REAGENT: RETHINKING AGENT TRAINING THROUGH SCENARIO CO-DESIGN

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic reinforcement learning scales training by expanding programmatic environments and synthesizing diverse tasks. Yet this scaling can paradoxically narrow training coverage through how data are constructed. To make such synthesis tractable at scale, existing pipelines often derive tasks from executable paths or successful workflows, thereby inducing a **construction-induced selection bias** toward feasible requests. Real users, however, state goals without full visibility into the environment state or its constraints, so agents must identify conflicts, guide users toward feasible revisions, and refuse when necessary. Such resolution behaviors cannot be captured by task synthesis alone, as conflicts and valid resolutions are co-constructed by environment constraints, user intent and appropriate reward. This interdependence motivates us to rethink agent training by shifting the unit of construction from feasible tasks alone to *the whole training scenario* (i.e., task, user, and reward together). We call this principle **scenario co-design** and instantiate it through **ReAgent**. Starting from conflicts and resolutions grounded in the environment, ReAgent jointly constructs the task request, simulated user behavior, and reward criteria as a coherent conflict scenario, enabling scalable training with broader coverage of feasible, blocked, and partially resolvable requests. Empirically, augmenting training with these co-designed conflict scenarios not only directly improves Qwen3-8B and Qwen3-30B-A3B-Thinking-2507 on constraint-governed agent benchmarks such as τ^2-Bench but also even enables them to outperform competitive baselines across general agent benchmarks. These results suggest that broadening training coverage within environments through scenario co-design offers a promising direction for scaling agent training beyond simply expanding environments and feasible tasks.

1 INTRODUCTION

Agentic reinforcement learning (RL) trains large language model (LLM) agents through programmatic environments that implement tools as executable code, maintain dynamic state, and return verifiable feedback (Song et al., 2026; Tu et al., 2026; Xu et al., 2026). Recent work expands these environments and synthesizes tasks across increasingly diverse domains, from web navigation to customer service (Qi et al., 2025; Gao et al., 2026). This progress rests on a natural but consequential scaling assumption that sufficiently broad environment coverage and diverse task workflows will prepare agents for the various interactions encountered in real-world deployment (Fang et al., 2026).

We argue, however, that the common practices adopted in pursuit of this assumption can paradoxically *narrow the range of interactions* represented in training through a **construction-induced selection bias**. To make large-scale synthesis tractable, existing pipelines typically derive tasks from executable tool-call chains or successful workflows that already admit a completion path (Chen et al., 2026a), and build the remaining scenario components (e.g., user agents and rewards) around these feasible tasks (Prabhakar et al., 2025; Qian et al., 2025), forming *feasible scenarios*. Feasibility thereby becomes an implicit precondition for a request to enter training. Scaling can thus span more domains and more complex workflows, yet the interactions represented in training remain confined to cases where the user's request is already compatible with the environment's state and constraints.

But real requests in deployment are not naturally compatible in this way. Users cannot observe the state of the environment or know its specific constraints, so a request may be feasible as stated, may

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

require revision before it becomes feasible, or may have to be blocked. Consider the case in Appendix D.2, for instance, a user asks to approve a parent appeal immediately, while the environment updates a parent appeal only after all of its child appeals are completed, so the correct behavior is not to execute the request as stated but to guide the user toward a compliant path. However, such resolution behaviors cannot be captured by task synthesis alone, as conflicts and valid resolutions are co-constructed by environment constraints, user intent, and appropriate reward. Without the user and reward carefully designed for the interactive resolution of conflicts, an agent cannot learn to take the right action in each situation. This interdependence motivates us to rethink agent training by shifting the unit of construction from feasible tasks alone to *the whole training scenario*, i.e., task, user, and reward together. We call this principle **scenario co-design**.

We introduce **ReAgent**, which constructs *conflict scenarios* at scale via scenario co-design to broaden coverage beyond feasible scenarios. From a programmatic environment's state and constraints, ReAgent identifies candidate conflicts and their valid resolutions, and jointly constructs a **Task–Playbook–Oracle package**. The task states the user request, the playbook drives the simulated user's behavior, and the oracle records prohibited operations and valid resolutions that guide reward construction. ReAgent then cross-checks the package components for semantic consistency and retains only coherent scenarios. Semantic coherence alone, however, does not guarantee executable correctness, as bugs in synthesized code can fabricate spurious conflicts. ReAgent therefore applies **execution-guided code curation** to every executable component, discarding any scenario whose code remains faulty after repair so that only semantically coherent, execution-verified data enter training. For training, ReAgent adopts a **two-stage RL curriculum** that progresses from easier to harder capabilities. A Multi-Step stage first trains the agent with tools alone on feasible scenarios to build a foundation of tool use, and a Multi-Turn stage then activates simulated users over feasible and conflict scenarios, extending this foundation to user interaction and conflict resolution.

We construct 2,415 programmatic environments with 4,513 feasible Multi-Step scenarios alongside 2,360 feasible and 2,252 conflict Multi-Turn scenarios. On τ^2-Bench, ReAgent models based on Qwen3-8B and Qwen3-30B-A3B-Thinking-2507 improve over their corresponding baselines across feasible, blocked, and partially resolvable requests, with relative gains ranging from 14.4% to 37.2% across request types and model scales. The advantage extends to other general agent benchmarks, where ReAgent-8B even ranks first among all evaluated 8B agent models. Ablations further validate the contributions of conflict scenarios, code curation, and the two-stage curriculum. Together, these results show that scenario co-design broadens training coverage within environments, offering a much-needed complement to scaling environments and feasible tasks. Our contributions are:

- We identify a construction-induced selection bias in current agent task-synthesis pipelines that systematically underrepresents user–environment conflicts common in real-world deployment.
- We rethink how agent training data are constructed and propose scenario co-design, which holds that producing coherent conflict scenarios requires jointly constructing the task, user, and reward as a whole. We apply this in ReAgent, a scalable framework for conflict scenario construction.
- We validate ReAgent at two model scales on three benchmarks, showing consistent gains on both general agent tasks and tasks requiring adequate conflict resolution.

2 Problem Setup: Environments and Training Scenarios

Agentic RL trains an agent through scenarios instantiated in programmatic environments. In this section, we first define the environment that supports interaction, and then formalize a training scenario as the combination of an environment instance, task, user specification, and reward criteria.

Programmatic environments. We adopt programmatic environments to obtain deterministic state transitions. Following Song et al. (2026), we represent a programmatic environment at two levels: environment skeleton and environment instance. The environment skeleton is represented as

$$\mathcal{E} = \langle E_{\text{doc}}, F_{\text{exec}}, \Sigma_{\text{tool}} \rangle. \tag{1}$$

Observed by the agent, the documentation E_{doc} describes the environment and its environment-specific constraints Φ_E; some constraints also suggest alternative ways to handle violations. The executable program F_{exec} defines the state structure and implements the tool code for querying or

updating that state. The interface set Σ_{tool} specifies each tool through its name, arguments, and description, with every $f \in \Sigma_{\text{tool}}$ implemented in F_{exec}. The agent observes E_{doc} and Σ_{tool}, while F_{exec} remains hidden behind these interfaces. Given this skeleton, we obtain the environment instance by assigning an initialization configuration s_0 to the environment skeleton, yielding $e = \langle \mathcal{E}, s_0 \rangle$. At turn t, s_t denotes the current state of the environment instance. The agent cannot observe s_t directly and must invoke tools in Σ_{tool} to query the state or update it to s_{t+1}.

Training scenarios. A training scenario specifies the agentic interaction in Multi-Step or Multi-Turn training. We define a training scenario as

$$\mathcal{X} = \langle e, q, p_u, \rho_{\mathcal{X}} \rangle, \tag{2}$$

where e is the environment instance, q is the task presented to the agent, p_u specifies the simulated user's information and behavior, and $\rho_{\mathcal{X}}$ specifies the set of reward criteria. At the start of each episode, the agent receives E_{doc}, Σ_{tool}, and q. In Multi-Step scenarios, q contains all information that must come from the user and $p_u = \varnothing$, so no further user interaction occurs. In Multi-Turn scenarios, $p_u \neq \varnothing$ and the simulated user remains active, allowing the agent to obtain additional user-held information through dialogue. In either setting, the agent conditions on the interaction history $h_t = (q, a_0, o_1, \ldots, a_{t-1}, o_t)$ at turn t to sample an action $a_t \sim \pi_\theta(\cdot \mid E_{\text{doc}}, \Sigma_{\text{tool}}, h_t)$ from $\mathcal{A} = \mathcal{A}_{\text{tool}} \cup \mathcal{A}_{\text{resp}}$. A tool action invokes an interface in Σ_{tool}, returns an observation o_{t+1}, and may update the environment state. In Multi-Turn settings, a response action produces a natural-language message for the user and elicits a reply $o_{t+1} \sim \pi_u(\cdot \mid h_t, a_t, p_u)$. An episode that terminates after T turns yields a trajectory $\tau = \big(q, (a_t, o_{t+1})_{t=0}^{T-1}\big)$ and a terminal environment state s_T. The reward criteria are then applied to the terminal state and trajectory to determine the reward $r_{\mathcal{X}} = \rho_{\mathcal{X}}(s_T, \tau)$.

Scenario co-design for conflict scenarios. Constructing a conflict scenario requires more than carefully crafting q alone. The user specification p_u must define how the user responds to the conflict and under what conditions they accept a revision, while $\rho_{\mathcal{X}}$ must adequately evaluate the valid resolutions. Otherwise, the user may accept a compliant revision while the reward still requires completion of the original request. Scenario co-design therefore calls for taking the whole training scenario as the unit of construction, jointly grounding the task, user behavior, and reward criteria in conflicts and valid resolutions supported by e. Section 3 instantiates this principle in ReAgent.

3 REAGENT

In this section, we present ReAgent, which co-designs semantically coherent conflict scenarios (Section 3.1), curates code for executable correctness (Section 3.2), and uses a two-stage RL curriculum to progress from simple tool use to complex conflict resolution (Section 3.3). Together, these components make conflict scenarios constructible and trainable at scale. Figure 1 gives an overview.

Environment construction. As defined in Section 2, our programmatic environments $\mathcal{E}$ expose tools through interfaces and constraints through documentation, while their Python code implements state structures and tools. Following EnvScaler (Song et al., 2026), we first collect materials from public tool-use resources and synthetic task specifications. We then use an LLM to retain materials suitable for constructing stateful programmatic environments and abstract each into a task-independent environment description. We deduplicate and cluster these descriptions to retain representative candidates. For each candidate, an LLM plans the state structure, tools, and constraints and implements the state structure and tools as a Python class. After code curation (Section 3.2), we generate an initial configuration s_0 for each retained $\mathcal{E}$, yielding $e = \langle \mathcal{E}, s_0 \rangle$. The resulting instances provide the executable foundation for subsequent scenario construction. Notably, all LLM-based components in this section use Qwen3.6-27B (Qwen Team, 2026) unless otherwise specified.

3.1 CONFLICT SCENARIO CO-DESIGN

Conflict-and-resolution specification. Before synthesizing a conflict scenario, we use LLMs to identify reasonable conflicts grounded in an environment instance $e = \langle \mathcal{E}, s_0 \rangle$. A *constraint-based conflict* occurs when a requested action or outcome violates an environment-specific constraint in Φ_E, whereas a *state-based conflict* occurs when a request condition or user assumption is inconsistent with s_0. A scenario may contain multiple conflicts of either or both types. For each selected

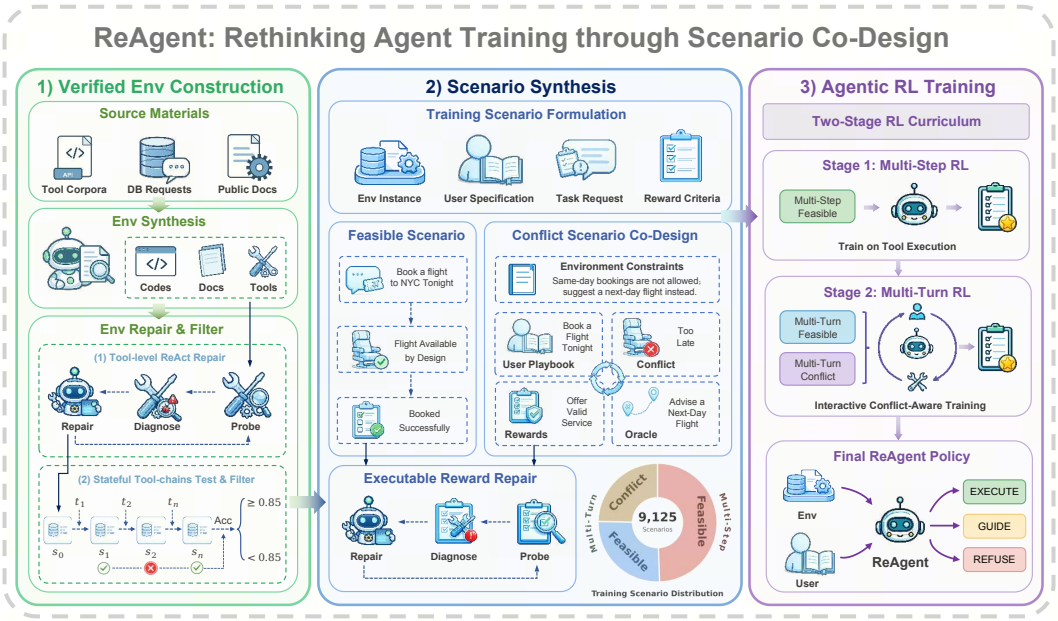


Figure 1: **Overview of ReAgent.** To make conflict scenarios constructible and trainable, ReAgent co-designs the Task–Playbook–Oracle package around environment-grounded conflicts and resolutions to ensure semantic coherence, and curates code for executable correctness. ReAgent also uses two-stage RL to progressively develop agents' tool-use and conflict-resolution abilities.

conflict, the LLM proposes environment-specific resolutions that respect the environment's state facts and constraints, including completing the feasible parts, guiding the user toward a compliant revision, or refusing the infeasible part when no compliant revision is available.

Jointly construct the task, playbook, and oracle. Given e, an LLM analyzes the selected conflicts and resolutions to jointly construct a task d, a user playbook p, and a structured oracle ω:

$$e \xrightarrow{\text{conflicts and valid resolutions}} \mathcal{P}_{\mathcal{X}} = \langle d, p, \omega \rangle. \tag{3}$$

The three components serve distinct roles while representing the same selected conflicts and valid resolutions. The *task* d states the user request. The *user playbook* p instantiates the user specification p_u and records the user's information, disclosure behavior, revision conditions, and stopping condition, governing how the user responds to the agent. The *structured oracle* ω organizes conflict-resolution supervision into six fields: `immutable_facts`, `environment_constraints`, `guidance_boundaries`, `refusal_boundaries`, `required_agent_behaviors`, and `non_negotiable_checks`. The first two record relevant state facts and constraints for the selected conflicts. The guidance and refusal boundaries specify what the agent should and should not do when guiding or refusing the user. Based on the task and these boundaries, the required agent behaviors specify the key actions that the agent must query, execute, guide and refuse to resolve the scenario correctly. The non-negotiable checks turn these mandatory requirements into an evaluation checklist. After constructing the package, we use LLMs to cross-check its components against e, jointly revise them to resolve inconsistencies among the user goal, user responses, and valid resolutions, and filter out low-quality packages, retaining only semantically coherent scenarios. Finally, we rewrite d into the agent-facing task q by removing any tool-retrievable facts or detailed operation sequences. Appendix D.2 provides a complete example.

Reward formulation. From each retained package, we use LLMs to construct the reward-criterion set $\rho_{\mathcal{X}}$. Current feasible-task synthesis typically relies on programmatic check functions for reward computation (Tu et al., 2026; Song et al., 2026). These functions efficiently evaluate multiple conditions on the terminal environment state s_T, averaging their binary outcomes into a task reward. For conflict scenarios, however, this formulation is insufficient because correctness depends not only on

the final outcome but also on how the agent reaches it. In particular, whether the agent responds with appropriate guidance or refusal cannot be reliably inferred from s_T alone. Moreover, simple averaging allows success on other criteria to compensate for non-negotiable violations. We therefore organize the reward criteria by evidence source and severity, yielding four types:

$$\rho_{\mathcal{X}} = \rho_{\mathcal{X}}^{\text{func}} \cup \rho_{\mathcal{X}}^{\text{func-hard}} \cup \rho_{\mathcal{X}}^{\text{llm}} \cup \rho_{\mathcal{X}}^{\text{llm-hard}}. \tag{4}$$

Function criteria are programmatic check functions over s_T, whereas LLM criteria are rubric-based evaluations of the trajectory τ. Every criterion may be regular or hard, depending on its severity. To construct $\rho_{\mathcal{X}}$, we first use LLMs to derive a checklist from the package. Each check item is expressed as a natural-language requirement and assigned a type according to its evidence source and severity. We then implement function-tagged items as Python check functions, while LLM-tagged items remain natural-language rubrics. In addition to these synthesized criteria, we directly convert the conflict-resolution supervision encoded in the oracle ω into hard rubrics and include them in $\rho_{\mathcal{X}}^{\text{llm-hard}}$, preventing the criterion set synthesized by LLMs from omitting key checks needed to determine whether the agent resolves the scenario correctly.

Let $z_c \in \{0, 1\}$ denote whether criterion c is satisfied. As suggested in Chen et al. (2026b), we penalize redundant tool use with $P_{\text{dup}}(\tau) = \min(0.02 N_{\text{dup}}(\tau), 0.1)$, where $N_{\text{dup}}(\tau)$ counts the distinct $(\texttt{tool_name}, \texttt{arguments})$ pairs that occur more than once in τ. The final reward is

$$r_{\mathcal{X}}(s_T, \tau) = \begin{cases} 0, & \exists c \in \rho_{\mathcal{X}}^{\text{func-hard}} \cup \rho_{\mathcal{X}}^{\text{llm-hard}} : \ z_c = 0, \\ \max\left(0, \dfrac{1}{|\rho_{\mathcal{X}}|} \displaystyle\sum_{c \in \rho_{\mathcal{X}}} z_c - P_{\text{dup}}(\tau)\right), & \text{otherwise.} \end{cases} \tag{5}$$

Thus, any failed hard criterion sets the reward to zero; otherwise, the reward averages all criterion outcomes and subtracts the repetition penalty.

3.2 Code Curation

Even when the package is semantically coherent, the synthesized environment and reward functions may contain bugs. We therefore curate both through iterative testing, repair, and filtering. Appendix A provides the details.

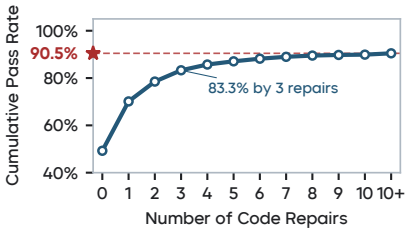


Figure 2: **Cumulative pass rate in tool-level repair stage.** The number of repairs is summed across all tools in each environment.

Environment curation. We first validate the syntax and class structure of each environment through abstract syntax tree checks. Then we conduct a ReAct-guided repair loop (Yao et al., 2023) for each tool. In this process, an LLM generates a pair of test calls: a positive call with valid arguments and a negative call with invalid arguments. In each repair round, we execute both calls and assess whether their outcomes meet expectations. If either call fails, the LLM uses the feedback to diagnose and repair the relevant tool or class code before rerunning both calls. If either call still fails after up to five repair rounds, we discard the environment. Figure 2 shows the cumulative fraction of all synthesized environments that pass tool-level repair within a given total number of code repairs. For these filtered environments, we further conduct stateful tool-chain testing to verify state transitions across consecutive tool calls. Over 30 steps, a testing LLM selects one tool call at a time based on the current environment state. After each call is executed, the resulting state is used to select the next call. A checking LLM determines whether each call behaves as expected based on its input and output. We retain only environments in which at least 85% of calls are judged acceptable.

Reward curation. We apply a similar iterative procedure to each check function in $\rho_{\mathcal{X}}$. In this process, an LLM constructs one positive state that satisfies the criterion of this check function and one negative state that violates it, requiring the function to return `True` and `False`, respectively. If either result is incorrect, the LLM uses the execution feedback to repair the function and reruns both examples. We discard the scenario if the function still fails after three repair rounds. LLM rubrics do not involve executable functions and are therefore excluded from this executable repair process, but we separately validate their semantic quality and the reliability of rollout judges in Appendix C.3.

Table 1: **Main results on agentic benchmarks (Avg@4, %).** Among open-source foundation and agent-trained models, the best score is **bold** and the second best is underlined. We do not report τ^2-Bench or ACEBench-Agent results for the Agent-World models because their checkpoints are not publicly available.

Models	BFCL-V4							ACEBench-Agent			τ^2-Bench			
	Memory	Multi-T.	No Live	Live	Relev.	Irrelev.	Avg.	Multi-Step	Multi-Turn	Avg.	Retail	Telecom	Airline	Avg.
Frontier Proprietary Models														
GPT-5.2 High⋆	45.8	48.5	81.9	70.4	75.0	88.7	59.8	–	–	–	–	–	–	–
Claude Sonnet-4.5⋆	65.0	61.4	88.7	81.1	68.8	86.6	71.3	–	–	–	–	–	–	–
Gemini-3 Pro⋆	61.7	60.8	90.7	83.1	68.8	85.6	70.7	–	–	–	–	–	–	–
Seed 2.0⋆	57.8	62.3	89.0	82.2	76.6	75.0	68.6	–	–	–	–	–	–	–
Open-Source Foundation Models														
DeepSeek-V3.2-685B⋆	**54.2**	37.4	34.9	53.7	37.5	**93.2**	50.3	–	–	–	–	–	–	–
Qwen3-8B	19.5	38.4	89.0	81.2	93.8	78.1	50.3	62.5	46.7	54.6	46.1	37.9	19.0	34.3
Qwen3-14B⋆	19.8	36.9	90.0	82.4	81.3	79.4	50.3	72.5	56.7	64.6	51.1	40.4	38.0	43.1
Qwen3-32B⋆	15.7	43.3	**90.3**	82.0	81.3	82.4	52.0	72.5	57.5	65.0	54.2	31.8	49.5	45.2
Qwen3-30B-A3B-Thinking-2507	25.2	49.4	89.7	**85.1**	98.4	80.4	56.7	92.5	53.3	72.9	65.6	33.4	58.0	52.3
Open-Source Agent Training Methods														
AgenticQwen-8B†	24.2	38.7	42.0	78.8	82.8	78.4	45.4	68.8	58.3	63.5	52.0	45.4	31.5	43.0
EnvFactory-8B	20.3	38.6	86.9	81.7	**100.0**	72.7	49.7	67.5	51.7	59.6	55.5	42.3	29.0	42.3
EnvScaler-8B	22.6	48.3	88.1	82.3	98.4	76.5	54.6	85.0	55.0	70.0	58.7	38.8	29.5	42.3
AWM-8B	16.4	38.2	88.8	80.8	95.3	75.9	49.1	57.5	48.3	52.9	46.5	35.7	29.5	37.2
Agent-World-8B⋆	21.7	44.5	83.3	79.6	93.8	80.2	52.5	–	–	–	–	–	–	–
Agent-World-14B⋆	23.9	53.9	82.3	79.3	93.8	81.0	56.5	–	–	–	–	–	–	–
ReAgent-8B	28.9	50.8	87.2	82.4	93.8	76.1	57.0	**96.3**	**65.8**	**81.0**	65.6	**63.4**	48.5	59.1
ReAgent-30B-A3B	32.0	**55.0**	88.2	82.6	84.4	87.8	**61.0**	95.0	61.7	78.3	**79.4**	56.6	**63.5**	**66.5**

† AgenticQwen-8B is evaluated in non-thinking mode following the original paper (Lyu et al., 2026).
⋆ For these models, BFCL-V4 results are from Agent-World (Dong et al., 2026); see Appendix B.3 for alignment of our evaluation and Agent-World on BFCL-V4. However, all τ^2-Bench and ACEBench-Agent results are ours.

3.3 The Dataset and the Two-Stage RL Curriculum

The construction and curation procedures above yield 2,415 execution-verified environments and 2,252 conflict Multi-Turn scenarios. To retain broad tool-use coverage alongside conflict resolution, we additionally construct 4,513 feasible Multi-Step scenarios and 2,360 feasible Multi-Turn scenarios. Appendix A details their construction process, and Table 5 reports the full dataset statistics.

With these data, we train with GRPO (Shao et al., 2024) in two stages. In the Multi-Step RL stage, we train the agent with tools alone on feasible Multi-Step scenarios, where q contains all user-held information and no simulated user is active, to establish a foundation of tool use. In the Multi-Turn RL stage, we activate simulated users and continue training on feasible and conflict Multi-Turn scenarios, extending this foundation to user interaction and conflict resolution. Together, the two stages enable the agent to benefit from the broader interaction coverage introduced by scenario co-design, progressing from tool use to conflict resolution.

4 Experiments

In this section, we evaluate ReAgent by testing whether conflict scenarios constructed through scenario co-design improve conflict-resolution capabilities (Section 4.3) while preserving general tool-use competence (Section 4.2). We further conduct ablation studies to examine the roles of conflict scenarios, execution-guided code curation, and the two-stage curriculum in ReAgent (Section 4.4).

4.1 Experimental Setup

Training. We instantiate ReAgent with Qwen3-8B and Qwen3-30B-A3B-Thinking-2507 (Yang et al., 2025), using the scenarios constructed in Section 3 and holding out 100 randomly sampled scenarios per stage for validation. Both models follow the Multi-Step-to-Multi-Turn RL curriculum with thinking enabled. Checkpoints are selected by validation reward within each stage. For

Table 2: **Performance across request types on τ^2-Bench (Avg@4, %).** In Overall, underlining marks the strongest baseline for each request type within each model-scale block; bold shaded entries are ReAgent scores, with relative gains over the underlined baselines shown in dark-orange. Agent-World is not included because its model checkpoints are not publicly available so we cannot obtain the needed scores.

Models	Airline			Retail			Telecom		Overall (Micro Mean)		
	FEASIBLE $N=11$	BLOCKED $N=22$	PARTIAL $N=17$	FEASIBLE $N=79$	BLOCKED $N=5$	PARTIAL $N=30$	FEASIBLE $N=94$	BLOCKED $N=20$	FEASIBLE $N=184$	BLOCKED $N=47$	PARTIAL $N=47$
Qwen3-8B	29.5	23.9	5.9	50.3	80.0	29.2	41.2	22.5	44.4	29.3	20.7
AgenticQwen-8B	27.3	44.3	17.7	55.4	95.0	35.8	49.7	25.0	50.8	41.5	29.3
EnvFactory-8B	43.2	29.5	19.1	61.7	85.0	34.2	46.5	22.5	52.9	32.4	28.7
EnvScaler-8B	47.7	33.0	13.2	62.7	75.0	45.7	42.3	22.5	51.4	33.0	33.9
AWM-8B	34.1	40.9	11.8	48.4	80.0	35.8	39.4	18.8	42.9	35.6	27.1
ReAgent-8B	45.5	71.6	20.6	69.6	100.0	49.2	70.5	30.0	**68.6**$^{+29.8\%}$	**56.9**$^{+37.2\%}$	**38.8**$^{+14.4\%}$
Qwen3-30B-A3B-Thinking-2507	63.6	80.7	25.0	68.0	85.0	55.8	36.2	20.3	51.5	55.4	44.7
ReAgent-30B-A3B	56.8	92.1	30.9	79.8	100.0	75.0	61.0	40.0	**68.3**$^{+32.7\%}$	**70.7**$^{+27.6\%}$	**59.0**$^{+32.1\%}$

Qwen3-8B, we additionally perform cold-start SFT on 10K trajectories from feasible scenarios before reinforcement learning. Appendix B.1 gives more details on training setup.

Evaluation and comparisons. We evaluate on τ^2-Bench (Barres et al., 2026) for state- and constraint-dependent service interactions, ACEBench-Agent (Chen et al., 2025) for Multi-Step execution and Multi-Turn user interaction, and BFCL-V4 (Patil et al., 2025) for general function calling. Comparisons include proprietary and open-source foundation models and five agent-training methods, AgenticQwen (Lyu et al., 2026), EnvFactory (Xu et al., 2026), EnvScaler (Song et al., 2026), AWM (Wang et al., 2026), and Agent-World (Dong et al., 2026). All models evaluated by us use the same implementation and evaluation settings, ensuring a controlled comparison. Detailed evaluation configurations are provided in Appendix B.2.

4.2 Performance across Agent Benchmarks

As shown in Table 1, ReAgent-8B ranks first among the evaluated open-source 8B models on all three benchmarks. ACEBench Multi-Step improves alongside Multi-Turn, indicating that broader interaction coverage does not come at the expense of tool execution. This balance is essential, since an agent that resolves conflicts must still possess effective tool-use capabilities. The pattern persists at the larger scale. ReAgent-30B-A3B ranks first among the reported open-source models on τ^2-Bench and BFCL-V4 and second on ACEBench-Agent. Relative to its Qwen3-30B-A3B-Thinking-2507 backbone, it improves the three benchmark averages by 14.2, 5.4, and 4.3 points, respectively. Taken together, these cross-benchmark gains show that ReAgent's advantage is not confined to conflict-heavy interactions and does not come at the expense of general tool-use competence. Instead, the full ReAgent training recipe improves both environment-grounded interaction and general agent performance across scales.

4.3 Performance across Request Types

To further evaluate conflict-resolution performance across request types, we manually partition τ^2-Bench requests by the resolution warranted by environment states and constraints. FEASIBLE requests can be completed as stated; BLOCKED requests have a prohibited core goal; and PARTIAL requests require either guiding the user toward a compliant revision or simply completing the feasible part of the request. Table 2 reports the results. At both scales, ReAgent outperforms the corresponding baselines across all three request types. In terms of average performance, the gains on BLOCKED requests are accompanied by improvements on FEASIBLE and PARTIAL requests. PARTIAL remains the most challenging category because success requires recognizing the conflict and still pursuing a valid resolution through guidance or partial completion. Together, these results show that scenario co-design broadens training coverage beyond requests that are feasible as stated, improving agents' ability to execute, guide, and refuse as appropriate rather than simply increasing refusal tendency. We observe similar results on τ-Bench (Yao et al., 2025) reported in Table 8.

Table 3: **Incremental effects of the two training stages (Avg@4, %).** The stages are cumulative within each backbone: Multi-Step training performs Multi-Step RL, while Multi-Turn training continues from that checkpoint with Multi-Turn interactions.

Models	τ^2-Bench				ACEBench-Agent			BFCL-V4 Multi-Turn				
	Retail	Telecom	Airline	**Avg.**	Multi-Step	Multi-Turn	**Avg.**	Base	Miss-F.	Miss-P.	Long.	**Avg.**
Qwen3-8B	46.1	37.9	19.0	34.3	62.5	46.7	54.6	50.3	38.8	35.8	28.8	38.4
+ SFT	50.4	34.2	39.0	41.2	77.5	50.8	64.2	61.8	47.8	37.3	36.8	45.9
+ Multi-Step RL	61.2	55.7	36.0	51.0	87.5	57.5	72.5	70.4	50.5	34.3	43.4	49.6
+ **Multi-Turn RL**	65.6	63.4	48.5	59.1	96.3	65.8	81.0	70.5	51.1	37.1	44.5	50.8
Qwen3-30B-A3B-Thinking-2507	65.6	33.4	58.0	52.3	92.5	53.3	72.9	66.8	53.5	34.9	42.6	49.4
+ Multi-Step RL	73.7	51.1	61.5	62.1	92.5	56.7	74.6	71.0	57.4	41.3	46.0	53.9
+ **Multi-Turn RL**	79.4	56.6	63.5	66.5	95.0	61.7	78.3	71.8	56.9	45.3	46.0	55.0

Table 4: **Ablations on training-scenario construction (Avg@4, %).** At Multi-Turn stage, we study removing conflict scenarios or removing LLM rubrics. We construct additional feasible scenarios to replace the removed conflict scenarios to keep training-set size fixed. We separately ablate code curation at the Multi-Step stage due to training resource limits. Within each block, scores below the corresponding ReAgent-8B are red, while scores that are unchanged or higher are green.

Variants	τ^2-Bench				ACEBench-Agent			BFCL-V4 Multi-Turn				
	Retail	Telecom	Airline	**Avg.**	Multi-Step	Multi-Turn	**Avg.**	Base	Miss-F.	Miss-P.	Long.	**Avg.**
Multi-Turn Stage												
ReAgent-8B (Multi-Turn)	65.6	63.4	48.5	59.1	96.3	65.8	81.0	70.5	51.1	37.1	44.5	50.8
w/o Conflict Scenarios	63.4	55.7	44.5	54.5	91.3	54.7	73.0	67.9	51.9	34.9	43.1	49.4
w/o LLM Rubrics	65.1	55.0	39.0	53.1	86.3	59.2	72.7	65.8	50.5	38.5	42.5	49.3
Multi-Step Stage												
ReAgent-8B (Multi-Step)	61.2	55.7	36.0	51.0	87.5	57.5	72.5	70.4	50.5	34.3	43.4	49.6
w/o Code Curation	55.9	45.6	29.0	43.5	71.3	50.0	60.6	63.8	44.6	28.9	41.1	44.6

4.4 Ablation Studies

Ablations on Training Stage. Table 3 reports performance after each cumulative training stage. Across both backbones, Multi-Step RL provides the first broad improvement, especially on stateful tool execution and multi-turn function calling. Continuing with Multi-Turn RL further improves the average performance on τ^2-Bench, ACEBench-Agent, and BFCL-V4 Multi-Turn. This progression is consistent with the intended curriculum, in which Multi-Step RL first builds a foundation of tool use, while Multi-Turn RL extends this foundation to user interaction and conflict resolution.

Ablations on training-scenario construction. As presented in Table 4, we first ablate the components that define Multi-Turn training scenarios. Removing conflict scenarios lowers all three benchmark averages, indicating that feasible scenarios alone do not provide sufficient coverage of environment-dependent user requests. Removing LLM-judged rubrics likewise lowers all three averages, showing that terminal-state functions alone do not fully supervise trajectory-dependent explanation, information acquisition, and valid resolution boundaries. We then separately ablate code curation at the Multi-Step stage. Its removal degrades every reported subset, supporting the need to repair recoverable code defects and filter candidates that remain unreliable after testing. Together, these ablations support complementary roles for all three components. Conflict scenarios broaden interaction coverage, LLM-judged rubrics supervise valid resolutions throughout the trajectory, and code curation improves the reliability of environments and reward evaluation.

4.5 Analysis

To examine how Multi-Turn RL develops conflict-resolution capabilities, Figure 3 analyzes two validation signals: (1) mean rewards over feasible and conflict subsets and (2) mean success rates of

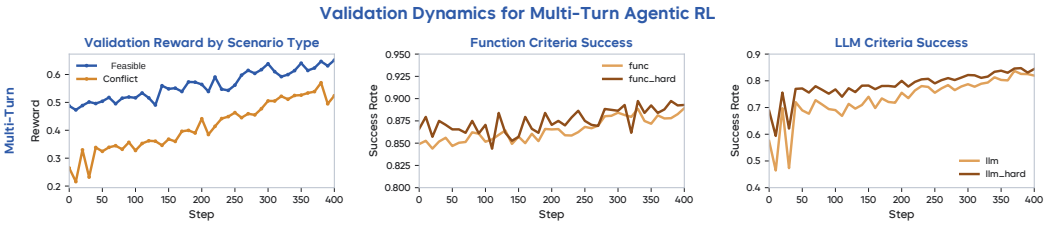


Figure 3: **Validation dynamics during Multi-Turn RL Training with ReAgent-8B.**

function-based and LLM-judged criteria. We note that, conflict reward increases more than feasible reward, substantially narrowing their gap. Having established a tool-use foundation during Multi-Step RL, the model is already strong at execution, so function-based criteria start high and improve only moderately; during Multi-Turn RL, it mainly learns trajectory-level conflict-resolution behaviors, as reflected in the larger gains of LLM-judged criteria. Together with the ablation study in Table 4, these results suggest that, beyond the executable tool-use capabilities established by Multi-Step RL, LLM-judged rubrics penalize trajectories that fail to follow valid resolutions, thereby strengthening explanation, information acquisition, and adherence to resolution boundaries.

5 Related Work

Environment and task synthesis. Agentic reinforcement learning depends on environments that support interaction and verifiable feedback (Zhang et al., 2026). Automated construction has therefore focused on expanding the available environment substrate (Froger et al., 2025; Li et al., 2026a; Xi et al., 2026). Recent work studies both LLM-simulated transitions (Li et al., 2026b; 2025; Chen et al., 2026c) and programmatic environments (Sullivan et al., 2025; Cai et al., 2025; Fang et al., 2026; Wang et al., 2026; Tu et al., 2026; Tian et al., 2026) for scalable environment synthesis. On top of these environments, task synthesis derives verifiable tasks, including those sampled from executable tool-call paths and dependency graphs (Xu et al., 2026; Dong et al., 2026), instantiated from reachable environment states (Song et al., 2026), or reverse-derived from executed traces (Chen et al., 2026a). Such requests are feasible by construction, leaving conflicts between user goals and the environment's state or constraints underrepresented. ReAgent builds on programmatic construction and makes this user–environment relation an explicit axis of training coverage.

User interaction and supervision. Training interactive agents requires both a user to drive interaction and proper supervision to evaluate the resulting behavior. As real users are hard to access at scale, most work relies on simulated users to sustain multi-turn interaction (Prabhakar et al., 2025; Zhao et al., 2025; Qian et al., 2025), treating them primarily as information providers. On the supervision side, rewards come mainly from executable checks by check functions on the terminal state (Song et al., 2026; Tu et al., 2026) or from rubric-based evaluation by llms (Gunjal et al., 2026; Shao et al., 2026). Yet both the simulated user and the reward criteria are largely built around already-feasible tasks, assuming that the request can be completed as stated. This leaves requests that require revision or must be blocked poorly represented. ReAgent instead grounds both components in conflicts and valid resolutions supported by the same environment instance, extending training coverage beyond feasible requests to blocked and partially resolvable ones.

6 Conclusion

In this work, we rethink how agent training data are constructed, identifying a construction-induced selection bias toward requests that are feasible as stated. To address this bias, we propose scenario co-design for constructing conflict scenarios, shifting the unit of construction from feasible tasks alone to the whole training scenario and jointly grounding the task, user, and reward in environment-supported conflicts and valid resolutions. ReAgent instantiates this principle, with experiments at two model scales on three benchmarks showing improvements in both conflict resolution and general agent performance. These findings support broadening training coverage within environments as a complement to scaling environments and feasible tasks.

AI USE STATEMENT

In this work, we used LLMs to synthesize training data, simulate users during agent rollouts and judge rubric-based reward criteria. The synthesized data were manually reviewed, and all AI-assisted language edits were checked by the authors. The authors take responsibility for the final content of this work.

ETHICS STATEMENT

ReAgent studies agents operating in synthetic programmatic environments and public benchmarks. The study does not involve human subjects, real users, or the collection of personal or sensitive data. All state changes and user behaviors in the reported experiments are simulated. The method is designed to train agents to identify environment-grounded conflicts, provide evidence-based explanations, guide users toward compliant alternatives, and avoid prohibited operations. Given this scope, we do not identify any specific ethical concern raised by the present study.

REPRODUCIBILITY STATEMENT

To support reproducibility, Section 3 specifies conflict scenario co-design, reward formulation, execution-guided code curation, and the two-stage RL curriculum. Appendix A documents environment and scenario construction, execution-based testing and repair, dataset composition, and full statistics. Appendix B.1 reports the training hyperparameters, model-selection procedure, rollout information flow, and reward evaluation, while Appendix B.2 specifies the benchmark implementations, sampling settings, and aggregate metrics. Appendix B.3 verifies alignment with the external BFCL-V4 evaluation, Appendix C provides detailed results and training dynamics, and Appendix D gives complete feasible and conflict scenario examples.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. In *Forty-third International Conference on Machine Learning*, 2026.

ByteDance Seed. Seed1.8 Model Card: Towards Generalized Real-World Agency. arXiv preprint arXiv:2603.20633, 2026.

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. AutoForge: Automated Environment Synthesis for Agentic Reinforcement Learning. arXiv preprint arXiv:2512.22857, 2025.

Aili Chen, Chi Zhang, Junteng Liu, Jiangjie Chen, Chengyu Du, Yunji Li, Ming Zhong, Qin Wang, Zhengmao Zhu, Jiayuan Song, Ke Ji, Junxian He, Pengyu Zhao, and Yanghua Xiao. DIVE: Scaling Diversity in Agentic Task Synthesis for Generalizable Tool Use. In *Forty-third International Conference on Machine Learning*, 2026a.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzhi, and Wu Liu. ACEBench: A Comprehensive Evaluation of LLM Tool Usage. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12970–12998. Association for Computational Linguistics, November 2025.

Liuji Chen, Dianxing Tang, Xing Shi, Dingshuo Chen, Qiang Liu, Shu Wu, and Liang Wang. Learning when not to act: Mitigating tool abuse in agentic reinforcement learning, 2026b.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Xuefei Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason E Weston, and Dat Huynh. Scaling Agent Learning via Experience Synthesis. In *The Fourteenth International Conference on Learning Representations*, 2026c.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-World: Scaling Real-World Environment Synthesis for Evolving General Agent Intelligence. arXiv preprint arXiv:2604.18292, 2026.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Ningyu Zhang, Fei Huang, Wentao Zhang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17610–17621, San Diego, California, United States, July 2026. Association for Computational Linguistics.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. ARE: Scaling Up Agent Environments and Evaluations. arXiv preprint arXiv:2509.17158, 2025.

Tianhong Gao, Jundong Shen, Jiapeng Wang, Bei Shi, Ying Ju, Junfeng Yao, and Huiyu Yu. Benchmarking and learning real-world customer service dialogue, 2026.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean M. Hendryx. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jiachun Li, Zhuoran Jin, Tianyi Men, Yupu Hao, Kejian Zhu, Lingshuai Wang, Dongqi Huang, Longxiang Wang, Shengjia Hua, Lu Wang, Jinshan Gao, Hongbang Yuan, Ruilin Xu, Kang Liu, and Jun Zhao. Agentic Environment Engineering for Large Language Models: A Survey of Environment Modeling, Synthesis, Evaluation, and Application. arXiv preprint arXiv:2606.12191, 2026a.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116. Association for Computational Linguistics, December 2023.

Yixia Li, Hongru Wang, Jiahao Qiu, Zhenfei Yin, Dongdong Zhang, Cheng Qian, Zeping Li, Xiaoteng Ma, Guanhua Chen, and Heng Ji. From Word to World: Can Large Language Models be Implicit Text-based World Models? In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, July 2026b.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. Simulating Environments with Reasoning Models for Agent Training. arXiv preprint arXiv:2511.01824, 2025.

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, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. ToolACE: Winning the Points of LLM Function Calling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=8EB8k6DdCU.

Yuanjie Lyu, Chengyu Wang, Haonan Zheng, Yuanhao Yue, Junbing Yan, Ming Wang, and Jun Huang. AgenticQwen: Training Small Agentic Language Models with Dual Data Flywheels for Industrial-Scale Tool Use. arXiv preprint arXiv:2604.21590, 2026.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Forty-second International Conference on Machine Learning*, 2025.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Quoc Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. APIGen-MT: Agentic Pipeline for Multi-Turn Data Generation via Simulated Agent-Human Interplay. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. UserBench: An Interactive Gym Environment for User-Centric Agents. arXiv preprint arXiv:2507.22034, 2025.

Qwen Team. Qwen3.6-27B: Flagship-Level Coding in a 27B Dense Model, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b.

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, Hannaneh Hajishirzi, and Pang Wei Koh. DR Tulu: Reinforcement Learning with Evolving Rubrics for Deep Research. In *Forty-third International Conference on Machine Learning*, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. arXiv preprint arXiv:2402.03300, 2024.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling Tool-Interactive Environments for LLM Agent via Programmatic Synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*. Association for Computational Linguistics, 2026.

Michael Sullivan, Mareike Hartmann, and Alexander Koller. Procedural Environment Generation for Tool-Use Agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, November 2025.

Xiaoyu Tian, Haotian Wang, Shuaiting Chen, Hao Zhou, Kaichi Yu, Yudian Zhang, Jade Ouyang, Junxi Yin, Jiong Chen, Baoyan Guo, Lei Zhang, Junjie Tao, Yuansheng Song, Ming Cui, and Chengwei Liu. ASTRA: Automated Synthesis of agentic Trajectories and Reinforcement Arenas. arXiv preprint arXiv:2601.21558, 2026.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, Qi GU, Hui Su, and Xunliang Cai. ScaleEnv: Scaling Environment Synthesis from Scratch for Generalist Interactive Tool-Use Agent Training. In *Forty-third International Conference on Machine Learning*, 2026.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent World Model: Infinity Synthetic Environments for Agentic Reinforcement Learning. In *Forty-third International Conference on Machine Learning*, 2026.

Ziqiao Xi, Shuang Liang, Qi Liu, Jiaqing Zhang, Letian Peng, Fang Nan, Meshal Nayim, Tianhui Zhang, Rishika Mundada, Lianhui Qin, Biwei Huang, and Kun Zhou. C-World: A Computer Use Agent Environment Creator. arXiv preprint arXiv:2601.06328, 2026.

Minrui Xu, Zilin Wang, Mengyi DENG, Zhiwei Li, Zhicheng Yang, Xiao Zhu, Yinhong Liu, Boyu Zhu, Baiyu Huang, Chao Chen, Heyuan Deng, Fei Mi, Lifeng Shang, Xingshan Zeng, and Zhijiang Guo. EnvFactory: Scaling Tool-Use Agents via Executable Environments Synthesis and Robust RL. arXiv preprint arXiv:2605.18703, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 Technical Report. arXiv preprint arXiv:2505.09388, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations*, 2025.

Guibin Zhang, Hejia Geng, Xiaohang Yu, Zhenfei Yin, Zaibin Zhang, Zelin Tan, Heng Zhou, Zhong-Zhi Li, Xiangyuan Xue, Yijiang Li, Yifan Zhou, Yang Chen, Chen Zhang, Yutao Fan, Zihu Wang, Songtao Huang, Francisco Piedrahita Velez, Yue Liao, Hongru WANG, Mengyue Yang, Heng Ji, Jun Wang, Shuicheng YAN, Philip Torr, and LEI BAI. The Landscape of Agentic Reinforcement Learning for LLMs: A Survey. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856.

Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao Zhai, and Xunliang Cai. MUA-RL: Multi-turn User-interacting Agent Reinforcement Learning for agentic tool use. arXiv preprint arXiv:2508.18669, 2025.

A Construction and Code Curation Details

Section 3 presents ReAgent's scenario co-design and code curation at a high level. This section provides the corresponding construction details, following the pipeline from environment construction and curation through feasible scenario construction and conflict scenario co-design to reward curation and final dataset composition. The organization follows the order in which executable environments and training scenarios are constructed and filtered.

A.1 Environment Construction and Curation

ReAgent first constructs and verifies the programmatic environments that provide the execution grounding for subsequent scenario construction. Environment tools determine the state facts, state transitions, and terminal outcomes on which task feasibility, conflicts, valid resolutions, and executable reward criteria are defined. We therefore repair tool-level execution failures and filter environments through stateful tool-chain testing before using them to construct training scenarios.

Environment discovery and implementation. Following the environment construction summarized in Section 3 and the representation in Equation 1, we provide the screening and implementation details here.

- **Source task collection and environment discovery.** Following EnvScaler (Song et al., 2026), we use task descriptions from the Level 1, Level 2, and Level 3 training splits of API-Bank (Li et al., 2023) and from ToolACE (Liu et al., 2025) as source materials for environment discovery. We first filter non-English and multimodal requests, together with requests containing external links, contact identifiers, or other predefined exclusion patterns. Within each source, we remove exact duplicate task strings before merging the two collections. An LLM[1] then retains tasks that require interaction with a persistent, domain-specific service supporting both state queries and state-changing operations. For each retained task, a second LLM infers a task-independent environment description and filters the descriptions according to their programmatic modelability and general usefulness. We remove exact duplicate environment summaries, encode the remaining descriptions with `all-MiniLM-L6-v2`[2], and apply K-means clustering, retaining the candidate nearest to each cluster centroid for implementation. The source tasks seed environment discovery and implementation, while the final training scenarios are constructed separately from the verified executable environments.
- **Environment implementation.** Conditioned on each retained environment description and its source material, an LLM plans the environment skeleton, supported tool operations, and a structured set of environment-specific constraints, denoted by Φ_E, that govern the validity of agent operations. The retained environment description and Φ_E together constitute the environment documentation E_{doc}. The environment state plan specifies the entities, attributes, and relations maintained by the environment, while the operation plan specifies how this state can be queried or updated. We synthesize the state representation and tool implementations as a Python class, whose attributes store the environment state and whose methods implement the planned operations. The resulting class forms F_{exec}, while the method names, arguments, and descriptions form Σ_{tool}. Programs that cannot be assembled into syntactically valid Python or parsed into the required class structure and tool metadata are discarded.

Tool-level execution repair and stateful tool-chain testing. After syntax validation, each generated environment undergoes targeted tool-level execution repair and stateful tool-chain testing.

- **Tool-level execution repair.** For each generated environment, we generate a temporary test initialization configuration and instantiate the environment for code curation. This temporary configuration is used only to exercise and repair the tools and is distinct from the scenario-specific initialization s_0. We then apply a ReAct-guided repair loop (Yao et al., 2023) to each tool. The loop terminates when both probes pass or after five repair rounds.

[1] Unless otherwise specified, all LLM-based components in this section use Qwen3.6-27B (Qwen Team, 2026). We use sampling temperatures of 0.6, 0.2, and 0.1 for package generation, cross-component reflection, and scoring, respectively.

[2] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Algorithm A.1: Tool-Level Execution Repair

INPUT. A generated environment with temporary test initialization, and a target tool.

PROBE. An LLM constructs a positive call with valid, state-consistent arguments and a negative call with invalid or boundary arguments. We execute each call and record its arguments, return value, runtime behavior, and the environment states before and after execution.

DIAGNOSE. When a tool call does not produce the expected outcome, an LLM analyzes the collected execution evidence and attributes the failure to one of four sources: `[TEST_CASE]`, `[TOOL_IMPLEMENTATION]`, `[CLASS_DEFINITION]`, or `[STATE_COVERAGE]`.

REPAIR. Based on the diagnosis, an LLM regenerates the test case, updates the temporary test initialization configuration, or repairs the relevant code. When the temporary configuration or code changes, we first reassemble and instantiate the complete test environment. After every repair, we rerun both the positive and negative probes.

OUTPUT. A tool that passes both probes within five repair rounds, or an unresolved failure that causes the environment to be discarded.

- **Stateful tool-chain testing and filtering.** Environments with unresolved tool-level failures are discarded. For each remaining environment, a testing agent generates a sequence of tool calls within the same test environment instance over 30 steps. At each step, it selects the tool and arguments based on the available interfaces, the current environment state, and the coverage of previous tests. Each call is executed against the current state, and any resulting state update is preserved for subsequent calls. A checking agent examines the tool implementation, call arguments, return value, and the environment states before and after execution. It labels each call as `[pass]`, `[warning]`, or `[fail]`. A warning indicates functionally valid behavior with a minor robustness issue, which is also judged acceptable. We retain environments whose combined proportion of `[pass]` and `[warning]` calls reaches a threshold γ[3].

The remaining environment skeletons form the retained pool $\mathcal{E}_{\text{ret}}$ under our execution-testing protocol. These environments provide the executable state, transition logic, and environment-specific constraints used to construct feasible scenarios and co-design conflict scenarios.

A.2 Training Scenario Construction

Following the training-scenario formulation in Section 2, each skeleton in $\mathcal{E}_{\text{ret}}$ defines the executable operations, transition logic, and environment-specific constraints available to subsequent scenarios. Only after an environment is retained do we generate its scenario-specific initialization configuration s_0 and instantiate $e = \langle \mathcal{E}, s_0 \rangle$. Construction then proceeds along two branches. The feasible branch constructs a task whose requested outcome is compatible with the environment. The conflict branch first identifies candidate conflicts and valid resolutions from the same environment instance and uses them as the common basis for scenario co-design.

A.2.1 Feasible Scenario Construction

Task and function-criterion construction. Following EnvScaler (Song et al., 2026), an LLM constructs a task d from the environment instance e and derives its terminal-state function criteria

$$e \longrightarrow d \longrightarrow \rho_{\mathcal{X}}^{\text{func}}. \tag{6}$$

The task records the complete objective, target entities, and required terminal outcome. We require it to be feasible under the initial environment state s_0 and the environment-specific constraints Φ_E, to involve dependent query and update operations, and to yield conditions that can be verified programmatically. An LLM decomposes these conditions into a checklist and implements each item as an executable check function over the terminal state. The resulting functions form $\rho_{\mathcal{X}}^{\text{func}}$.

Task rewriting and user specification construction. We rewrite the task d into the agent-facing task request and construct the user specification required by each interaction setting.

[3] We use $\gamma = 0.85$ in this work.

- For Multi-Step scenarios, we rewrite d as a concise, self-contained task request q^{MS}. The rewrite removes tool names, internal fields, and prescribed operation sequences while preserving all information that must come from the user. We set $p_u = \varnothing$.
- For Multi-Turn scenarios, we jointly generate a concise initial task request q^{MT} and a user specification p_u. The request may leave user-held information implicit or user choices unresolved. The user specification records the user's goal, available information, disclosure behavior, and stopping condition. A reflection step checks that q^{MT} and p_u preserve the objective specified in d.

This construction yields

$$\mathcal{X}^{\mathrm{feas}}_{\mathrm{MS}} = \left\langle e, q^{\mathrm{MS}}, \varnothing, \rho^{\mathrm{func}}_{\mathcal{X}} \right\rangle, \qquad \mathcal{X}^{\mathrm{feas}}_{\mathrm{MT}} = \left\langle e, q^{\mathrm{MT}}, p_u, \rho^{\mathrm{func}}_{\mathcal{X}} \right\rangle. \tag{7}$$

After reward curation in Section A.2.3, the retained scenarios form the Multi-Step set $\mathcal{D}_{\mathrm{MS}}$ and the feasible Multi-Turn set $\mathcal{D}^{\mathrm{feas}}_{\mathrm{MT}}$.

A.2.2 Conflict Scenario Co-Design

Conflict-and-resolution specification. Following Section 3.1, we distinguish two sources of incompatibility between a requested action or outcome and an instantiated environment e.

- **Constraint-based conflict.** A requested action or outcome violates an environment-specific constraint in Φ_E. For example, a user asks to book a flight for tonight, although an environment-specific constraint prohibits same-day bookings.
- **State-based conflict.** A request condition or user assumption is inconsistent with the initial environment state s_0. For example, a user requests an operation under the assumption that an account balance is \$1,000 when the initial balance is \$100.

A conflict scenario may contain multiple constraint-based or state-based conflicts, or both. A valid resolution specifies how the agent can respond while respecting the environment-specific constraints and the state facts. Depending on the selected conflicts, it may require completing the feasible parts of the request, guiding the user toward a compliant revision, or refusing the infeasible part when no compliant revision is available.

Jointly construct the task, playbook, and oracle. Following Equation 3, scenario co-design starts from conflicts supported by the environment instance and their valid resolutions. Given an environment instance e, an LLM first identifies candidate conflicts and valid resolutions from the initial environment state and environment-specific constraints. Using them as a common basis, it jointly constructs the Task–Playbook–Oracle package $\mathcal{P}_{\mathcal{X}} = \langle d, p, \omega \rangle$. The three components serve distinct roles while remaining grounded in the same selected conflicts and valid resolutions.

- **Task d.** The task states the original request, including the user's goal and requested outcome. It provides the source specification from which the agent-facing task request q is derived.
- **User playbook p.** The playbook instantiates the user specification p_u for the scenario. It records the user's information, disclosure behavior, revision conditions, and stopping condition, thereby governing how the simulated user responds to the agent.
- **Structured oracle ω.** The oracle organizes conflict-resolution supervision into six fields: `immutable_facts`, `environment_constraints`, `guidance_boundaries`, `refusal_boundaries`, `required_agent_behaviors`, and `non_negotiable_checks`. The first two record the state facts and applicable environment-specific constraints used to establish the selected conflicts. The guidance and refusal boundaries specify what the agent should and should not do when guiding or refusing the user. The required agent behaviors specify the key actions that the agent must query, execute, guide, or refuse, while the non-negotiable checks collect these mandatory requirements into an checklist.

Together, the task expresses the request, the playbook governs user behavior, and the oracle records the conflict facts and valid resolution boundaries. Their common grounding keeps these components semantically coherent without prescribing a single tool-use or dialogue path. Appendix D.2 provides a complete example.

Cross-component reflection and filtering. The Task–Playbook–Oracle package undergoes cross-component reflection and filtering before the task request and reward criteria are derived.

Algorithm A.2: Conflict Scenario Co-Design

INPUT. An execution-verified environment instance e.

GROUND. Identify candidate conflicts from s_0 and Φ_E, together with valid resolutions that respect the corresponding state facts and constraints.

CONSTRUCT. Jointly construct $\mathcal{P}_{\mathcal{X}} = \langle d, p, \omega \rangle$ from the selected conflicts and valid resolutions.

REFLECT. Compare $\mathcal{P}_{\mathcal{X}}$ with e and check whether the request in d, the user response conditions in p, and the conflict facts and resolution boundaries in ω remain consistent with the same selected conflicts and valid resolutions. Jointly revise the components when an inconsistency is identified.

FILTER. A separate LLM evaluates the revised package on seven binary dimensions covering task design, environment consistency, concealment of the selected conflicts in the user request represented by d, reasoning complexity, interaction design, oracle design, and evaluation value. Reject any package that fails a check.

DERIVE. From each retained package, derive the agent-facing task request q from d and construct the reward-criterion set $\rho_{\mathcal{X}}$.

OUTPUT. A candidate conflict scenario whose executable reward criteria proceed to reward curation.

Task request derivation. From each retained Task–Playbook–Oracle package, an LLM rewrites d as a concise agent-facing task request q. The rewrite preserves the original user goal and target entities while removing tool names, implementation details, tool-retrievable values, and prescribed operation sequences. It also avoids exposing the selected conflicts and valid resolutions recorded elsewhere in the package. A reflection step checks that q preserves the intended request without revealing this hidden construction context.

Reward criteria derivation. From the same retained package, an LLM constructs criteria for terminal outcomes and trajectory-dependent behavior, including explanation, guidance, and completion of the remaining feasible work. Following the four-way taxonomy in Equation 4, these criteria form the scenario-specific reward-criterion set $\rho_{\mathcal{X}}$.

- **Function criteria.** Criteria evaluated from the terminal environment state are implemented as executable check functions and tagged as `function` or `function_hard`. A failed `function_hard` criterion sets the complete reward to zero.
- **LLM criteria.** Criteria evaluated from the interaction trajectory are expressed as LLM-judged rubrics. They are tagged as `llm` or `llm_hard` according to whether their violation constitutes a hard failure.

In addition to the criteria synthesized from the package, we directly convert the conflict-resolution supervision encoded in the structured oracle into `llm_hard` rubrics. Concretely, this supervision is written through the `immutable_facts`, `environment_constraints`, `guidance_boundaries`, `refusal_boundaries`, and `non_negotiable_checks` keys. The resulting hard trajectory rubrics evaluate the conflict-specific facts, environment-specific constraints, and valid agent resolution boundaries recorded by these oracle keys. The remaining `required_agent_behaviors` key is not directly converted, because it serves as an intermediate behavioral specification for synthesizing scenario-specific criteria; its mandatory requirements are already reflected in the derived terminal and trajectory criteria, especially the `non_negotiable_checks`.

Scenario assembly. The playbook p instantiates the user specification p_u for the conflict scenario. The derived components therefore define

$$\mathcal{X}_{\mathrm{MT}}^{\mathrm{conf}} = \langle e, q, p_u, \rho_{\mathcal{X}} \rangle . \tag{8}$$

Table 5: **Statistics of the execution-verified environments and constructed training scenarios.**

Assets	Initial Candidates	Retained	Environments	Function Criteria Per Scenario		LLM Criteria Per Scenario	
				Regular	Hard	Regular	Hard
Environment	3,039	2,415	-	-	-	-	-
Multi-Step	-	4,513	2,360	10.12	-	-	-
Feasible Multi-Step	4,830	4,513	2,360	10.12	-	-	-
Multi-Turn	-	4,612	2,409	7.06	0.36	2.39	2.80
Feasible Multi-Turn	2,415	2,360	2,360	10.28	-	-	-
Conflict Multi-Turn	2,415	2,252	2,252	3.69	0.74	4.89	5.73

During rollout, the user simulator conditions on the realized interaction history and on p_u, as specified in Equation 10. After reward curation in Section A.2.3, the retained scenarios form the conflict Multi-Turn set $\mathcal{D}_{\mathrm{MT}}^{\mathrm{conf}}$.

A.2.3 Reward Curation

As summarized in Section 3.2, we test and repair every executable reward criterion generated in the feasible and conflict branches before forming the training sets. Specifically, we test all functions in $\rho_{\mathcal{X}}^{\mathrm{func}}$ and $\rho_{\mathcal{X}}^{\text{func-hard}}$. Each function is evaluated on generated positive and negative terminal states for its stated criterion. Trajectory-level rubrics have no executable check function and are evaluated during rollout. A scenario is retained only when every executable criterion passes both tests within the repair budget.

Algorithm A.3: Reward-Function Curation

Input. A candidate reward function and its stated criterion.

Probe. For each candidate reward function, an LLM constructs a positive terminal state that satisfies the intended criterion and a negative terminal state that directly violates it. We execute the function on both states and require `[True]` for the positive state and `[False]` for the negative state.

Diagnose. When either execution produces an unexpected outcome, an LLM analyzes the criterion, reward-function code, test states, and execution results. It assigns the failure to `[CHECKER_CODE]` or `[TEST_STATES]`.

Repair. The LLM repairs the reward function or regenerates the test states and then reruns both executions. The loop stops when both tests pass or after three repair rounds.

Output. A reward function that passes both tests, or a failed function that causes the scenario to be discarded.

A.2.4 Dataset Composition

After reward curation, the feasible scenarios form $\mathcal{D}_{\mathrm{MS}}$ and $\mathcal{D}_{\mathrm{MT}}^{\mathrm{feas}}$, while the conflict scenarios form $\mathcal{D}_{\mathrm{MT}}^{\mathrm{conf}}$. The two Multi-Turn scenario sets constitute

$$\mathcal{D}_{\mathrm{MT}} = \mathcal{D}_{\mathrm{MT}}^{\mathrm{feas}} \cup \mathcal{D}_{\mathrm{MT}}^{\mathrm{conf}}. \tag{9}$$

Because the two construction branches are filtered independently, a retained environment need not contribute a retained scenario to every set. Table 5 reports the detailed statistics and the number of distinct environments represented in each retained set. Together, stateful tool-chain testing, cross-component reflection, and reward-function curation filter the constructed scenarios for semantic coherence and executable correctness. The resulting Multi-Step and Multi-Turn datasets form the training data used by the two-stage curriculum detailed in Appendix B.

B Training and Evaluation Details

Section 4.1 summarizes the training and evaluation setup used for the main results. This section specifies the training schedule, model-selection procedure, rollout information flow, reward eval-

Table 6: **Reported training hyperparameters.** Slash-separated values are listed as Multi-Step / Multi-Turn.

Setting	Qwen3-8B	Qwen3-30B-A3B-Thinking-2507
SFT epochs	3	–
SFT learning rate	10^{-5}	–
SFT maximum sequence length	32K	–
RL epochs per stage	10	10
RL learning rate	10^{-6}	10^{-6}
RL batch size	64 / 32	64 / 64
Rollouts per prompt	16 / 8	16 / 8
RL maximum prompt length	10,240	10,240
RL maximum response length	16,384 / 32,768	24,576 / 40,960

uation, and benchmark configurations. It also compares our BFCL-V4 implementation with the external results reported by Agent-World.

B.1 Training Configuration

From the 4,513 Multi-Step scenarios and 4,612 Multi-Turn scenarios, we randomly hold out 100 scenarios from each stage for validation. The Multi-Turn pool contains 2,360 feasible and 2,252 conflict scenarios. For Qwen3-8B, cold-start SFT uses 10,000 trajectories from feasible scenarios, divided evenly between Multi-Step and Multi-Turn trajectories. These SFT trajectories are generated by Qwen3-30B-A3B-Thinking-2507. Multi-Step and Multi-Turn RL each run for ten epochs with GRPO at a learning rate of 1×10^{-6}. We use neither a KL penalty in the reward nor a KL loss in the actor objective. Thinking mode is enabled for both backbones throughout training and evaluation. Multi-Turn RL starts from the Multi-Step checkpoint with the highest validation reward, and the final model is selected by the highest Multi-Turn validation reward. During training, user simulation and rubric judging use Qwen3.6-27B (Qwen Team, 2026), both with a sampling temperature of 0.6.

Motivation for the stage order. In early experiments that started directly with Multi-Turn RL, the agent rapidly relied on the user simulator for task-step-level guidance. This observation motivates first building a foundation of tool use before opening the additional user information channel. The stage order also postpones user-simulation overhead to Multi-Turn training.

Information access during rollout. At turn t, the agent conditions on the documentation, interfaces, and history $h_t = (q, a_0, o_1, \ldots, a_{t-1}, o_t)$. Tool actions return observations and may update s_t. In a conflict scenario, after the agent emits a response action a_t, the simulator generates the next user message according to

$$o^u_{t+1} \sim \pi_u(\cdot \mid h_t, a_t, p). \tag{10}$$

For a feasible Multi-Turn scenario, the same process uses the user specification p_u in place of the playbook p. In both cases, the simulator conditions each response on the realized dialogue and tool-use history. The corresponding user specification determines which information the user holds, how that information is disclosed, how the user responds to the agent's behavior, and when the interaction stops. The playbook p instantiates p_u in conflict scenarios and is provided only to the user simulator. The task d, structured oracle ω, and reward criteria remain hidden from both the agent and the user simulator and are used only for evaluation.

Reward criteria. Function criteria evaluate conditions on the terminal state s_T, whereas LLM criteria evaluate trajectory-dependent requirements with the task d and corresponding rubric as evaluator context. A failed hard criterion sets the complete reward to zero; otherwise, the reward averages all criterion outcomes and subtracts the repetition penalty, as defined in Equation 5. Specifically, $P_{\text{dup}}(\tau) = \min(0.02 N_{\text{dup}}(\tau), 0.1)$, where $N_{\text{dup}}(\tau)$ counts the distinct $(\texttt{tool_name}, \texttt{arguments})$ pairs that occur more than once in τ.

Table 7: **Alignment between our evaluation and Agent-World.** We compare independently obtained results for three shared open-source baselines. Detailed subset scores are reported for our evaluation (denoted as Ours), while Agent-World provides only category-level averages (denoted as AW). KV denotes key–value retrieval. For scoring categories, paired results with an absolute difference of at most 2.0 percentage points are shaded in green, while larger differences are in red.

	BFCL-V4																			
	Multi-Turn					Memory				Live					No-Live					Irrel.	Avg.
	Base	Miss-F.	Miss-P.	Long.	**Avg.**	KV	Vec.	Rec.	**Avg.**	Sim.	Mul.	Para.	Mul-P.	**Avg.**	Sim.	Mul.	Para.	Mul-P.	**Avg.**		
Qwen3-8B																					
AW	–	–	–	–	35.4	–	–	–	17.6	–	–	–	–	80.9	–	–	–	–	90.2	77.2	48.7
Ours	50.3	38.8	35.8	28.8	38.4	6.8	15.3	36.3	19.5	83.5	80.7	81.3	79.2	81.2	75.2	95.9	94.6	90.1	89.0	78.1	50.3
EnvScaler-8B																					
AW	–	–	–	–	47.1	–	–	–	21.9	–	–	–	–	82.2	–	–	–	–	88.5	74.6	53.8
Ours	65.1	47.4	39.8	40.9	48.3	9.8	14.8	43.2	22.6	84.9	81.8	84.4	75.0	82.3	74.7	95.3	93.4	89.0	88.1	76.5	54.6
AWM-8B																					
AW	–	–	–	–	34.9	–	–	–	15.7	–	–	–	–	80.5	–	–	–	–	90.2	73.9	47.6
Ours	49.1	38.5	34.5	30.8	38.2	5.0	12.7	31.5	16.4	84.6	79.8	81.3	83.3	80.8	75.1	95.4	94.8	90.0	88.8	75.9	49.1

B.2 Evaluation Protocol

Following the benchmark selection in Section 4.1, we use the same benchmark implementations as for ReAgent for all models evaluated by us. The benchmark-specific configurations and aggregate metrics are given below.

τ^2-Bench. We evaluate the base split of the Retail, Telecom, and Airline domains at temperature 0.6, using Seed1.8 (ByteDance Seed, 2026) as the user simulator. We expose the native domain tools through an MCP-style adapter. The agent first retrieves the available tool specifications through `list_tools` and then invokes a selected domain tool through `call_tool`; the adapter forwards the call to the underlying τ^2-Bench environment. For Telecom, we add an execution-boundary clarification to distinguish assistant-accessible account and service tools from phone-side diagnostic actions. The latter must be requested from the simulated user unless they are explicitly included in the available tool list. We compute the overall score as the average across the three domains:

$$A_{\tau^2} = \frac{A_{\text{Retail}} + A_{\text{Telecom}} + A_{\text{Airline}}}{3}. \tag{11}$$

τ-Bench. For the additional request-type analysis on the original τ-Bench in Table 8, we evaluate the Retail and Airline domains with the same user-simulation setup as τ^2-Bench, using Seed1.8 as the user simulator and reporting Avg@4. Request-type overall scores are computed as Micro Means over all evaluated rollouts in the corresponding request type.

ACEBench-Agent. We evaluate the Multi-Step and Multi-Turn subsets using the public EnvScaler evaluation implementation [4]. We use a sampling temperature of 0.7. The reported overall score assigns equal weight to the two subsets:

$$A_{\text{ACE}} = \frac{A_{\text{Multi-Step}} + A_{\text{Multi-Turn}}}{2}. \tag{12}$$

BFCL-V4. We use the official BFCL-V4 evaluation implementation with a sampling temperature of 0.7. Consistent with Table 1, we report Memory, Multi-Turn, No-Live, Live, Relevance, Irrelevance, and the BFCL-V4 aggregate. We omit WebSearch because its execution requires access to the paid Google Search application programming interface. Following the category weighting used in the official evaluation rule [5] and excluding the weight of WebSearch, we calculate the reported

[4] https://github.com/RUC-NLPIR/EnvScaler/tree/main/interact_with_env

[5] https://github.com/ShishirPatil/gorilla/blob/main/berkeley-function-call-leaderboard/bfcl_eval/eval_checker/eval_runner_helper.py

aggregate as

$$A_{\text{BFCL}} = \frac{0.3A_{\text{Multi-Turn}} + 0.2A_{\text{Memory}} + 0.1A_{\text{NoLive}} + 0.1A_{\text{Live}} + 0.1A_{\text{Irrelevance}}}{0.8}. \quad (13)$$

B.3 Evaluation Alignment with Agent-World

To examine whether our benchmark results are aligned with those reported by Agent-World, we independently evaluate three shared open-source baselines on BFCL-V4. As shown in Table 7, the two evaluations produce closely matched category-level and overall results. All reported aggregate scores differ by at most 2.0 percentage points, except for the Multi-Turn results of Qwen3-8B and AWM-8B, while the differences in the final average remain within 1.6 points for all three models. This comparison indicates that the independently implemented evaluation pipelines yield results broadly consistent with those from Agent-World.

Together, these configurations specify the training, model selection, reward evaluation, and benchmark implementations used for the reported results. The BFCL-V4 comparison shows that our independently implemented evaluation pipeline yields results broadly consistent with those reported by Agent-World.

C Additional Experiments and Analysis

Beyond the experiments reported in Section 4, this section presents additional analyses: request-type results on τ-Bench and the training dynamics of the two-stage curriculum.

C.1 Additional Request-Type Results on τ-Bench

The request-type results in Table 2 are based on a manual partition of the τ^2-Bench requests according to the valid resolution under each task's environment state and applicable constraints. FEASIBLE requests can be completed as stated, BLOCKED requests have a prohibited core goal, and PARTIAL requests require rejecting an invalid part, guiding the user toward a compliant alternative, or completing only the feasible remainder. This partition is rule-based: requests whose valid resolution is full completion or full blocking are assigned directly, while the remaining cases are labeled PARTIAL when valid handling requires partial completion or user guidance; the authors manually verified the consistency of the resulting labels.

Using the same annotation criterion, we additionally partition the original τ-Bench (Yao et al., 2025), which likewise evaluates stateful service interactions governed by domain policies. Table 8 reports the results on its Retail and Airline domains. Notably, ReAgent improves over the corresponding baselines in Overall at both scales: ReAgent-8B exceeds the strongest 8B baseline by 4.9%, 17.3%, and 27.2% on FEASIBLE, BLOCKED, and PARTIAL, respectively, while ReAgent-30B-A3B exceeds the original Qwen3-30B-A3B-Thinking-2507 by 2.1%, 22.6%, and 11.0%. The gains on blocked requests are accompanied by improvements on feasible and partially resolvable requests, providing further evidence that scenario co-design broadens coverage beyond requests that are feasible as stated rather than merely encouraging refusal.

C.2 Training and Validation Dynamics

Training dynamics. Figure 4 shows the training dynamics of the two-stage RL curriculum introduced in Section 3.3. In the Multi-Step stage, validation reward increases overall, while the average number of turns reaches about 25.9 by the end of training. The agent therefore continues to use extended tool use as performance improves. After switching to Multi-Turn training, validation reward continues to improve to about 0.60 by step 400, while the average turn count remains around 30. The entropy curves remain stable in both stages. These dynamics follow the intended progression from a foundation of tool use to user interaction and conflict resolution.

Together, these experiments extend the main results with request-type evidence on τ-Bench and the training dynamics of the two-stage curriculum.

Table 8: **Performance across request types on τ-Bench (Avg@4, %).** All results are obtained using our evaluation setup. In Overall, underlining marks the strongest baseline for each request type within each model-scale block; bold shaded entries are ReAgent scores, with relative gains over the underlined baselines shown in dark-orange.

Models	Retail			Airline			Overall (Micro Mean)		
	FEASIBLE $N=73$	BLOCKED $N=4$	PARTIAL $N=38$	FEASIBLE $N=20$	BLOCKED $N=19$	PARTIAL $N=11$	FEASIBLE $N=93$	BLOCKED $N=23$	PARTIAL $N=49$
Qwen3-8B	57.7	62.5	46.1	18.8	40.8	9.1	49.3	44.6	37.8
AgenticQwen-8B	53.6	87.5	58.4	18.8	39.5	8.0	46.1	47.3	47.1
EnvFactory-8B	52.9	62.5	53.3	24.1	44.7	11.4	46.7	47.8	43.9
EnvScaler-8B	55.0	56.3	58.9	26.3	56.6	9.1	48.8	56.5	47.7
AWM-8B	57.5	68.8	45.3	15.0	42.1	4.6	48.4	46.7	36.2
ReAgent-8B	56.9	93.8	71.7	32.9	60.5	22.7	**51.7**$^{+4.9\%}$	**66.3**$^{+17.3\%}$	**60.7**$^{+27.2\%}$
Qwen3-30B-A3B-Thinking-2507	60.1	56.3	73.7	47.4	57.9	16.3	57.4	57.6	60.8
ReAgent-30B-A3B	64.3	68.8	80.9	38.0	71.1	20.9	**58.6**$^{+2.1\%}$	**70.7**$^{+22.6\%}$	**67.5**$^{+11.0\%}$

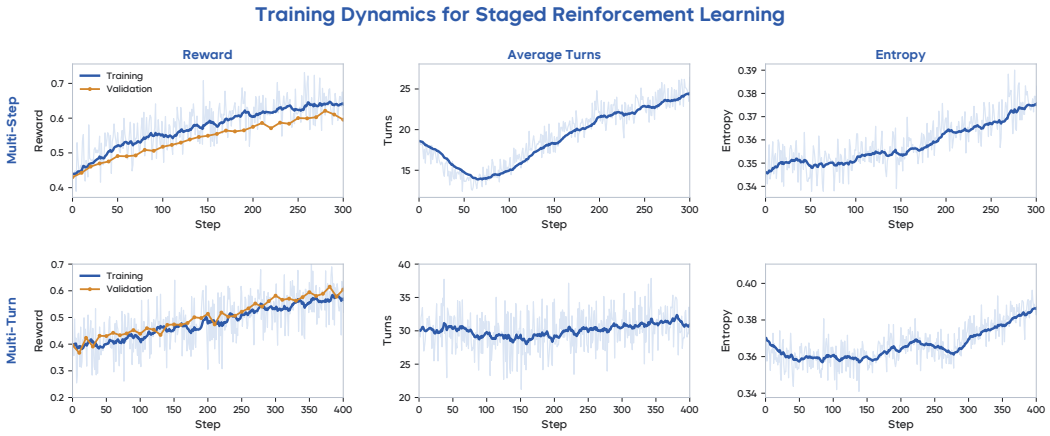
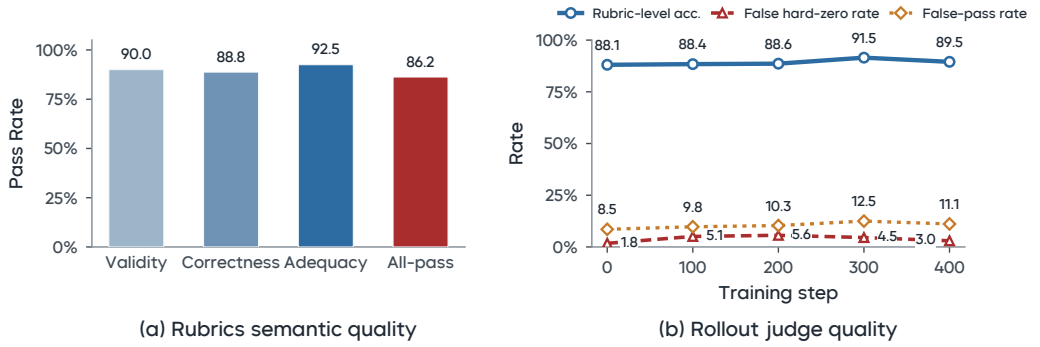


Figure 4: **Training dynamics of the two-stage RL curriculum with Qwen3-8B.**

C.3 Semantic Validation of Rubrics and Judges

We conduct an additional semantic check of the supervision signals in the constructed data. The checks in this section are performed with `gpt-5.6-sol`. We additionally manually inspected 10% of judgments and verified that the `gpt-5.6-sol` review decisions were almost always consistent with the authors' judgments.

Semantic quality of synthesized rubrics. To check the semantic quality of rubrics synthesized for conflict scenarios, we randomly sample 80 scenarios from the synthesized conflict-scenario pool and evaluate them along three dimensions. *Validity* checks whether the conflict is genuine and supported by the environment state or constraints. *Correctness* checks whether the rubrics faithfully express the oracle and environment constraints. *Adequacy* checks whether the rubrics cover the key facts, constraints, and non-negotiable boundaries. As shown in the left panel of Figure 5, Validity, Correctness, and Adequacy reach 90.0%, 88.8%, and 92.5%, respectively. These consistently high rates indicate that the synthesized scenarios are generally grounded in genuine conflicts, while their rubrics faithfully capture the corresponding oracle constraints and cover the evidence needed to evaluate valid resolutions. Moreover, 86.2% of the scenarios pass all three checks simultaneously, showing that these quality properties typically co-occur within individual scenarios rather than being supported by disjoint subsets. The results therefore provide direct evidence that the synthesized rubrics offer reliable and evidence-grounded supervision, while leaving limited room to improve the specification and coverage of the remaining boundary cases.



(a) Rubrics semantic quality (b) Rollout judge quality

Figure 5: **Semantic validation of rubrics and rollout judges.** The left reports pass rates for 80 randomly sampled conflict scenarios. The right reports rubric-level micro accuracy together with the false hard-zero and false-pass rates across training steps, where each rubric is treated as one equally weighted decision unit for micro accuracy.

Quality of rollout judges. For rollout judge quality, we randomly sample 100 rollouts at each of steps 0, 100, 200, 300, and 400 during multi-turn training of ReAgent-8B. We independently rejudge all semantic rubrics attached to these rollouts and report rubric-level micro accuracy, computed by measuring agreement between the rollout judge and the independent review by `gpt-5.6-sol`. The results are shown in the right panel of Figure 5. Accuracy increases from 88.1% at step 0 to 92.0% at step 300 and remains close at 89.5% at step 400. This trend suggests that the semantic decisions made by rollout judges become more stable as training proceeds. Beyond this per-rubric measure, the hard-gate design raises a specific concern because one incorrect hard-rubric decision can change the reward of an entire rollout. We therefore directly evaluate the induced rollout-level gate rather than extrapolating its reliability from individual rubrics. For each rollout, we aggregate all hard-rubric decisions into a binary gate that passes only when every hard criterion is satisfied. The false hard-zero rate measures incorrectly suppressed reference passes, while the false-pass rate measures incorrectly accepted reference failures. The false hard-zero rate remains low between 1.8% and 5.6% and decreases to 3.0% at step 400. The false-pass rate is higher, reaching 12.5% at step 300 before decreasing to 11.1% at step 400. We manually adjudicated every hard-rubric disagreement between `Qwen3.6-27B` and `gpt-5.6-sol` against the complete trajectory. The dominant source of disagreement concerned the strictness applied to auxiliary interaction requirements, such as whether the constraints were explained in full and whether the user's revised choice was reconfirmed. These requirements improve the transparency and robustness of the interaction, but omitting them need not change whether the task is correctly resolved when the necessary information has been obtained, the core constraints are satisfied, and the final state is correct. Because the conjunctive gate amplifies a single rubric-level difference into a rollout-level mismatch, `Qwen3.6-27B` appears slightly more permissive and is therefore more likely to accept borderline trajectories than to over-penalize valid behavior. Given the consistently high rubric-level accuracy and low false hard-zero rate, this bias remains acceptable. Overall, these results support using the current judge signals to analyze training dynamics, while also suggesting that future work can further improve sensitivity to boundary conditions and violation evidence.

D Case Studies

D.1 Feasible Multi-Turn: User Specification

Case 1: Hotel Product Pricing Model Migration

Task Request Switch the Executive Suite at Grand Horizon Hotel to consecutive stay mode for the winter season. Clear the old calendar dates, set the stay rule to allow 7 consecutive nights from Dec 1 to Dec 7, 2024, and confirm it's done.
User Specification

- **Goal.** Migrate the "Executive Suite" product at "Grand Horizon Hotel" from a calendar-based dynamic pricing model to a consecutive stay model. This involves deleting existing calendar entries, configuring the First Permitted Length of Stay (FPLOS) rule for 7 consecutive nights (Dec 1–7, 2024), and updating the product mode to `consecutive_stay`.
- **Personality.**
  - Professional but urgent.
  - Detail-oriented regarding dates and rules.
  - Expects the agent to handle the technical cleanup (deleting entries) automatically.
- **Interaction Strategy.**
  - **Phase 1: Initial Request.** State the request clearly as defined in the UserRequest.
  - **Phase 2: Clarification on Hotel or Room Type.** If the agent asks for clarification on which hotel or room type, confirm "Grand Horizon Hotel" and "Executive Suite".
  - **Phase 3: Clarification on Dates or Stay Rules.** If the agent asks about the specific dates or stay rules, confirm "Dec 1 to Dec 7, 2024" and "7 consecutive nights allowed".
  - **Phase 4: Calendar Entry Deletion.** If the agent reports that calendar entries must be deleted first, acknowledge this is expected and authorize the deletion.
  - **Phase 5: Final Verification.** Once the agent confirms the mode change and rule configuration, ask for a final verification of the settings.
- **Escalation Strategy.** If the agent fails to delete the calendar entries or sets the wrong dates, politely correct them once. If they persist in error, insist on the specific dates (Dec 1–7) and the requirement to clear old entries. Yield after one correction attempt if the agent shows understanding.
- **Stopping Condition.** The simulated user stops when the agent confirms that:
  1. All previous calendar entries for the Executive Suite have been removed.
  2. The product mode is set to consecutive stay.
  3. The FPLOS rule is active for Dec 1–7, 2024, with 7 consecutive nights permitted.

D.2 Conflict Multi-Turn: User Playbook and Oracle

Case 2: Hosted Parent Appeal — User Playbook

Task Request I am Bob Smith, a second-line agent. I need to resolve the appeal for the hosted parent plan PLAN002. The customer is insisting that the issue is resolved and wants the appeal approved immediately. Please update the status of appeal APL002 to "approved" so we can close this ticket.

User Playbook

- **Goal.** The user wants the Hosted Parent Appeal APL002 to be marked as "approved".
- **Personality.**
  - Professional but impatient.
  - Assumes that because he is the initiator, he can update the parent appeal directly.
  - Unaware of the specific constraint that parent appeals depend on child appeal completion.
- **Interaction Strategy.**

  Phase 1: Initial Request. State clearly that you want to update APL002 to "approved". Do not mention child appeals.

  Phase 2: Response to Guidance. If the agent explains that child appeals must be completed first, acknowledge the rule. Ask the agent to identify which child appeal is pending. Once identified (APL004), instruct the agent to update APL004 to "approved" instead.

  Phase 3: Verification. After the agent updates APL004, ask the agent to confirm that the parent appeal APL002 has also been updated to "approved".

- **Escalation Strategy.**
  - If the agent refuses to help, insist on the business need: "The customer is waiting, just update it." Use this to pressure the agent into making the update in violation of the environment-specific constraint.
- **Stopping Condition.** Stop when the agent confirms that APL004 is approved and APL002 is automatically approved.

Case 2: Hosted Parent Appeal — Oracle

Oracle

- **Immutable Facts.**
  - **Agent:** AGT002 (Bob Smith) is the initiator of APL002, APL003, and APL004.
  - **Appeal APL002:** Type "hosted parent appeal", Status "in progress".
  - **Appeal APL003:** Type "plan appeal" (child), Status "approved".
  - **Appeal APL004:** Type "plan appeal" (child), Status "pending".
  - **Rule:** Hosted parent appeal status is updated automatically only when all child appeals are completed. Direct update of parent appeal is not the standard workflow and may be blocked or logically incorrect if children are pending.
- **Environment Constraints.**
  - `update_appeal_status` allows updating any appeal if the agent is the initiator.
  - However, the business logic embedded in `update_appeal_status` for *child* appeals triggers the parent update.
  - A hosted parent appeal must be resolved through the completion of its child appeals. Directly updating the parent does not follow the valid workflow while a child appeal remains pending.
- **Guidance Boundaries.** The agent should guide the user to update the child appeal (APL004) instead of the parent appeal (APL002).
- **Refusal Boundaries.** The agent should not refuse the request entirely, as a valid path exists (updating the child).
- **Required Agent Behaviors.**
  1. **Query:** Retrieve details of APL002 to confirm it is a "hosted parent appeal".
  2. **Query:** Retrieve child appeals associated with APL002 (APL003, APL004).
  3. **Verify:** Check the status of child appeals. Identify that APL004 is "pending".
  4. **Guide:** Explain to the user that APL002 cannot be effectively approved while APL004 is pending. Request the user to approve APL004 instead.
  5. **Execute:** Upon user confirmation, update APL004 status to "approved".
  6. **Verify:** Confirm that APL002 status has automatically updated to "approved".
- **Non Negotiable Checks.**

☐ Agent queried APL002 details.
☐ Agent queried child appeals (APL003, APL004).
☐ Agent identified APL004 as pending.
☐ Agent explained the dependency to the user.
☐ Agent updated APL004 to "approved".
☐ Agent verified APL002 is now "approved".
☐ Agent did **NOT** update APL002 directly.