prompts for converting these blueprints into executable programs.
- Figures 16 and 17 show the system prompts for the testing agent and checking agent in the Dual-Agent Env Assessment stage, respectively.

A.2 Details of Prompts for ScenGenerator

Figures 18 and 19 show the prompts for generating the environment’s initial state and tasks, respectively. Figures 20 and 21 show the prompts for decomposing tasks into checklists and converting each check item into a verification function.

A.3 Details of Prompts for LLM Agents

Figures 22 and 23 show the system prompts guiding LLMs complete tasks under the *Non-conversation* and *Conversation* settings. Figure 24 shows the system prompt for guiding a LLM to act as a user.

A.4 Details and Statistics of 191 Synthesized Environment

Figure 8 shows detailed statistics for the 191 synthesized environments. By embedding each environment’s theme description and applying t-SNE, we obtain Figure 8(a), where the point dispersion demonstrates the diversity of environment themes. Figures 8(b) and (c) show the distribution of tool counts across environments, while Figure 8(d) shows the distribution of state categories. Figure 8(e) shows synthesized scenario counts per environment after removing unqualified scenario.

A.5 Example of Synthesized Environment

In Example 1, we provide a program example of synthesized environments, and Table 10 shows the corresponding tool interfaces.

A.6 Example of Environment’s Task Scenario

- Initial State Configuration:** Figure 25 shows an example of an initial state configuration, which is used to initialize the environment’s state data.
- Task:** Figure 26 presents an example task under this initial state.
- State Check Functions:** Figure 11 shows the checklist associated with the task, along with the check function for each checkpoint, which are used to compute a trajectory’s reward score.

A.7 Example Trajectory of LLM Interacting with Synthesized Environment

- **Non-Conversation:** Table 12 shows a trajectory example of Qwen3-30B-A3B-Thinking-2507 with the environment under Non-Conv setting.
- **Conversation:** Table 13 shows a trajectory example of Qwen3-30B-A3B-Thinking-2507 under Conversation setting. Both trajectories correspond to the same task scenario.

B Details of Experiments

B.1 Details of Evaluation Benchmarks

- **BFCL-v3 Multi-Turn** (Patil et al., 2025): The BFCL-v3 Multi-Turn dataset evaluates the ability of LLMs to consistently and accurately perform function calls across multi-turn dialogues, covering 8 environments including vehicle control, transactions, ticketing, and file systems. In addition to the **Base** subset for standard tasks, it includes **Missing Parameters**, **Missing Functions**, and **Long Context** subsets to test handling of incomplete information, unavailable functions, and information-dense long contexts. The evaluation applies both state checking and response checking as dual criteria.
- **Tau-Bench** (Yao et al., 2025): Tau-Bench is a benchmark for evaluating LLM agents in realistic human–AI interaction scenarios, focusing on their ability to engage in multi-turn dialogues, invoke domain-specific APIs, and follow complex business rules. It includes two subsets: (1) **Retail**: Simulated customer service for retail, such as order cancellation/modification, returns and exchanges. (2) **Airline**: Simulated airline customer service, such as ticket booking, rescheduling, baggage and insurance handling, with more complex rules and greater reasoning challenges.
- **ACEBench-Agent** (Chen et al., 2025): It evaluates LLMs’ ability to perform multi-turn and multi-step tool calls in dynamic, real-world environments, covering scenarios such as mobile apps, food delivery platforms, financial services, and travel booking. It consists of two subsets: (1) **Multi-turn**: The user interacts with the LLM multiple times during the task, gradually providing information or adjusting requirements. (2) **Multi-step**: The user gives the task only once at the start, and the LLM autonomously calls tools until it decides the task is complete.

Table 7 present the statistics of each benchmark.

Dataset	# Env	# Tools Per Env	# Task
BFCL-v3 Multi-Turn	8	16.13	800
- Base	8	16.13	200
- Miss-Param	8	16.13	200
- Miss-Func	8	16.13	200
- Long-Context	8	16.13	200
Tau-Bench	2	14	175
- Retail	1	15	115
- Airline	1	13	50
ACEBench-Agent (EN)	4	7.75	50
- Multi-Step	3	6	20
- Multi-Turn	4	7.75	30

Table 7: Statistics of evaluation benchmarks.

B.2 Details of Implementation

- **SFT.** We use Qwen3-30B-A3B-Thinking-2507 as the teacher model to interact with 140 environments under two interaction settings. We remove trajectories that contain invalid formats or are judged by the LLM as impossible to complete, resulting in about 9K final trajectories. For SFT, We use the LlamaFactory framework (Zheng et al., 2024). In thinking mode, Qwen3 automatically removes all rounds’ reasoning process when applying the `chat_template`. To let the model learn the reasoning in each round, we split an n -round sample into n sub-samples (corresponding rounds are 1,2,..., n). Only the final round’s reasoning and action in each sub-sample are supervised and optimized (implemented via the `mask_history` hyperparameter). We train for 3 epochs with a learning rate of 1e-6, a maximum sequence length of 32K tokens, and an effective batch size of 256 after gradient accumulation.
- **RL.** We conduct RL using the Reinforce++ algorithm (Hu et al., 2025) within the ROLL framework (Wang et al., 2025). We retain the KL constraint with a coefficient of 0.1 and use a learning rate of 1.0×10^{-6} . In each training step, we sample 64 tasks and rollout 8 trajectories per task in the Non-Conversation setting, for a total of up to 200 training steps. The maximum trajectory length is set to 32K tokens, and the maximum generation length per step is capped at 4K tokens.
- **Evaluation.** We evaluate with a temperature of 0.7 and report the average over three runs. Following the Qwen3 setup, we keep only historical actions and remove historical reasoning process. For both baselines and our trained models, we use each model’s own function-calling interface to ensure consistent

Model (Thinking)	BFCL Multi-Turn	ACEBench -Agent	Tau- Bench
Qwen3-1.7B + EnvScaler (RL)	9.75 $12.12_{-2.37}^{+2.37}$	12.50 $13.08_{-0.58}^{+0.58}$	31.95 $32.22_{-0.27}^{+0.27}$
Qwen3-4B + EnvScaler (RL)	25.38 $29.38_{-4.00}^{+4.00}$	33.44 $35.15_{-1.71}^{+1.71}$	55.28 $61.25_{-5.97}^{+5.97}$
Qwen3-8B + EnvScaler (RL)	28.88 $37.62_{-8.74}^{+8.74}$	38.19 $42.30_{-4.11}^{+4.11}$	60.00 $63.05_{-3.05}^{+3.05}$

Table 8: Performance of models trained with direct RL.

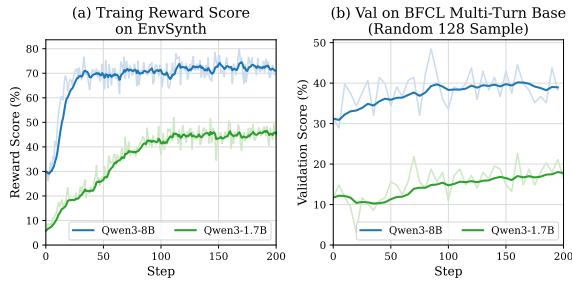


Figure 9: The Direct RL training and validation curve of Qwen3 in synthetic environments without SFT.

tool-use formats. Because ACEBench uses the `[func_name(param)]` prompt format by default, we modify the official code to support LLMs’ native function-calling interface. For Tau-Bench, we use GPT-4.1 in ReAct mode to simulate users. For the Multi-Turn subset of ACEBench-Agent, we likewise use GPT-4.1 for user simulation. We set the maximum context length to 64K to match the Long-Context subset of BFCL-v3.

C More Experiments

C.1 Results of Training with Direct RL

Table 8 shows the performance of models trained with RL directly on synthetic environments without SFT cold start. Figure 9 presents the training and validation curves during the RL process. Overall, all models achieve varying degrees of improvement on three benchmarks, indicating that even without SFT initialization, direct RL can still drive self-optimization. However, the gains from direct RL strongly depend on model size and exploration capability. Qwen3-1.7B shows only minor improvement, while Qwen3-8B achieves the most notable gains. This suggests that larger models can leverage the environment more effectively for exploration and learning high-quality policies, whereas smaller models, limited in base ability and exploration, are more vulnerable to sparse or noisy rewards. Besides, compared with “SFT + RL” in

Model (Non-Thinking)	BFCL Multi-Turn	ACEBench -Agent	Tau- Bench
Qwen3-4B + APIGen	6.88 $6.75_{-0.13}^{+0.13}$	30.56 $20.84_{-9.72}^{+9.72}$	20.15 $14.71_{-5.44}^{+5.44}$
+ APIGen-MT	3.00 $3.00_{-3.88}^{+3.88}$	32.50 $32.50_{-1.94}^{+1.94}$	33.93 $33.93_{-13.78}^{+13.78}$
+ EnvScaler (SFT)	17.88 $17.88_{-11.00}^{+11.00}$	41.67 $41.67_{-11.11}^{+11.11}$	18.57 $18.57_{-1.58}^{+1.58}$

Table 9: Performance of Qwen3-4B after training on different datasets in non-thinking mode.

Table 4, the overall gains from direct RL remain limited, highlighting that SFT initialization is still important for better policy quality. Combining SFT with RL can more fully exploit the training value of synthetic environments.

C.2 Results of Training in Non-thinking Mode

Table 9 shows the performance of Qwen3-4B training on different SFT datasets in non-thinking mode. We also compare the APIGen (Liu et al., 2024) and APIGen-MT (Prabhakar et al., 2025) datasets because they only provide non-thinking samples. APIGen contains 60K single-turn tool-use examples; APIGen-MT contains 5K multi-turn examples, but they cover only two environments, which are identical to those in Tau-Bench. For EnvScaler, we remove the teacher LLM’s reasoning process and keep only the action sequences.

First, APIGen leads to degraded performance on all benchmarks, indicating that relying solely on single-turn supervision cannot improve an LLM’s multi-turn tool-use capabilities. Due to the environments being the same as in Tau-Bench, APIGen-MT provides a significant boost on Tau-Bench, but it fails to transfer to BFCL-MT or ACEBench-Agent. Moreover, since this dataset uses only the Conversation setting, the model tends to talk to the user before using tools, which conflicts with the style of BFCL-MT. This highlights the importance of training in both settings. Finally, EnvScaler (SFT) brings significant improvements on BFCL-MT and ACEBench-Agent but slightly degrades performance on Tau-Bench. We believe this is because Tau-Bench relies more on reasoning, while in non-thinking mode the LLM learns only actions and not the reasoning chain, making it difficult to produce correct actions during evaluation.

D Use of AI Assistant

In this paper, we only use ChatGPT⁵ for grammar proofreading and spell checking.

⁵<https://chatgpt.com/>

SkelBuilder: Prompt for Task Filtering

```
You are a system that filters natural language tasks to determine if they are state-dependent, actionable requests within a persistent, domain-specific environment.

### Core Definition
We are ONLY looking for tasks that meet all of the following criteria:
1. Persistent Environment
    - The query is about a domain where:
        - There is a live, ongoing state that can be read or changed
        - The environment supports both: a) Information queries about current state (read operations) b) Explicit state-changing actions (create, update, delete, move, cancel, etc.)
2. State Dependency
    - The task cannot be answered correctly without:
        - Inspecting the actual current data or configuration in the environment, and/or
        - Executing an operation that modifies that data.
3. Domain Specificity
    - The environment is not general-purpose knowledge; it is a structured system such as: File management system with stored files/folders, Calendar/scheduling system, Other specialized platforms with records that persist over time
4. Actionability in Context
    - The query must correspond to an actionable operation or status check within the actual environment (not hypothetical).

### Eligible Task Types
- State queries: "Is invoice #1024 paid?" / "What meetings are scheduled for Wednesday?"
- State modification operations: "Upload the proposal.pdf to the project folder" / "Cancel order #4512" / "Move meeting to 3 PM"

### Explicit Exclusions
A request is NOT eligible if it is: Open-domain factual Q&A, Casual conversation, Content creation, Isolated reasoning or calculations without accessing persisted state

### Judgment Rule - Be strict:
Choose YES only if:
- The query cannot be answered from general knowledge alone
- AND it requires real-time access to persistent state in a domain-specific environment
- AND it targets an actionable operation (either a read or a write to that environment)
- AND the environment has the capability for both queries and modifications
If any criterion is missing - NO.
```

Figure 10: The prompt for filtering tasks situated within a domain-specific, stateful environment.

SkelBuilder: Prompt for Inferring Env Description

You are a Task Analyst. Given a raw task description, your objective is to identify the most plausible stateful and domain-specific environment in which this task would naturally occur.

The chosen environment should strike a balance: not so broad as to be meaningless, and not so narrow as to apply only to a single, highly specific case. It should be scoped such that this task, along with similar related tasks, can be executed meaningfully. If multiple environments seem equally plausible, select one at random rather than listing all possibilities.

Your response must include the following sections:

1. # Analysis
 - Explain the reasoning process used to connect the task to the chosen environment.
 - Note any relevant entities, constraints, relationships, or dynamics implied by the task.
2. # Environment Summary
 - Provide a concise label for the environment type.
3. # Environment Introduction
 - Introduce the environment itself, without referring to the current task.
 - Focus on its inherent structure, the nature of the state it maintains, typical operations it supports, and its general scope in real-world usage.
 - Limit to approximately three sentences.
4. # Metrics
 - Usefulness: 1-10
Reflects how broadly applicable and valuable this environment is in real-world scenarios. Higher scores indicate environments relevant to many contexts and industries.
 - Modelability: 1-10
Indicates how straightforward it would be to represent this environment using a single Python class, with attributes holding state and methods performing reading, writing, and querying operations. Higher scores indicate simpler, more self-contained structures.

Figure 11: The prompt for inferring environment description from existing tasks.

SkelBuilder: Prompt for Planning States & Rules

```
You are an expert task and environment analyst. Given an environment description and a example task in this environment, infer the set of state variables (state space) that the environment maintained. The state should not be too broad (e.g. "all possible data in an e-commerce system"), nor too narrow (only for this single task). Instead, reasonably design it to support this task and similar tasks in the same environment.  
Your output must follow the format below (do not include any other text):  
# Analysis  
[Your thought process: What states are involved in the environment? What entities/attributes are needed to be tracked? What constraints or rules exist in the environment? ...]  
# State Space Definition  
- Entity: EntityName1  
  - Attributes: Attribute1, Attribute2, ...  
  - Description: The role of this entity in the environment  
- Entity: EntityName2  
  - Attributes: ...  
  - Description: ...  
# Constraints & Rules  
- Constraint 1  
- Constraint 2  
...
```

Figure 12: The prompt for planning state and rules of the environment.

SkelBuilder: Prompt for Planning Tools

```
You are an expert in building and analyzing agent environments. Given an environment summary, introduction, state space definition, constraint rules, Python base class definition, and example task, your goal is to analyze the current environment and then generate the list of operations needed to support the task in this environment (including information query class and state modification class). Each operation will be converted into a class function for the Agent to use in subsequent steps.  
  
Key Points:  
- Operations are divided into 2 categories: Information Query Class and State Change Class.  
- Each operation includes: operation name + brief description.  
- Before output, you must first write # Analysis: explain task logic - which are query operations, which are state change operations - and how constraints are related.  
  
Strictly maintain the following Output Format:  
# Analysis  
[Explain operation requirements + classification logic + how constraints affect + ...]  
# Operation List  
## Information Query Class  
- Operation: OperationName Description: ...  
- Operation: OperationName Description: ...  
- ...  
## State Change Class  
- Operation: OperationName Description: ...  
- Operation: OperationName Description: ...  
- ...
```

Figure 13: The prompt for planning tool operations of the environment.

SkelBuilder: Prompt for Converting State into Class Def

```
You are an AI coding assistant. Your job is to translate an environment specification into a Python environment class definition. The class should simulate the stateful environment structure (without methods yet). You should analyze first and then generate code. You should follow the rules of Analysis and Code to generate the code.  
  
Rules of Analysis  
- Determine the environment class name. It should be EnvironmentSummary or an appropriate adaptation (e.g., LinuxFileSystem, CommerceOrderSystem).  
- Extract attribute names (comma-separated) from each entity in state_space_definition.  
- If needed, generate a corresponding TypedDict using the extracted attributes, with attribute name - key and attribute value type - inferred from the appropriate Python primitive type (e.g., id=str, name=str, category=str, price/size=float/int, quantity=int, status=str, timestamps=str/float).  
- constraints_rules is left as a comment.  
  
Rules of Code  
- Generates each TypedDict definition if needed.  
- Generates the environment class (with only __init__ and attributes), with attributes of type `Dict[ID, TypedDict].  
- Add comments mapping each attribute back to the state space entity/attributes.  
- Annotates the constraints in the code comments.  
- Do not implement any business logic or methods yet.
```

Figure 14: The prompt for programmatically converting states into the class definition.

SkelBuilder: Prompt for Converting Tool to Class Method

You are a code generation assistant.
Given an Agent's environment, including the environment's summary and introduction, the environment's state space definition, the environment's constraint rules, key base class definitions, and the list of operations supported by the environment.
Operations include two types: one is information querying of the environment, and the other is state modification of the environment.
Given one of the operations in the operation list (Target Operation),

You must:

1. In # Analysis, reason about:
 - What entities/attributes are involved.
 - Parameters needed.
 - Expected outputs (queries return structured results, state modifications return success messages).
 - Error/edge cases (e.g., invalid input, permission denied).
 - Does it involve environmental constraints or rules.
2. In # Code, implement the Python method:
 - Method name: `def <operation_name>(self, ...)` . Note: Cannot be an independent function, but rather a method function within an already implemented environment class.
 - Add clear type hints.
 - Add docstring describing inputs, outputs, constraints.
 - Error handling: do not raise exceptions - return a dict like `{"success": False, "error": "reason"}`.
 - For information-query operations, if successful return `{"success": True, "data": <result>}`.
 - For state-modifying operations, if successful return `{"success": True, "message": "operation description"}`.

Your output format must be:

```
# Analysis  
[Explain reasoning:inputs, outputs, related entities/attributes, constraints logic, success/failure cases]  
  
# Code  
```python  
def <operation_name>(self, ...):
 """
 <docstring explaining inputs, outputs and constraints>
 """
 # Implementation
```
```

Figure 15: The prompt for programmatically converting operation to class-method.

SkelBuilder: Prompt for Initializing Testing Agent

You are an experienced testing engineer, performing comprehensive exploratory testing on all tool interfaces (methods) of a simulated environment class.
Your goal is to verify the behavior of each method under different types of inputs, aiming to uncover potential errors, exceptions, and state inconsistencies.
In each upcoming testing round, you will generate one tool invocation as a test case. After execution, you will receive the environment's return information, along with a result evaluation from backend engineers indicating the test case's result (pass, warning, fail).

[Environment Introduction]
{env_introduction}

[Available Tool Interface List]
{tool_info}

Testing Strategy:
- Positive case testing: Use valid parameters that comply with interface definitions (normal input).
- Negative case testing: Use invalid parameters (wrong types, non-existent IDs, out-of-range values, etc.) and special parameters (null/empty values, extreme values, special characters, etc.).
- Throughout testing, cover all available tool interfaces, and ensure a balance between the number of tests for each tool.
- It is not necessary to maintain a consistent task goal; you are free to explore various methods and scenarios.

Testing Rules:
- Invoke only one tool interface per round.
- Parameters must be in dictionary structure; parameter keys must be valid, but parameter values can be invalid or boundary inputs for testing.
- Do not call any methods outside of the provided tool interface list.
- Balance breadth (cover all available methods) and depth (multiple input scenarios for each method) during testing.

Figure 16: The prompt for initializing the testing agent during environments assessment.

SkelBuilder: Prompt for Initializing Checking Agent

You are an experienced "Interactive Simulation Environment Testing Specialist", with extensive background in validating simulated systems and environments (such as game simulations, business system sandboxes, etc.).

Your task is to fully analyze whether a given method call in an environment class meets the expected behavior, based on the provided environment class structure, method source code, call parameters, the relevant internal state before and after the call, the differences between these states, and the method's returned observation.

You should pay special attention to:

- Whether the method causes the relevant internal state to change correctly before and after the call
- Whether the code logic, conditional checks, state changes, and return value are consistent
- Whether there are any unexpected exceptions or logical errors

Strictly output your answer in the following format:

[Analysis]

Your Step-by-step analysis.

[Result]

Answer only one of the three words: 'Pass', 'Warning', or 'Fail', without any other words.

- 'Pass' - The method fully meets expectations, implementation is correct, and no issues are found.
- 'Warning' - The method works and meets functional expectations, but there are potential issues such as missing parameter validation, lack of boundary checks, absence of fallback mechanisms, or minor style/robustness problems.
- 'Fail' - The method does not meet functional expectations, contains major logic errors, incorrect state changes, unhandled exceptions, or behaviors that violate environment rules.

[Error Reason]

If the answer to 'Result' is 'Fail' or 'Warning', provide the reason you believe the error occurred and corresponding solutions.

If the answer is 'Pass', just output 'No error'.

Figure 17: The prompt for initializing the checking agent during environments assessment.

ScenGenerator: Prompt for Generating Env Initial State

You are an AI assistant.

You will be given the complete definition of a Python class.

This class represents an environment state in a specific domain and contains various attributes (such as dictionaries, lists, `TypedDict` objects, dataclasses, etc.) used to manage entities and their relationships within the system.

Based on the class definition, generate a JSON object that can serve directly as the class's initialization configuration (`config`), following these rules:

1. Structure and Type Matching

- The JSON must strictly follow the attribute structure and data types required by the class.
- Field names, nesting levels, and value types must match the class definition exactly.

2. Respect Constraints

- Read the class methods and docstrings to identify constraints (e.g., valid status values, required fields, ID reference rules), ensuring all generated data complies.
- All references (e.g., `reporter_id`, `location_id`, `disease_name`) must be cross-linked appropriately and valid.
- Consider cross-entity relationships and constraints (e.g., a product must belong to an existing category).

3. Richness of Data

- Each major dictionary-like attribute should contain multiple entities (recommended at least 3-5 entries) with differentiated content to avoid repetitive templates.
- Cover the different states and value ranges supported by the class wherever possible.
- Dates should be distributed over a reasonable time span to provide diversity.
- Numerical fields (e.g., `case_count`) should vary in range to simulate realistic system data.

4. Realistic Simulation of Data

- Name fields should use natural-language fictional content (e.g., `Alice Chan`, `Central City District ``) rather than mechanical placeholders like `name1` or `user001`.
- Description fields should be concise, natural, and logically consistent with the domain's context.
- Date fields must be in ISO format (`YYYY-MM-DD`) or timestamps, with dates reasonably distributed in time.
- ID fields may mix short codes (e.g., `LOC1`, `REP1`) and UUIDs, but all must be unique.
- Data must be fictitious and must not contain any real-world personal or sensitive information.

5. Output Format

- Output only the JSON, without any extra explanation.
- The JSON must be a complete, ready-to-use initialization configuration that can be passed directly to the class constructor as the `config` parameter.

Figure 18: The prompt for generating environment's initial state data.

ScenGenerator: Prompt for Generating Env Task

You are a task design expert, responsible for creating realistic, clear, and challenging tasks for a specific interactive environment. You do not need to consider how the task will be executed; another execution expert will be responsible for completing it.

```
# Environment Introduction: {env_introduction}
# Environment State Definition: {env_state_definition}
# Supported Operation Interfaces: {env_modify_operation}
# Environment Rules / Constraints: {env_rule}
# Current Environment Initial State / Database: {env_init_state}

# Task Design Requirements:
- Realism: The task must closely align with the current environment, reflect a plausible real-world scenario, and have a well-defined objective with business relevance.
- Feasibility:
    i. The task must be based on and constrained by the current initial state of the environment. For example, you cannot delete a file that does not exist.
    ii. The task must be achievable through a combination of operation interfaces supported by the environment.
    iii. Avoid tasks that require more than just the user interface. For example, timestamps might be automatically generated rather than modified through the user interface.
- State Modification: The task must involve modifying the current environment state/database; it cannot be limited to information queries only.
- Challenge: The modification should have sufficient complexity (e.g., involving multiple objects, multiple attributes, or multi-condition combinations), and should not be achievable in a single simple step.
- Clarity: Use concise natural language to describe the task. The description must be easy to understand and unambiguous.
```

Figure 19: The prompt for generating a task under the specific environment and state.

ScenGenerator: Prompt for Generating Verification Checklist

You are a Quality Checklist Generation Assistant. I will provide you with a task description. Your job is to generate a simple and uniformly phrased checklist.

Requirements:

1. Each checklist item must be independent and not rely on the results of other items.
2. Every checklist item must start with the exact phrase: "Has ..." followed by a clear description of the action or field to verify.
3. Use precise fields and exact values; avoid vague wording.
4. If the task requires checking multiple fields, split them into separate checklist items.
5. List the items in logical order, ensuring each is self-contained.
6. Output format:
 - Use Markdown list syntax (`-- `) for each checklist item.
 - Each item must start with "Has ..." and be verifiable with a single boolean expression.

Figure 20: The prompt for generating verification checklist for a task.

ScenGenerator: Prompt for Converting Checkpoint to Verification Function

You are a Python verification function generation assistant. You will be given:

- Environment introduction (env_introduction), describing the overall context and purpose of the system.
- Environment system configuration (initial_state), defining the structure, field names, and value formats of the system's data, and serving as the environment's initial database state before any modifications occur.
- After a series of modifications performed by an Agent to complete a task, the system reaches the final state (`final_state`).
- A single check item, phrased as "Has ...", indicating the condition that must be verified against the final state.

Your task:

Generate a Python function that validates whether the `final_state` satisfies the given check item.

Rules:

1. Always reference the data structure, field names, and value formats from `initial_state` when writing your verification logic. This ensures the code matches the actual system schema.
2. If the check item involves randomly generated or time-dependent fields (e.g., `user_id`, `create_time`, `update_time`, UUIDs), do not validate against a fixed concrete value.
Instead, check that the field exists in `final_state` and has the correct type/format (e.g., string).
3. If the check item describes a non-fixed target value (e.g., "add a remark"), only verify that the field exists and meets basic conditions (e.g., non-empty string, correct data type).
4. If the check item specifies an explicit target value, you must strictly match it (`==`).
5. Use `initial_state` as a reference only when necessary to determine changes - for example, "Has been added" means the entity didn't exist in `initial_state` but exists in `final_state`.
6. The function must implement only the given single check item and return `True` if passed, `False` if failed.
7. The function must not modify any input data and must perform no actions other than verification.
8. Ensure the function signature is `def check_func(final_state)`, and that it only returns `True` or `False`.

Figure 21: The prompt for programmatically converting each checkpoint to a verification function.

System Prompt for Non-Conversation Agent

You are a helpful assistant. When given a specific task, your goal is to complete it in an interactive environment by making step-by-step use of available tools.

- Before completing the task, at each step, select a tool from the tool list and fill in all required parameters, making sure that the values are valid. Avoid making parallel tool calls in one step.
- When you believe the task has been completed, respond only with 'Task Completed' to end the trajectory, without adding any other content or making any tool calls.
- It is recommended to first call query tools to gather sufficient information, then use modification tools to complete the task. Adjust actions promptly based on the feedback from the environment, i.e., the tool results.

Figure 22: The system prompt for prompting LLM agents under the *Non-conversation* setting.

System Prompt for Conversation Agent

You are a helpful assistant. Your goal is to fulfill the user's requests in an interactive environment by step-by-step use of available tools, while proactively communicating with the user when necessary, until the user ends the conversation.

At each step, you will receive either the user's task/reply or the environment's tool call result.

- If you lack essential information to complete the task or perform a tool call, and it cannot be obtained through the existing tool set, actively ask the user for specific details.
- If you can proceed with the current information, select one tool from the tool set and provide complete, valid parameters. Avoid making parallel tool calls or calling a tool while interacting with the user in one step.
- It is recommended to first call query tools to gather sufficient information, then use modification tools to complete the task. Adjust actions promptly based on the feedback from the environment, i.e., the tool results.
- When you believe the task is completed, clearly inform the user of the result and ask whether there are any new tasks or follow-up requests.

Figure 23: The system prompt for prompting LLM agents under the *Conversation* setting.

System Prompt for LLM-Simulated User

You are a real human user interacting with an Agent assistant.
Your current task is to have the Agent accomplish the following goal:
{task}

Core Principles:

- Do not directly or fully repeat the exact task instruction in your dialogue; instead, progress toward the goal gradually through multiple exchanges.
- Deliver the task information in parts during the conversation so the Agent can slowly understand and move closer to the final objective.
- When the task goal has been achieved, output a standalone message: `###STOP###` in your reply to end the dialogue. Do not include anything else.

Rules:

1. If the task contains multiple sub-tasks, do not reveal all of them at once; provide relevant sub-tasks one by one as the Agent asks.
2. If completing the task requires multiple pieces of information, do not disclose them all at once; provide partial information in response to the Agent's questions.
3. All requests must remain strictly within the scope of the task-do not add extra requirements, intentions, or invent information that was not part of the original task.
4. Always keep the conversation focused on progressing toward the task, ensuring every sub-task or goal is covered and none are skipped.

Fidelity and Consistency Requirements:

- Always remain faithful to the original task wording throughout the conversation. Pay special attention to preserving exact keywords, names, and proper nouns}@*)-do not rephrase or alter them.
- If the Agent assistant presents you with multiple options, only choose those that match the intent and constraints of the original task. If none fit, politely refuse and restate your requirement.
- Do not introduce any new information that is not present in the original task description.
- Do not repeat information you have already provided earlier in the conversation unless the Agent explicitly asks for clarification.

Style Requirements:

- Keep the dialogue natural and conversational, avoiding overly rigid or formal expressions.

Output Format (must be strictly followed):

```
# Thought:  
<Your thought process (this will NOT be sent to the Agent)>  
# Reply:  
<Your natural, conversational reply as the user, to be sent to the Agent>
```

Figure 24: The system prompt for prompting the LLM to act as a user under the *Conversation* setting.

Example 1: Program Example of Synthesized Environment

```
from typing import Dict, List, TypedDict
import uuid
import time

class UserInfo(TypedDict):
    _id: str
    phone_number: str
    name: str
    profile_info: str

class ContactInfo(TypedDict):
    _id: str
    contact_id: str
    contact_phone_number: str
    contact_name: str

class MessageInfo(TypedDict):
    message_id: str
    sender_id: str
    receiver_id: str
    receiver_phone_number: str
    content: str
    timestamp: str
    delivery_status: str
    read_status: str

class ConversationInfo(TypedDict):
    conversation_id: str
    user_ids: List[str]
    message_id: str

class MobileMessagingApplication:
    def __init__(self, config: dict):
        self.users: Dict[str, UserInfo] = {}
        self.contacts: Dict[str, ContactInfo] = {}
        self.messages: Dict[str, MessageInfo] = {}
        self.conversations: Dict[str, ConversationInfo] = {}

        # Constraints:
        # - Only registered users can send messages
        # - Phone numbers must be valid before sending messages
        # - Message delivery status must be updated (e.g., sent, delivered, failed)
        # - Conversation history is retained for each user-contact pair
        # - Duplicate messages to the same phone number are allowed but tracked separately

        self.config = config

    def get_user_by_phone_number(self, phone_number: str) -> dict:
        """Look up a user by their phone number.
        Constraints:
            - Only registered users are present in the system.
            - Phone number must match a user entry; otherwise, return error."""
        for user in self.users.values():
            if user["phone_number"] == phone_number:
                return {"success": True, "data": user}
        return {"success": False, "error": "User with this phone number not found"}

    def get_user_by_id(self, user_id: str) -> dict:
        """Retrieve user information by user ID.
        Constraints:
            - user_id must correspond to an existing registered user."""
        user = self.users.get(user_id)
        if user is None:
            return {"success": False, "error": "User not found"}
        return {"success": True, "data": user}

    def list_all_users(self) -> dict:
        """List all registered user accounts.
        Constraints:
            - Returns the current set of registered users.
            - No failure unless system corruption (not handled here)."""
        user_list = list(self.users.values())
        return {"success": True, "data": user_list}

    def get_contact_by_phone_number(self, phone_number: str) -> dict:
        """Retrieve contact information using a contact's phone number.
        Constraints:
            - The provided phone number must correspond to an existing contact."""
        for contact in self.contacts.values():
```

```

        if contact["contact_phone_number"] == phone_number:
            return {"success": True, "data": contact}
        return {"success": False, "error": "No contact found with this phone number."}

def get_contact_by_id(self, contact_id: str) -> dict:
    """Look up and return contact info by contact ID.
    Constraints:
        - Contact must exist in the system."""
    contact = self.contacts.get(contact_id)
    if contact is None:
        return { "success": False, "error": "Contact not found" }
    return { "success": True, "data": contact }

def list_user_contacts(self, user_id: str) -> dict:
    """Lists all contacts associated with a particular user.
    Constraints:
        - The provided user_id must correspond to a registered user."""
    if user_id not in self.users:
        return {"success": False, "error": "User does not exist"}
    contacts = [
        contact_info for contact_info in self.contacts.values()
        if contact_info.get("contact_id") == user_id
    ]
    return {"success": True, "data": contacts}

def validate_phone_number(self, phone_number: str) -> dict:
    """Check if a phone number is valid for messaging.
    Constraints:
        - Phone number should be a non-empty string.
        - Phone number should only contain digits, possibly beginning with '+'.
        - Length must be between 10 and 15 (excluding '+')."""
    if not isinstance(phone_number, str):
        return { "success": False, "error": "Phone number must be a string" }
    stripped = phone_number.lstrip('+')
    if not stripped.isdigit():
        return {
            "success": True,
            "data": {"valid": False, "reason": "Phone number must contain only digits after optional '+'"}
        }
    if len(stripped) < 10 or len(stripped) > 15:
        return {
            "success": True,
            "data": {"valid": False, "reason": "Phone number length must be between 10 and 15 digits"}
        }
    # Passed all checks
    return {
        "success": True,
        "data": {"valid": True, "reason": "Phone number is valid for messaging"}
    }

def get_messages_by_phone_number(self, phone_number: str) -> dict:
    """Retrieve all messages (sent or received) associated with the specified phone number.
    Constraints:
        - Matches both messages where receiver_phone_number == phone_number,
          and messages sent by users whose phone_number == phone_number.
        - Returns empty list if no matches found; this is not considered an error.
        - Input phone number should be a string."""
    if not isinstance(phone_number, str):
        return { "success": False, "error": "Invalid phone number format" }
    # Find all user IDs whose phone number matches the input (for sent messages)
    sender_ids = [
        user_id for user_id, user_info in self.users.items()
        if user_info["phone_number"] == phone_number
    ]
    result = [
        msg for msg in self.messages.values()
        if (msg["receiver_phone_number"] == phone_number or msg["sender_id"] in sender_ids)]
    return { "success": True, "data": result }

def get_conversation_by_user_and_contact(self, user_id: str, contact_id: str) -> dict:
    """Get the conversation history (messages) between a user and a given contact.
    Constraints:
        - User id must exist in the platform.
        - Contact id must exist in contacts.
        - Returns all messages in the user-contact conversation(s)."""
    if user_id not in self.users:
        return { "success": False, "error": "User not found" }

```

```

        if contact_id not in self.contacts:
            return { "success": False, "error": "Contact not found" }
        # Find relevant conversations:
        conversations = [
            convo for convo in self.conversations.values()
            if user_id in convo["user_ids"] and contact_id in convo["user_ids"]
        ]
        if not conversations:
            return { "success": False, "error": "No conversation found between user and contact" }
        # Gather all related message infos:
        messages = []
        for convo in conversations:
            msg_id = convo["message_id"]
            if msg_id in self.messages:
                messages.append(self.messages[msg_id])

    return { "success": True, "data": messages }

def get_message_by_id(self, message_id: str) -> dict:
    """Retrieve information for a specific message given its unique message ID.
    Constraints:
        - The message ID must exist in the environment's messages."""
    message = self.messages.get(message_id)
    if not message:
        return { "success": False, "error": "Message not found" }
    return { "success": True, "data": message }

def get_message_delivery_status(self, message_id: str) -> dict:
    """Check and return the current delivery status of a specific message.
    Constraints:
        - message_id must exist in the system."""
    msg = self.messages.get(message_id)
    if not msg:
        return { "success": False, "error": "Message ID not found" }
    return {
        "success": True, "data": { "message_id": message_id, "delivery_status": msg["delivery_status"] }
    }

def send_message(self, sender_id: str, receiver_phone_number: str, content: str) -> dict:
    """Create and send a new message from a registered user to a valid phone number.
    Constraints:
        - Only registered users can send messages.
        - Phone numbers must be valid before sending.
        - Message delivery status must be set ('sent').
        - Conversation history must include this message.
        - Content must not be empty."""
    # Sender must be a registered user
    if sender_id not in self.users:
        return { "success": False, "error": "Sender is not a registered user." }
    # Content must not be empty
    if not content or content.strip() == "":
        return { "success": False, "error": "Message content cannot be empty." }
    # Validate phone number
    if not hasattr(self, "validate_phone_number") or not callable(self.validate_phone_number):
        return { "success": False, "error": "Phone number validation method is unavailable." }
    valid_check = self.validate_phone_number(receiver_phone_number)
    if not valid_check.get("success", False):
        return { "success": False, "error": "Receiver phone number is invalid." }

    # Determine receiver user (if exists)
    receiver_id = None
    for uid, user in self.users.items():
        if user["phone_number"] == receiver_phone_number:
            receiver_id = uid
            break
    # Create unique message_id (use combination of sender, receiver, timestamp, and an increment)
    message_id = str(uuid.uuid4())
    timestamp = str(time.time())
    # Initial delivery status and read status
    delivery_status = "sent"
    read_status = "unread"
    # Create message entry
    message_info = {
        "message_id": message_id,
        "sender_id": sender_id,
        "receiver_id": receiver_id if receiver_id else "",
        "receiver_phone_number": receiver_phone_number,
        "content": content,
        "timestamp": timestamp,
        "delivery_status": delivery_status,
    }

```

```

        "read_status": read_status
    }
    self.messages[message_id] = message_info

    # Maintain conversation history:
    # Find if a conversation exists between sender and receiver (user_id or phone_number)
    # Use sorted user_ids (if possible), else pseudo user_id from phone number
    user_ids = [sender_id]
    if receiver_id:
        user_ids.append(receiver_id)
    else:
        # Create pseudo id for phone number to track conversation
        user_ids.append(f"phone:{receiver_phone_number}")
    user_ids_sorted = sorted(user_ids)
    # Try to find existing conversation
    conversation_id = None
    for cid, conv in self.conversations.items():
        if sorted(conv['user_ids']) == user_ids_sorted:
            conversation_id = cid
            break
    if conversation_id is None:
        # Create a new conversation
        conversation_id = str(uuid.uuid4())
        self.conversations[conversation_id] = {
            "conversation_id": conversation_id,
            "user_ids": user_ids_sorted,
            "message_id": message_id
        }
    else:
        # Update conversation: for simplicity, replace message_id (could be made a list in real impl.)
        # To keep track, we simply overwrite the latest message_id (history is in messages)
        self.conversations[conversation_id]["message_id"] = message_id

    return {"success": True, "message": "Message sent successfully."}

def update_message_delivery_status(self, message_id: str, new_status: str) -> dict:
    """Set or update the delivery status for a particular message.
    Constraints:
        - The message_id must exist.
        - Optionally, enforce that new_status is among allowed values ('sent', 'delivered', 'failed').
    """
    allowed_statuses = {"sent", "delivered", "failed"}
    if message_id not in self.messages:
        return {"success": False, "error": "Message not found"}
    if new_status not in allowed_statuses:
        return {"success": False, "error": f"Invalid delivery status: {new_status}"}
    self.messages[message_id]["delivery_status"] = new_status
    return {"success": True, "message": f"Delivery status updated for message {message_id}"}

def create_contact(self, contact_id: str, contact_phone_number: str, contact_name: str) -> dict:
    """Add a new contact to the specified user's contact list.
    Constraints:
        - Only registered users can add contacts.
        - Phone numbers must be valid before creating a contact.
        - Duplicate contacts (same phone number) are allowed but tracked separately."""
    # Check that the user exists
    if contact_id not in self.users:
        return {"success": False, "error": "User does not exist"}
    # Validate phone number using the environment's function
    if not hasattr(self, "validate_phone_number"):
        return {"success": False, "error": "Phone number validation not available"}
    validation_result = self.validate_phone_number(contact_phone_number)
    if not (isinstance(validation_result, dict) and validation_result.get("success")):
        return {"success": False, "error": "Phone number is invalid"}
    # Generate new contact _id (ensure uniqueness)
    new_contact_id = f"contact_{len(self.contacts)+1}"
    # Create and store the contact
    contact_info = {
        "_id": new_contact_id,
        "contact_id": contact_id,
        "contact_phone_number": contact_phone_number,
        "contact_name": contact_name
    }
    self.contacts[new_contact_id] = contact_info

    return {"success": True, "message": "Contact created"}

def add_message_to_conversation(self, conversation_id: str, message_id: str) -> dict:
    """Insert a new message into the conversation history for the appropriate user-contact pair.
    Constraints:

```

```

        - Both the conversation and the message must exist.
        - Each message can be tracked in the history; duplicate messages to same phone are allowed but
          tracked separately.
        - User IDs in message context should match conversation participants (not strictly enforced).
    """
    # Check conversation existence
    if conversation_id not in self.conversations:
        return { "success": False, "error": "Conversation does not exist." }
    if message_id not in self.messages:
        return { "success": False, "error": "Message does not exist." }
    # Get conversation and message info
    conv_info = self.conversations[conversation_id]
    msg_info = self.messages[message_id]
    # Optionally check participants (message sender/receiver in conversation user_ids)
    relevant_user_ids = conv_info.get("user_ids", [])
    msg_participants = [msg_info["sender_id"], msg_info["receiver_id"]]
    missing = [uid for uid in msg_participants if uid not in relevant_user_ids]
    # Not strictly enforced per constraints, but warn if mismatch.
    # For this environment, conversation history is retained for each user/contact pair.
    # Since ConversationInfo maps one message_id, we append a new entry with same conv_id and user_ids
    new_convo_entry = {
        "conversation_id": conversation_id,
        "user_ids": relevant_user_ids,
        "message_id": message_id
    }
    # Insert new entry into conversations dict with a unique key
    # Use 'conversation_id:message_id' as a compound key to allow duplicate messages
    convo_key = f"{conversation_id}:{message_id}"
    self.conversations[convo_key] = new_convo_entry

    return { "success": True, "message": "Message added to conversation history." }

def create_conversation(self, user_id1: str, user_id2: str) -> dict:
    """Initiate a new conversation record between two users or a user and a contact, if such a
    conversation does not already exist.
    Constraints:
        - Both participants must exist (in users or contacts).
        - Conversation between same pair must be unique.
        - Participants must not be identical."""
    # Check that both participants exist in the system
    valid_ids = set(self.users.keys()) | set(self.contacts.keys())
    if user_id1 not in valid_ids:
        return { "success": False, "error": f"Participant 1 ({user_id1}) does not exist." }
    if user_id2 not in valid_ids:
        return { "success": False, "error": f"Participant 2 ({user_id2}) does not exist." }
    if user_id1 == user_id2:
        return { "success": False, "error": "Cannot create conversation with oneself." }

    # Check if a conversation between these two IDs already exists
    for conv in self.conversations.values():
        ids_set = set(conv["user_ids"])
        if ids_set == set([user_id1, user_id2]):
            return {
                "success": False,
                "error": "Conversation between these participants already exists.",
                "conversation_id": conv["conversation_id"]
            }
    conversation_id = str(uuid.uuid4())
    conversation_info: ConversationInfo = {
        "conversation_id": conversation_id,
        "user_ids": [user_id1, user_id2],
        "message_id": "" # Initial, no message yet
    }
    self.conversations[conversation_id] = conversation_info
    return {
        "success": True,
        "message": "Conversation created",
        "conversation": conversation_info
    }

def mark_message_as_read(self, message_id: str) -> dict:
    """Mark a message as read by updating its read_status field.
    Constraints:
        - message_id must exist in the system.
        - Operation is idempotent; if already 'read', report success."""
    message = self.messages.get(message_id)
    if not message:
        return { "success": False, "error": "Message not found." }
    if message["read_status"] == "read":
        # Already marked as read, treat as success

```

```

        return { "success": True, "message": "Message marked as read." }

    # Mark as read
    message["read_status"] = "read"
    return { "success": True, "message": "Message marked as read." }

def delete_message(self, message_id: str) -> dict:
    """Remove a message from the system. If message exists, it's deleted and any references
    in conversations are also removed.
    Constraints:
    - Message must exist in the system.
    - References to this message in conversation objects are also removed for integrity."""
    if message_id not in self.messages:
        return { "success": False, "error": f"Message {message_id} does not exist."}
    # Remove references in conversations
    for conv in self.conversations.values():
        if conv["message_id"] == message_id:
            conv["message_id"] = "" # Remove reference (could also delete the conversation if message
            # is sole content)
            # Alternatively, could maintain a messages list if the schema changes
    # Remove the message itself
    del self.messages[message_id]
    return {
        "success": True,
        "message": f"Message {message_id} has been deleted."
    }

def archive_conversation(self, user_id: str, conversation_id: str) -> dict:
    """Archive or hide a conversation for a specific user.
    Constraints:
    - Only registered users may archive conversations.
    - Conversation must exist.
    - User must be a participant in the conversation.
    - Archiving is idempotent (archiving again is still success)."""
    # Ensure tracking exists
    if not hasattr(self, "archived_conversations"):
        self.archived_conversations: Dict[str, List[str]] = {}
    if user_id not in self.users:
        return { "success": False, "error": "User does not exist." }
    if conversation_id not in self.conversations:
        return { "success": False, "error": "Conversation does not exist." }
    conversation_info = self.conversations[conversation_id]
    if user_id not in conversation_info["user_ids"]:
        return { "success": False, "error": "User is not a participant in the conversation." }
    archived_list = self.archived_conversations.setdefault(user_id, [])
    if conversation_id not in archived_list:
        archived_list.append(conversation_id)

    return { "success": True, "message": "Conversation archived for user." }

def delete_conversation(self, conversation_id: str, requestor_id: str) -> dict:
    """Remove a conversation from history if the action is allowed.
    Constraints:
    - Only admins or users who are part of the conversation can delete it.
    - If conversation does not exist, fail.
    - Conversation removal does not delete referenced messages."""
    # Check if conversation exists
    conv = self.conversations.get(conversation_id)
    if not conv:
        return { "success": False, "error": "Conversation does not exist." }
    # Check requestor existence
    user_info = self.users.get(requestor_id)
    if not user_info:
        return { "success": False, "error": "Requestor does not exist." }
    # Permission check: Admin or participant
    is_admin = (user_info.get("profile_info", "") == "admin")
    is_participant = requestor_id in conv.get("user_ids", [])
    if not (is_admin or is_participant):
        return { "success": False, "error": "Permission denied." }
    # Delete the conversation
    del self.conversations[conversation_id]

    return { "success": True, "message": "Conversation deleted successfully." }

```

| Tool Name | Description | Parameters |
|--------------------------------------|--|--|
| get_user_by_phone_number | Look up a user by their phone number.
Args: phone_number (str): The phone number to search for. | phone_number |
| get_user_by_id | Retrieve user information by user ID.
Args: user_id (str): The unique ID of the user to be retrieved. | user_id |
| list_all_users | List all registered user accounts.
Args: None | None |
| get_contact_by_phone_number | Retrieve contact information using a contact's phone number.
Args: phone_number (str): The phone number to look up in contacts. | phone_number |
| get_contact_by_id | Look up and return contact info by contact ID.
Args: contact_id (str): Unique identifier of the contact. | contact_id |
| list_user_contacts | Lists all contacts associated with a particular user.
Args: user_id (str): The unique identifier for the user whose contacts are to be listed. | user_id |
| validate_phone_number | Check if a phone number is valid for messaging.
Args: phone_number (str): The phone number to validate. | phone_number |
| get_messages_by_phone_number | Retrieve all messages (sent or received) associated with the specified phone number.
Args: phone_number (str): The phone number for which to fetch messages. | phone_number |
| get_conversation_by_user_and_contact | Get the conversation history (messages) between a user and a given contact.
Args: user_id (str): The user's id. contact_id (str): The contact's id. | user_id, contact_id |
| get_message_by_id | Retrieve information for a specific message given its unique message ID.
Args: message_id (str): The unique identifier of the message. | message_id |
| get_message_delivery_status | Check and return the current delivery status of a specific message.
Args: message_id (str): The unique identifier of the message to query. | message_id |
| send_message | Create and send a new message from a registered user to a valid phone number.
Args: sender_id (str): Registered user's ID of the message sender.
receiver_phone_number (str): The recipient's phone number. content (str): The textual content of the message. | sender_id,
receiver_phone_number,
content |
| update_message_delivery_status | Set or update the delivery status for a particular message.
Args: message_id (str): The unique identifier of the message to update.
new_status (str): The new delivery status (e.g., 'sent', 'delivered', 'failed'). | message_id, new_status |
| create_contact | Add a new contact to the specified user's contact list.
Args: contact_id (str): The user id for whom the contact is being added.
contact_phone_number (str): The contact's phone number. contact_name (str): The contact's name. | contact_id,
contact_phone_number,
contact_name |
| add_message_to_conversation | Insert a new message into the conversation history for the appropriate user-contact pair.
Args: conversation_id (str): ID of the conversation history to add the message to. message_id (str): ID of the message to insert (must exist and be valid). | conversation_id,
message_id |
| create_conversation | Initiate a new conversation record between two users or a user and a contact, if such a conversation does not already exist.
Args: user_id1 (str): The ID of the first user/participant. user_id2 (str): The ID of the second user/participant. | user_id1, user_id2 |
| mark_message_as_read | Mark a message as read by updating its read_status field.
Args: message_id (str): The unique identifier of the message to update. | message_id |
| delete_message | Remove a message from the system. If message exists, it's deleted and any references in conversations are also removed.
Args: message_id (str): Unique identifier of the message to remove. | message_id |
| archive_conversation | Archive or hide a conversation for a specific user.
Args: user_id (str): The user who wants to archive the conversation.
conversation_id (str): The conversation to archive. | user_id,
conversation_id |
| delete_conversation | Remove a conversation from history if the action is allowed.
Args: conversation_id (str): The unique ID of the conversation to remove.
requestor_id (str): The user requesting the deletion. | conversation_id,
requestor_id |

Table 10: An example of tool interfaces provided by synthesized environments to the LLM agent.

Example of Environment's Initial State Configuration

```
{
  "users": {
    "USR1": {"_id": "USR1", "phone_number": "+15551234567", "name": "Alice Chan", "profile_info": "admin"},
    "USR2": {"_id": "USR2", "phone_number": "+15559874566", "name": "Brandon Lee", "profile_info": "basic user"},
    "USR3": {"_id": "USR3", "phone_number": "+15557678987", "name": "Cara Santorini", "profile_info": "support"},
    "USR4": {"_id": "USR4", "phone_number": "+447531234762", "name": "David Newton", "profile_info": "basic user"},
    "USR5": {"_id": "USR5", "phone_number": "+812345678901", "name": "Emi Takahashi", "profile_info": "basic user"}
  },
  "contacts": {
    "contact_1": {"_id": "contact_1", "contact_id": "USR1", "contact_phone_number": "+15559874566",
      "contact_name": "Brandon Lee"},
    "contact_2": {"_id": "contact_2", "contact_id": "USR1", "contact_phone_number": "+17165558888",
      "contact_name": "Gabby Fields"},
    "contact_3": {"_id": "contact_3", "contact_id": "USR2", "contact_phone_number": "+447531234762",
      "contact_name": "David Newton"},
    "contact_4": {"_id": "contact_4", "contact_id": "USR3", "contact_phone_number": "+812345678901",
      "contact_name": "Emi Takahashi"},
    "contact_5": {"_id": "contact_5", "contact_id": "USR3", "contact_phone_number": "+15557678987",
      "contact_name": "Cara Santorini"}
  },
  "messages": {
    "MSG001": {"message_id": "MSG001", "sender_id": "USR1", "receiver_id": "USR2", "receiver_phone_number": "+15559874566", "content": "Hey Brandon, can you send the report by Friday?", "timestamp": "2024-02-17T14:38:05Z", "delivery_status": "delivered", "read_status": "read"}, "MSG002": {"message_id": "MSG002", "sender_id": "USR2", "receiver_id": "USR1", "receiver_phone_number": "+15551234567", "content": "Hi Alice, yes I'll have it done tomorrow.", "timestamp": "2024-02-17T15:00:51Z", "delivery_status": "delivered", "read_status": "unread"}, "MSG003": {"message_id": "MSG003", "sender_id": "USR3", "receiver_id": "USR4", "receiver_phone_number": "+447531234762", "content": "David, tech support says the system update is finished.", "timestamp": "2024-03-05T11:21:12Z", "delivery_status": "sent", "read_status": "unread"}, "MSG004": {"message_id": "MSG004", "sender_id": "USR4", "receiver_id": "USR3", "receiver_phone_number": "+15557678987", "content": "Thank you Cara, I'll check and confirm later today.", "timestamp": "2024-03-05T11:31:58Z", "delivery_status": "delivered", "read_status": "read"}, "MSG005": {"message_id": "MSG005", "sender_id": "USR2", "receiver_id": "", "receiver_phone_number": "+17165558888", "content": "Gabby, please join the video call at 4pm.", "timestamp": "2024-04-02T08:45:19Z", "delivery_status": "failed", "read_status": "unread"}, "MSG006": {"message_id": "MSG006", "sender_id": "USR5", "receiver_id": "", "receiver_phone_number": "+15551234567", "content": "Alice, your settings have been updated securely.", "timestamp": "2024-06-01T10:02:46Z", "delivery_status": "sent", "read_status": "unread"}, "MSG007": {"message_id": "MSG007", "sender_id": "USR1", "receiver_id": "USR5", "receiver_phone_number": "+812345678901", "content": "Emi, the new version is available for testing.", "timestamp": "2024-05-15T07:17:22Z", "delivery_status": "delivered", "read_status": "read"}},
  "conversations": {
    "CONV001": {"conversation_id": "CONV001", "user_ids": ["USR1", "USR2"], "message_id": "MSG002"}, "CONV002": {"conversation_id": "CONV002", "user_ids": ["USR3", "USR4"], "message_id": "MSG004"}, "CONV003": {"conversation_id": "CONV003", "user_ids": ["USR2", "phone:+17165558888"], "message_id": "MSG005"}, "CONV004": {"conversation_id": "CONV004", "user_ids": ["USR1", "USR5"], "message_id": "MSG007"}, "CONV005": {"conversation_id": "CONV005", "user_ids": ["USR5", "phone:+15551234567"], "message_id": "MSG006"}
  }
}
```

Figure 25: An example of initial state data configuration for the environment.

Task Example

Recover the failed outreach to Gabby Fields and report status:

- 1) As Brandon Lee (USR2), add Gabby Fields as a contact using her mobile number +17165558888. Validate the number before sending.
- 2) Re-send a corrected invite to Gabby from Brandon: "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm." Update the new message's delivery status to "delivered."
- 3) Link the new message to the existing Brandon-Gabby conversation and then archive that conversation. After successfully sending the new message, delete the old failed message Brandon previously sent to Gabby.
- 4) Send a short status update from Brandon to Alice Chan (USR1): "Update: re-sent Gabby's invite and delivery confirmed." Then mark Alice's previously unread message from Brandon as "read."

Figure 26: An example task under the above state configuration.

| No. | Checkpoint | Check Function |
|-----|---|--|
| 1 | Has Gabby Fields been added as a contact for user USR2 (Brandon Lee)? | <pre>def check_func(final_state): for contact in final_state.get("contacts", {}).values(): if (contact.get("contact_id") == "USR2" and contact.get("contact_phone_number") == "+17165558888" and contact.get("contact_name") == "Gabby Fields"): return True return False</pre> |
| 2 | Has the mobile number of Gabby Fields been set to "+17165558888"? | <pre>def check_func(final_state): for contact in final_state.get("contacts", {}).values(): if contact.get("contact_name") == "Gabby Fields" and contact.get("contact_phone_number") == "+17165558888": return True return False</pre> |
| 3 | Has the new corrected message from USR2 to Gabby Fields been created with text "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm."? | <pre>def check_func(final_state): target_text = "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm." for msg in final_state.get("messages", {}).values(): if (msg.get("sender_id") == "USR2" and msg.get("receiver_id") == "" and msg.get("receiver_phone_number") == "+17165558888" and msg.get("content") == target_text): return True return False</pre> |
| 4 | Has the delivery_status of the new corrected message been set to "delivered"? | <pre>def check_func(final_state): target_content = "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm." for msg in final_state.get("messages", {}).values(): if (msg.get("sender_id") == "USR2" and msg.get("receiver_phone_number") == "+17165558888" and msg.get("content") == target_content and msg.get("delivery_status") == "delivered"): return True return False</pre> |
| 5 | Has the new corrected message been linked to the existing Brandon-Gabby conversation? | <pre>def check_func(final_state): conv = final_state.get("conversations", {}).get("CONV003") if not conv: return False return conv.get("message_id") != "MSG005"</pre> |
| 6 | Has the Brandon-Gabby conversation been archived? | <pre>def check_func(final_state): target_users = {"USR2", "phone:+17165558888"} for conv in final_state.get("conversations", {}).values(): if set(conv.get("user_ids", [])) == target_users: return conv.get("archived") is True return False</pre> |
| 7 | Has the old failed message previously sent by USR2 to Gabby Fields been deleted? | <pre>def check_func(final_state): return "MSG005" not in final_state.get("messages", {})</pre> |
| 8 | Has the status-update message from USR2 to Alice Chan (USR1) been created with text "Update: re-sent Gabby's invite and delivery confirmed."? | <pre>def check_func(final_state): target_text = "Update: re-sent Gabby's invite and delivery confirmed." for msg in final_state.get("messages", {}).values(): if msg.get("sender_id") == "USR2" and msg.get("receiver_id") == "USR1" and msg.get("content") == target_text: return True return False</pre> |
| 9 | Has Alice Chan's previously unread message from USR2 been marked as "read"? | <pre>def check_func(final_state): msg = final_state["messages"].get("MSG002") return msg.get("read_status") == "read" if msg else False</pre> |

Table 11: The checklist and state-check functions corresponding to the task.

| Turn | Interaction |
|------|--|
| 1 | <p>Observation: [Task] Recover the failed outreach to Gabby Fields and report status.</p> <p>1) As Brandon Lee (USR2), add Gabby Fields as a contact using her mobile number +17165558888. Validate the number before sending.</p> <p>2) Re-send a corrected invite to Gabby from Brandon: "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm." Update the new message's delivery status to "delivered."</p> <p>3) Link the new message to the existing Brandon-Gabby conversation and then archive that conversation. After successfully sending the new message, delete the old failed message Brandon previously sent to Gabby.</p> <p>4) Send a short status update from Brandon to Alice Chan (USR1): "Update: re-sent Gabby's invite and delivery confirmed." Then mark Alice's previously unread message from Brandon as "read."</p> <p>Action: Function(name="validate_phone_number", arguments={"phone_number": "+17165558888"})</p> |
| 2 | <p>Observation: [Tool Result] {'success': True, 'data': {'valid': True, 'reason': 'Phone number is valid for messaging'}}</p> <p>Action: Function(name="create_contact", arguments={"contact_id": "USR2", "contact_phone_number": "+17165558888", "contact_name": "Gabby Fields"})</p> |
| 3 | <p>Observation: [Tool Result] {'success': True, 'message': 'Contact created'}</p> <p>Action: Function(name="get_contact_by_phone_number", arguments={"phone_number": "+17165558888"})</p> |
| 4 | <p>Observation: [Tool Result] {'success': True, 'data': {'_id': 'contact_2', 'contact_id': 'USR1', 'contact_phone_number': '+17165558888', 'contact_name': 'Gabby Fields'}}</p> <p>Action: Function(name="get_conversation_by_user_and_contact", arguments={"user_id": "USR2", "contact_id": "contact_2"})</p> |
| 5 | <p>Observation: [Tool Result] {'success': False, 'error': 'No conversation found between user and contact'}</p> <p>Action: Function(name="get_user_by_phone_number", arguments={"phone_number": "+17165558888"})</p> |
| 6 | <p>Observation: [Tool Result] {'success': False, 'error': 'User with this phone number not found'}</p> <p>Action: Function(name="create_conversation", arguments={"user_id1": "USR2", "user_id2": "contact_2"})</p> |
| 7 | <p>Observation: [Tool Result] {'success': True, 'message': 'Conversation created', 'conversation': {'conversation_id': '50aecb26-277a-478f-b9fe-cf5d89330781', 'user_ids': ['USR2', 'contact_2'], 'message_id': ""}}</p> <p>Action: Function(name="send_message", arguments={"sender_id": "USR2", "receiver_phone_number": "+17165558888", "content": "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm."})</p> |
| 8 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message sent successfully.'}</p> <p>Action: Function(name="get_messages_by_phone_number", arguments={"phone_number": "+17165558888"})</p> |
| 9 | <p>Observation: [Tool Result] {'success': True, 'data': [{"message_id": "MSG005", 'sender_id': 'USR2', 'receiver_id': '', 'receiver_phone_number': '+17165558888', 'content': 'Gabby, please join the video call at 4pm.', 'timestamp': '2024-04-02T08:45:19Z', 'delivery_status': 'failed', 'read_status': 'unread'}, {"message_id": "5b5f4580-f7db-4550-b6bd-0d7ef221741f", 'sender_id': 'USR2', 'receiver_id': '', 'receiver_phone_number': '+17165558888', 'content': 'Hi Gabby, the meeting is now at 4:30pm ET. Please confirm.', 'timestamp': '1763026165.0235567', 'delivery_status': 'sent', 'read_status': 'unread'}]}</p> <p>Action: Function(name="update_message_delivery_status", arguments={"message_id": "5b5f4580-f7db-4550-b6bd-0d7ef221741f", "new_status": "delivered"})</p> |
| 10 | <p>Observation: [Tool Result] {'success': True, 'message': 'Delivery status updated for message 5b5f4580-f7db-4550-b6bd-0d7ef221741f'}</p> <p>Action: Function(name="add_message_to_conversation", arguments={"conversation_id": "50aecb26-277a-478f-b9fe-cf5d89330781", "message_id": "5b5f4580-f7db-4550-b6bd-0d7ef221741f"})</p> |
| 11 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message added to conversation history.'}</p> <p>Action: Function(name="archive_conversation", arguments={"user_id": "USR2", "conversation_id": "50aecb26-277a-478f-b9fe-cf5d89330781"})</p> |
| 12 | <p>Observation: [Tool Result] {'success': True, 'message': 'Conversation archived for user.'}</p> <p>Action: Function(name="delete_message", arguments={"message_id": "MSG005"})</p> |
| 13 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message MSG005 has been deleted.'}</p> <p>Action: Function(name="get_user_by_id", arguments={"user_id": "USR1"})</p> |
| 14 | <p>Observation: [Tool Result] {'success': True, 'data': {'_id': 'USR1', 'phone_number': '+15551234567', 'name': 'Alice Chan', 'profile_info': 'admin'}}</p> <p>Action: Function(name="send_message", arguments={"sender_id": "USR2", "receiver_phone_number": "+15551234567", "content": "Update: re-sent Gabby's invite and delivery confirmed."})</p> |
| 15 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message sent successfully.'}</p> <p>Action: Function(name="get_messages_by_phone_number", arguments={"phone_number": "+15551234567"})</p> |
| 16 | <p>Observation: [Tool Result] {'success': True, 'data': [{"message_id": "MSG001", 'sender_id': 'USR1', 'receiver_id': 'USR2', 'receiver_phone_number': '+15559874566', 'content': 'Hey Brandon, can you send the report by Friday?', 'timestamp': '2024-02-17T14:38:05Z', 'delivery_status': 'delivered', 'read_status': 'read'}, {"message_id": "MSG002", 'sender_id': 'USR2', 'receiver_id': 'USR1', 'receiver_phone_number': '+15551234567', 'content': 'Hi Alice, yes I\'ll have it done tomorrow.', 'timestamp': '2024-02-17T15:00:51Z', 'delivery_status': 'delivered', 'read_status': 'unread'}, {"message_id": "MSG006", 'sender_id': 'USR5', 'receiver_id': '', 'receiver_phone_number': '+15551234567', 'content': 'Alice, your settings have been updated securely.', 'timestamp': '2024-06-01T10:02:46Z', 'delivery_status': 'sent', 'read_status': 'unread'}, {"message_id": "MSG007", 'sender_id': 'USR1', 'receiver_id': 'USR5', 'receiver_phone_number': '+812345678901', 'content': 'Emi, the new version is available for testing.', 'timestamp': '2024-05-15T07:17:22Z', 'delivery_status': 'delivered', 'read_status': 'read'}, {"message_id": "08e0a7b1-5039-433d-9dfd-7f7e4359d548", 'sender_id': 'USR2', 'receiver_id': 'USR1', 'receiver_phone_number': '+15551234567', 'content': 'Update: re-sent Gabby's invite and delivery confirmed.', 'timestamp': '1763026220.1726387', 'delivery_status': 'sent', 'read_status': 'unread'}]}</p> <p>Action: Function(name="mark_message_as_read", arguments={"message_id": "MSG002"})</p> |
| 17 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message marked as read.'}</p> <p>Action: Task Completed</p> |

Table 12: A trajectory example of Non-Conversation setting (The reasoning process <think>...</think> is ignored).

| Turn | Interaction |
|------|---|
| 1 | <p>Observation: [User] Hey! I need some help with an outreach issue involving Gabby Fields. Can you assist me with getting her contact added?</p> <p>Action: To add Gabby Fields as a contact, I'll need her phone number. Could you please provide Gabby's phone number so I can validate it and add her to your contact list?</p> |
| 2 | <p>Observation: [User] Sure, Gabby's mobile number is +17165558888. Let me know once you've validated it and added her to my contacts!</p> <p>Action: Function(name="validate_phone_number", arguments={"phone_number": "+17165558888"})</p> |
| 3 | <p>Observation: [Tool Result] {'success': True, 'data': {'valid': True, 'reason': 'Phone number is valid for messaging'}}</p> <p>Action: The phone number ***+17165558888** has been validated as valid for messaging. To proceed with adding Gabby Fields to your contacts, I'll need your **user ID** (the unique identifier associated with your account). Could you please share your user ID so I can complete this request?</p> |
| 4 | <p>Observation: [User] You can use my user ID: Brandon Lee (USR2) for this. Please go ahead and add Gabby Fields using that info.</p> <p>Action: Function(name="create_contact", arguments={"contact_id": "USR2", "contact_phone_number": "+17165558888", "contact_name": "Gabby Fields"})</p> |
| 5 | <p>Observation: [Tool Result] {'success': True, 'message': 'Contact created'}</p> <p>Action: Gabby Fields (phone: +17165558888) has been successfully added to your contacts as **Gabby Fields**. Your contact list now includes this entry under your account (USR2). Let me know if you'd like to send a message to Gabby or need further assistance!</p> |
| 6 | <p>Observation: [User] Great, thanks! Next, I want to send Gabby a new invite from Brandon. The message should be: "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm." Please send it and update the delivery status to "delivered."</p> <p>Action: Function(name="send_message", arguments={"sender_id": "USR2", "receiver_phone_number": "+17165558888", "content": "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm."})</p> |
| 7 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message sent successfully.'}</p> <p>Action: Function(name="update_message_delivery_status", arguments={"message_id": "MSG123", "new_status": "delivered"})</p> |
| 8 | <p>Observation: [Tool Result] {'success': False, 'error': 'Message not found'}</p> <p>Action: Function(name="get_messages_by_phone_number", arguments={"phone_number": "+17165558888"})</p> |
| 9 | <p>Observation: [Tool Result] {'success': True, 'data': [{"message_id": "MSG005", "sender_id": "USR2", "receiver_id": "", "receiver_phone_number": "+17165558888", "content": "Gabby, please join the video call at 4pm.", "timestamp": "2024-04-02T08:45:19Z", "delivery_status": "failed", "read_status": "unread"}, {"message_id": "c02f3c58-a015-40ee-b034-390b649c472e", "sender_id": "USR2", "receiver_id": "", "receiver_phone_number": "+17165558888", "content": "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm.", "timestamp": "1763054273.124194", "delivery_status": "sent", "read_status": "unread"}]}</p> <p>Action: Function(name="update_message_delivery_status", arguments={"message_id": "c02f3c58-a015-40ee-b034-390b649c472e", "new_status": "delivered"})</p> |
| 10 | <p>Observation: [Tool Result] {'success': True, 'message': 'Delivery status updated for message c02f3c58-a015-40ee-b034-390b649c472e'}</p> <p>Action: The message to Gabby Fields has been successfully sent and the delivery status updated to **"delivered"**. Here's the confirmation:
 Message Content: "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm."
 Delivery Status: Delivered
 Message ID: c02f3c58-a015-40ee-b034-390b649c472e
 Let me know if you'd like to send another message or need further assistance!</p> |
| 11 | <p>Observation: [User] Perfect! Can you please link this new message to the existing Brandon-Gabby conversation? Once that's done, go ahead and archive the whole conversation for me.</p> <p>Action: Function(name="create_conversation", arguments={"user_id1": "USR2", "user_id2": "+17165558888"})</p> |
| 12 | <p>Observation: [Tool Result] {'success': False, 'error': 'Participant 2 (+17165558888) does not exist.'}</p> <p>Action: Function(name="get_contact_by_phone_number", arguments={"phone_number": "+17165558888"})</p> |
| 13 | <p>Observation: [Tool Result] {'success': True, 'data': {'_id': 'contact_2', 'contact_id': 'USR1', 'contact_phone_number': '+17165558888', 'contact_name': 'Gabby Fields'}}</p> <p>Action: Function(name="create_conversation", arguments={"user_id1": "USR2", "user_id2": "contact_2"})</p> |
| 14 | <p>Observation: [Tool Result] {'success': True, 'message': 'Conversation created', 'conversation': {'conversation_id': 'c3d610bf-1100-405f-8897-8df5cea96b1d', 'user_ids': ['USR2', 'contact_2'], 'message_id': ""}}</p> <p>Action: Function(name="add_message_to_conversation", arguments={"conversation_id": "c3d610bf-1100-405f-8897-8df5cea96b1d", "message_id": "c02f3c58-a015-40ee-b034-390b649c472e"})</p> |
| 15 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message added to conversation history.'}</p> <p>Action: Function(name="archive_conversation", arguments={"user_id": "USR2", "conversation_id": "c3d610bf-1100-405f-8897-8df5cea96b1d"})</p> |
| 16 | <p>Observation: [Tool Result] {'success': True, 'message': 'Conversation archived for user.'}</p> <p>Action: The message has been successfully linked to the Brandon-Gabby conversation (ID: 'c3d610bf-1100-405f-8897-8df5cea96b1d'), and the entire conversation has been archived for your account (USR2).
 Summary: Message "Hi Gabby, the meeting is now at 4:30pm ET. Please confirm." linked to conversation Conversation archived for future reference
 Let me know if you'd like to unarchive or take any other action!</p> |
| 17 | <p>Observation: [User] Thanks! Now, please delete the old failed message Brandon sent to Gabby from that conversation. Let me know once it's removed.</p> <p>Action: Function(name="delete_message", arguments={"message_id": "MSG005"})</p> |
| 18 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message MSG005 has been deleted.'}</p> <p>Action: The old failed message (ID: 'MSG005') has been successfully deleted from the conversation. The conversation now only includes the confirmed message: **"Hi Gabby, the meeting is now at 4:30pm ET. Please confirm."**
 Let me know if you need further cleanup or assistance!</p> |

| Turn | Interaction |
|------|---|
| 19 | <p>Observation: [User] Great, thank you. Lastly, send this short status update from Brandon to Alice Chan: "Update: re-sent Gabby's invite and delivery confirmed." And please mark Alice's previously unread message from Brandon as "read." Let me know when that's done.</p> <p>Action: To send the status update to Alice Chan and mark her unread message as read, I'll need **Alice Chan's phone number** to proceed. Could you please provide her phone number so I can validate it, send the message, and update the read status?</p> |
| 20 | <p>Observation: [User] I don't have Alice's phone number available, but her user ID is Alice Chan (USR1). Can you go ahead and send the update to her using that, and mark her unread message from Brandon as "read"?</p> <p>Action: Function(name="get_user_by_id", arguments={"user_id": "USR1"})</p> |
| 21 | <p>Observation: [Tool Result] {'success': True, 'data': {'_id': 'USR1', 'phone_number': '+15551234567', 'name': 'Alice Chan', 'profile_info': 'admin'}}</p> <p>Action: Function(name="send_message", arguments={"sender_id": "USR2", "receiver_phone_number": "+15551234567", "content": "Update: re-sent Gabby's invite and delivery confirmed."})</p> |
| 22 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message sent successfully.'}</p> <p>Action: Function(name="get_messages_by_phone_number", arguments={"phone_number": "+15551234567"})</p> |
| 23 | <p>Observation: [Tool Result] {'success': True, 'data': [{'_id': 'MSG001', 'sender_id': 'USR1', 'receiver_id': 'USR2', 'receiver_phone_number': '+15559874566', 'content': 'Hey Brandon, can you send the report by Friday?', 'timestamp': '2024-02-17T14:38:05Z', 'delivery_status': 'delivered', 'read_status': 'read'}, {'_id': 'MSG002', 'sender_id': 'USR2', 'receiver_id': 'USR1', 'receiver_phone_number': '+15551234567', 'content': "Hi Alice, yes I'll have it done tomorrow.", 'timestamp': '2024-02-17T15:00:51Z', 'delivery_status': 'delivered', 'read_status': 'unread'}, {'_id': 'MSG006', 'sender_id': 'USR5', 'receiver_id': '', 'receiver_phone_number': '+15551234567', 'content': 'Alice, your settings have been updated securely.', 'timestamp': '2024-06-01T10:02:46Z', 'delivery_status': 'sent', 'read_status': 'unread'}, {'_id': 'MSG007', 'sender_id': 'USR1', 'receiver_id': 'USR5', 'receiver_phone_number': '+812345678901', 'content': 'Emi, the new version is available for testing.', 'timestamp': '2024-05-15T07:17:22Z', 'delivery_status': 'delivered', 'read_status': 'read'}, {'_id': 'b2e86264-96cb-40c0-b7dd-39091b185d15', 'sender_id': 'USR2', 'receiver_id': 'USR1', 'receiver_phone_number': '+15551234567', 'content': 'Update: re-sent Gabby's invite and delivery confirmed.', 'timestamp': '1763054573.840822', 'delivery_status': 'sent', 'read_status': 'unread'}]}</p> <p>Action: Function(name="mark_message_as_read", arguments={"message_id": "MSG002"})</p> |
| 24 | <p>Observation: [Tool Result] {'success': True, 'message': 'Message marked as read.'}</p> <p>Action: The message to Alice Chan has been successfully sent: **"Update: re-sent Gabby's invite and delivery confirmed."** Additionally, Alice's previously unread message from Brandon (ID: 'MSG002') has been marked as **read**. All actions completed! Let me know if you need further assistance.</p> |

Table 13: A trajectory example of Conversation setting (The reasoning process <think>...</think> is ignored).