VERIFIABLE STATE TRANSITIONS AS UNIFIED SUPERVISION FOR TRAINING STATEFUL TOOL-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training stateful tool-use agents requires evaluating not only whether they invoke the correct tools but also whether their actions produce the intended changes in persistent environments. However, supervision based primarily on final task outcomes does not explicitly assess intermediate execution, while tool-call matching alone cannot verify the actual effects of executed actions. Reliably assessing solution quality therefore requires jointly evaluating tool-call correctness and the resulting state transitions. We introduce **VESTA** (**V**erifiable **E**xecution and **S**tate **T**ransitions for **A**gents), a unified training framework that derives execution-grounded supervision from verifiable state transitions. VESTA constructs executable environments from API specifications and synthesizes tasks in an executable-first manner, recording reference tool calls and expected environment states at each task turn as state-transition contracts. These contracts enable joint verification of tool-call correctness and state changes, supporting SFT trajectory selection, policy-conditioned RL task selection, and RL reward computation without learned or LLM-based reward judges. Experiments across three Qwen3 model scales on BFCL-v3 Multi-Turn, ACEBench-Agent, and τ^2-Bench show improvements over the corresponding base models in all nine model–benchmark comparisons, with RL further improving over SFT in every setting. On Qwen3-4B, VESTA achieves absolute gains of 12.63, 20.00, and 8.76 points on the three benchmarks, respectively. Across model scales, VESTA outperforms EnvScaler in seven of nine comparisons and AWM in all six available comparisons.

1 INTRODUCTION

Large language models are increasingly used as agents that interact with external tools to complete complex tasks (Yao et al., 2023; Schick et al., 2023). Unlike isolated function calling, stateful tool use requires agents to resolve entities, bind arguments, coordinate dependent actions, and track persistent state across multiple turns (Yao et al., 2024; Lu et al., 2025; Trivedi et al., 2024). Recent benchmarks increasingly evaluate these capabilities in multi-turn environments (Patil et al., 2025; Barres et al., 2026; Liu et al., 2024a), motivating agent training through executable environments rather than static tool-call examples.

Recent work has developed diverse executable environments to support such training (Xi et al., 2025; Sullivan et al., 2025). EnvScaler and Agent World Model (AWM) scale synthetic stateful environment construction (Song et al., 2026; Wang et al., 2026b), while ScaleEnv, Agent-World, and EnvFactory further explore automated construction and grounding (Tu et al., 2026; Dong et al., 2026; Xu et al., 2026). In parallel, APIGen and ToolACE improve synthetic data quality through execution- and rule-based verification (Liu et al., 2024b; 2025), while Trajectory2Task and CoVe incorporate executable trajectories or explicit constraints into task generation (Wang et al., 2026c; Chen et al., 2026). Together, these advances establish executable environments and programmatic verification as foundations for tool-use agent training.

Despite this progress, reliably assessing execution quality in stateful interactions remains challenging. EnvScaler uses state checklists to verify final task outcomes (Song et al., 2026), while AWM applies step-level format checks but evaluates semantic success at the trajectory level (Wang et al., 2026b). Neither is designed to jointly verify intended tool calls and their resulting state changes at each task turn. Other approaches provide finer-grained supervision: TRACE estimates turn-level

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.

credit, OPRL learns implicit process rewards, and related methods employ learned reward models (Tao et al., 2026; Liu et al., 2026; Fan et al., 2026), rather than directly comparing both tool calls and state transitions against executable references. Automated evaluation also raises reliability concerns. Plan-RewardBench reports declining evaluator reliability with trajectory length (Wang et al., 2026a), while an audit of 496 expert-reviewed tool-use tasks finds 18.5% evaluator–human disagreement (Bhat et al., 2026). Yet executable task synthesis offers a natural opportunity to record reference tool calls and state transitions at each task turn and reuse them to jointly verify tool-call correctness and execution effects in subsequent rollouts.

To address this gap, we introduce **VESTA**, a unified training framework built around verifiable state transitions. VESTA constructs compact, executable environments from existing API specifications and adopts executable-first task synthesis, validating tool-use workflows before natural-language realization. Reference execution records the expected tool calls and environment states before and after each task turn, forming task-specific state-transition contracts. These contracts provide execution-derived targets for jointly verifying tool-call correctness and the resulting state changes without exposing reference information to the acting policy. *VESTA reuses this supervision for SFT trajectory selection, policy-conditioned RL task selection, and judge-free RL reward computation, aggregating turn-level execution verification into a trajectory-level reward for policy optimization.*

We evaluate VESTA on Qwen3 models at 1.7B, 4B, and 8B scales across BFCL-v3 Multi-Turn, ACEBench-Agent, and τ^2-Bench. Using a shared set of 3,600 SFT tasks and 2,400 policy-conditioned RL tasks selected separately for each model, VESTA improves all nine model–benchmark averages over the corresponding base checkpoints, with a mean absolute gain of 12.97 points. It outperforms EnvScaler in seven of nine comparisons, ties in one, and outperforms AWM in all six available comparisons. Further RL training improves upon SFT in every setting, while ablations demonstrate the complementary benefits of state- and tool-based rewards and policy-conditioned task selection. Our main contributions are as follows:

- We develop an **API-grounded, executable-first synthesis pipeline** that constructs stateful environments and validates tool-use workflows before natural-language realization, yielding 131 curated, replay-validated environments.
- We introduce **execution-derived state-transition contracts** that jointly verify tool-call correctness and resulting state changes, providing reusable supervision for SFT trajectory selection, policy-conditioned RL task selection, and judge-free RL rewards.
- We demonstrate the effectiveness of **unified execution-grounded supervision** across three model scales and three tool-use benchmarks, with analyses of reward design, task selection, rollout strategy, and environment and workflow scaling.

2 METHOD

VESTA constructs replay-validated executable environments and synthesizes tasks in an executable-first manner to derive state-transition contracts. These contracts provide reusable supervision for SFT trajectory selection, RL reward computation, and policy-conditioned task selection.

2.1 API-GROUNDED EXECUTABLE ENVIRONMENT CONSTRUCTION

Let $\mathcal{R}$ denote a source API repository and $\mathcal{E} = (\mathcal{S}, \mathcal{A}, F_{\mathcal{E}}, S_0)$ an executable environment, where $\mathcal{S}$ is the persistent state space, $\mathcal{A}$ is the tool-call space, and S_0 is the initial state. Executing tool call $a \in \mathcal{A}$ in state S yields an observation o and successor state S', denoted by $(o, S') = F_{\mathcal{E}}(S, a)$.

API Grounding and Core Extraction. In Step 1 of Figure 1, we parse API specifications into tools, parameter and response schemas, and persistent entities, organizing their read, write, and dependency relations into a typed tool–entity graph $G = (V_{\text{tool}} \cup V_{\text{ent}}, E)$. Model-assisted selection then extracts a compact, connected stateful core $G_c \subseteq G$, prioritizing persistent entities and observable mutations while excluding unrelated operations. In the illustrated example, the selected core connects order, payment, and refund entities through four tools, providing the API-grounded structure for subsequent semantic completion.

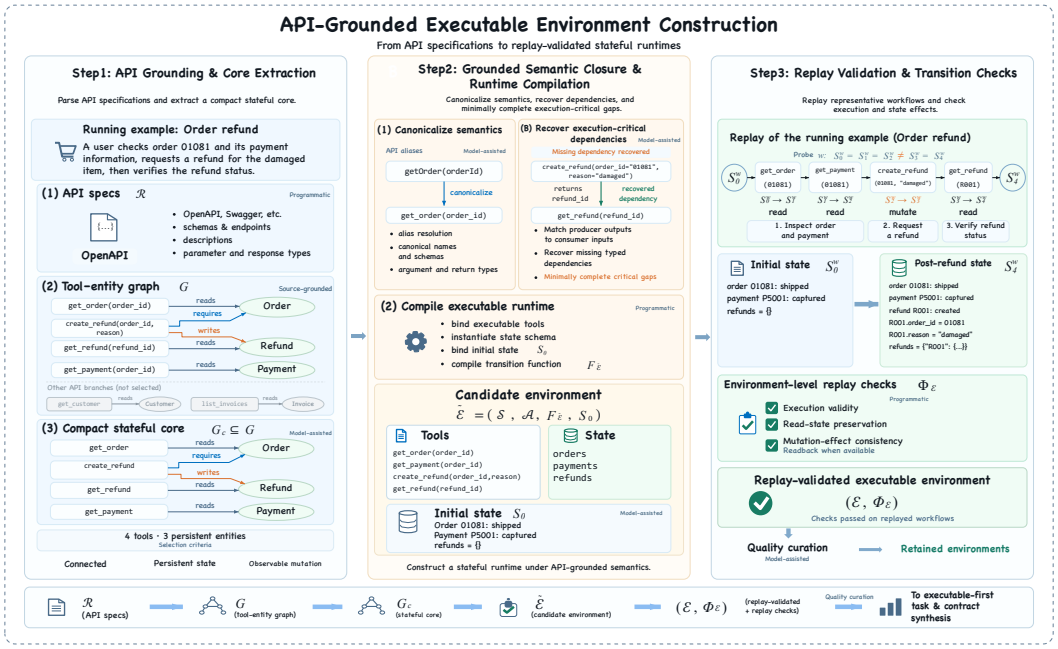


Figure 1: API-grounded executable environment construction. Step 1 extracts a compact stateful core from API specifications. Step 2 completes execution-critical semantics and compiles an executable runtime. Step 3 validates tool execution and state transitions through workflow replay.

Grounded Semantic Closure and Runtime Compilation. In Step 2, we complete the selected core by canonicalizing tool and entity semantics, recovering execution-critical dependencies, and conservatively filling gaps required for executable workflows. For example, the refund identifier returned by create_refund must be passed to the get_refund call. Model-assisted reconstruction completes the environment semantics and generates an initial state S_0 containing synthetic records and valid entity relationships. After structural validation, programmatic compilation integrates the completed semantics and initial state, binds executable tools, and implements persistent-state transition logic. The resulting candidate environment $\widetilde{\mathcal{E}} = (\mathcal{S}, \mathcal{A}, F_{\widetilde{\mathcal{E}}}, S_0)$ is executable but must still pass replay validation.

Replay Validation and Transition Checks. In Step 3, we replay representative workflows in isolated instances of the candidate environment and inspect their execution results and persistent-state transitions. The illustrated order-refund workflow proceeds from order and payment inspection to refund creation and subsequent verification. Environment-level checks $\Phi_{\mathcal{E}}$ assess successful execution, state preservation by read-only operations, consistency of observed mutations with their expected effects, and readback behavior when available. These checks establish internal consistency for the tested workflows rather than equivalence to the original remote APIs. Candidates passing replay validation undergo model-assisted quality curation, yielding retained environments for the task-specific workflow synthesis and state-transition contracts described next. Additional implementation details are provided in Appendix A.

2.2 Executable-First Synthesis of State-Transition Contracts

Building on the replay-validated environments from Section 2.1, we construct executable workflow skeletons and reference state-transition contracts (Step 1), realize these workflows through offline and online rollout strategies (Step 2), and evaluate the resulting trajectories against the contracts to derive SFT and RL supervision (Step 3), as illustrated in Figure 2.

Per-Stage Executable Skeletons and Reference State Transitions. In Step 1, each task x is represented by a workflow skeleton $\chi(x) = (g_t)_{t=1}^{T}$, where each stage specifies an operational requirement g_t and defines one task turn. We construct reference tool calls A_t^* for each stage and

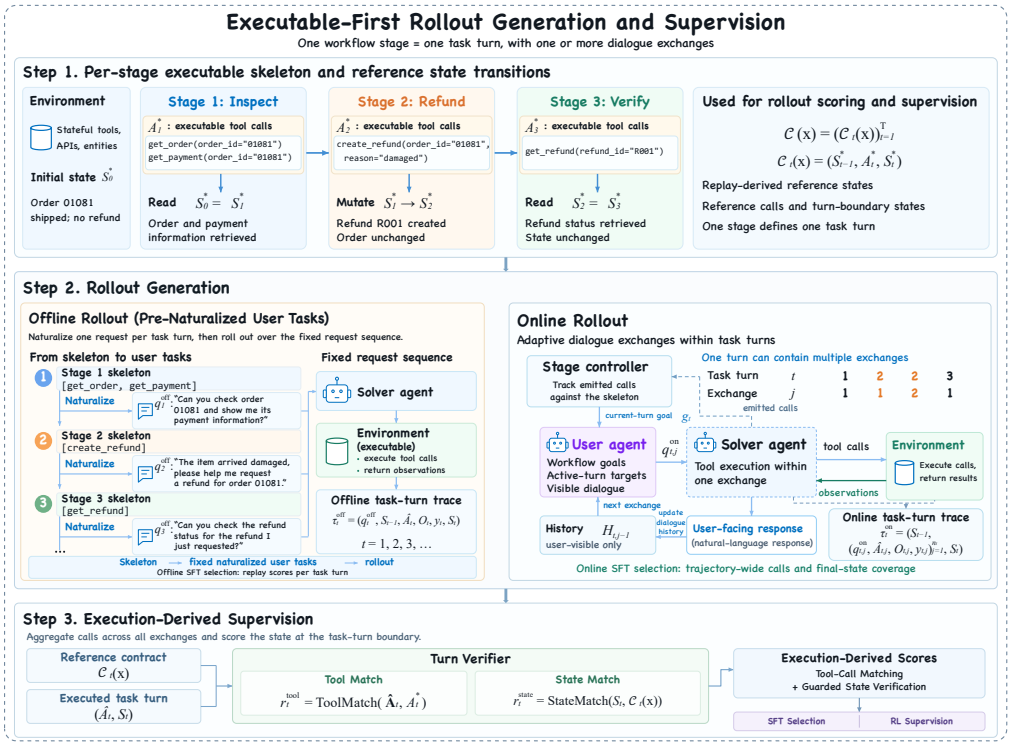


Figure 2: Executable-first rollout generation and supervision. Step 1 constructs executable workflow skeletons and reference state transitions. Step 2 realizes them through fixed-request offline rollouts or adaptive online interactions. Step 3 derives SFT selection scores and RL rewards through execution-based verification.

replay the complete workflow from its saved initial state S_0^*, recording pre- and post-turn states. As illustrated in Figure 2, the order-refund workflow comprises three stages: inspecting the order and payment, creating a refund, and verifying its status. The first and last stages preserve state, while refund creation produces the intended mutation. Workflows failing execution or state checks are discarded. For each retained task, we define the state-transition contract

$$\mathcal{C}_t(x) = \left(S_{t-1}^*, A_t^*, S_t^*\right), \qquad \mathcal{C}(x) = \left(\mathcal{C}_t(x)\right)_{t=1}^{T}. \tag{1}$$

Here, A_t^* denotes the reference tool-call sequence, and S_{t-1}^* and S_t^* are the reference pre- and post-turn states. The skeleton defines the task structure, while its contracts provide the targets for subsequent execution verification.

Offline and Online Rollout Generation. In Step 2, both strategies use the validated skeleton and its contracts but differ in how they guide the solver through the workflow. In the offline branch (left), a naturalizer converts each stage into one fixed user request while preserving the reference tool-call plan. All requests are generated before rollout, and the solver processes them sequentially while interacting with the executable environment. In the online branch (right), a user simulator instead guides the solver through the ordered stages, adaptively generating messages based on the active-stage goal and dialogue history. A stage controller tracks progression from the solver's emitted tool calls, allowing multiple exchanges for clarification, confirmation, or follow-up within a single task turn. These adaptive exchanges change how each stage is presented without altering the underlying workflow or its reference contracts. In both branches, the solver receives user messages, tool schemas, and actual execution observations, but not reference calls or target states.

Execution-Derived Supervision. In Step 3, we evaluate executed tool calls and observed states against the reference contracts from Step 1. For offline SFT selection, explicit task-turn boundaries allow per-turn verification and aggregation into a trajectory-level score. For online SFT selection,

potentially ambiguous stage boundaries motivate pooled tool-call matching and final-state coverage over the complete trajectory. For RL, the same contracts support turn-level tool and state verification, aggregated into trajectory-level rewards and used for policy-conditioned task selection (Section 2.3). Further synthesis and SFT scoring details are provided in Appendix B, with offline and online case studies in Appendix H.

2.3 Trajectory-Level RL Reward from Turn-Level Verification

For each task turn t, we aggregate solver calls across its dialogue exchanges into $\widehat{A}_t$ and record the observed post-turn state S_t. Let r_t^{tool} denote the multiset F1 between $\widehat{A}_t$ and the reference calls A_t^*, and let Δ_t^* denote the expected changed-state paths. We define a state-consistency guard over the compared paths outside the expected change set:

$$G_t = \mathbb{I}[S_t(p) = S_t^*(p),\ \forall p \notin \Delta_t^*]\,. \tag{2}$$

For mutating turns, changed-state recall measures the fraction of intended changes correctly realized:

$$r_t^{\text{state}} = \frac{1}{|\Delta_t^*|} \sum_{p \in \Delta_t^*} \mathbb{I}[S_t(p) = S_t^*(p)]\,. \tag{3}$$

Our default turn-level reward combines tool-call matching and state verification:

$$r_t = \begin{cases} G_t\, r_t^{\text{tool}}, & \Delta_t^* = \emptyset, \\ \dfrac{1}{2}\left(r_t^{\text{tool}} + G_t\, r_t^{\text{state}}\right), & \Delta_t^* \neq \emptyset. \end{cases} \tag{4}$$

For read-only turns, tool credit requires state preservation; for mutating turns, unintended changes outside the target set eliminate state credit without automatically eliminating tool credit. We average turn-level rewards into a single trajectory-level reward, $R_{\text{RL}}(\tau; x) = \frac{1}{T}\sum_{t=1}^{T} r_t$, for GRPO optimization.

SFT Filtering and Policy-Conditioned RL Task Selection. For SFT, a trajectory-level selection score of 1 indicates maximal agreement under the corresponding metric. Since such trajectories are scarce, we retain candidates scoring above 0.75. For RL, task selection is conditioned on the policy to be trained: we roll out the RL initialization checkpoint of each model on approximately 15,000 candidate tasks, score the resulting trajectories with R_{RL}, and select 2,400 tasks through stratified sampling while balancing environment and workflow coverage. The reward intervals $[0, 0.25)$, $[0.25, 0.5)$, $[0.5, 0.75)$, and $[0.75, 1]$ have target proportions of 5%, 20%, 45%, and 30%, respectively.

3 Experiment

3.1 Experimental Setup

Training Setup. We evaluate our framework on Qwen3-1.7B, Qwen3-4B, and Qwen3-8B (Yang et al., 2025), all trained in thinking mode. Training consists of two stages: supervised fine-tuning (SFT) on 3,600 tasks, each potentially spanning multiple turns, followed by GRPO (Shao et al., 2024) initialized from the corresponding SFT checkpoint. For RL, we select 2,400 policy-conditioned tasks per model and optimize with a trajectory-level reward obtained by averaging execution-derived turn rewards (Section 2.3). Environment and data statistics are reported in Section 3.2, with further training details provided in Appendix E.

Baselines. We compare against the corresponding Qwen3 base models in thinking mode and two recent executable-environment training baselines, *EnvScaler* (Song et al., 2026) and *AWM* (Wang et al., 2026b). For both baselines, we evaluate the checkpoints released by the original authors under the same protocol as our models (Appendix F); AWM is reported only at 4B and 8B because no 1.7B checkpoint is available. Since these baselines differ from VESTA in training data and scale, the comparison reflects complete training pipelines rather than a controlled comparison of individual components.

Evaluation. We evaluate all models on three agentic tool-use benchmarks: *BFCL-v3 Multi-Turn (BFCL-MT)* (Patil et al., 2025), *ACEBench-Agent* (Chen et al., 2025), and τ^2*-Bench* (Barres et al., 2026), following their official evaluation protocols and metrics. BFCL-MT includes Base, Missing Functions (MF), Missing Parameters (MP), and Long Context (LC); ACEBench-Agent includes Multi-step (MS) and Multi-turn (MT); and τ^2-Bench includes Airline, Retail, and Telecom. For interactive evaluation on ACEBench-Agent MT and τ^2-Bench, we use Qwen3-235B-A22B-Instruct-2507 as the user simulator. We make minor adjustments to tool-call parsing and extraction for Qwen3 compatibility without modifying the underlying tasks or evaluation criteria.

3.2 Environment and Data Construction

We construct 131 high-quality, replay-validated executable environments using the pipeline in Figure 1. The SFT pool contains 100 unique environments, comprising 50 offline and 80 online environments with 30 shared between them. We select 3,600 SFT tasks, including 900 offline and 2,700 online examples. For RL, we use 60 environments, including 32 shared with SFT and 28 newly introduced, and select 2,400 policy-conditioned tasks per model. Together, the SFT and RL training pools cover 128 unique environments.

Table 1: Structural statistics of the 131 retained executable environments.

Mean per environment			Structural validity			Tool composition
Tools	Entities	Relations	Entity refs	Tool bindings	State keys	Mutating tools
20.79	5.73	6.42	100.00%	99.96%	100.00%	67.17%

Table 1 summarizes the structural characteristics of the retained environments. Their nearly complete structural coverage and substantial proportion of mutating tools provide a foundation for constructing and verifying stateful workflows. Detailed metric definitions are provided in Appendix A.

Task Diversity by Design. To ensure diversity beyond environment coverage, we vary operation sequences, entity dependencies, state mutations, and verification steps within each environment. Our offline and online SFT catalogs contain 16 workflow styles each, yielding 32 styles in total, while the RL catalog reuses the 16 offline styles and introduces four additional styles. These patterns diversify task structures and user interactions across environments. Detailed environment pool composition and workflow catalogs are provided in Appendices D and C, respectively.

3.3 Main Results

We evaluate the effectiveness of our data construction framework through two training configurations: SFT on 3,600 tasks for 3 epochs (VESTA-SFT), and our full pipeline, which further applies 80 RL training steps on 2,400 tasks initialized from the corresponding SFT checkpoint (VESTA). Table 2 reports results across three model scales on BFCL-MT, ACEBench-Agent, and τ^2-Bench.

VESTA improves benchmark-average performance over the corresponding Qwen3 base models in all nine model–benchmark comparisons, with gains exceeding 10 points in six cases. The largest improvement occurs on ACEBench-Agent at the 1.7B scale, where the average score increases from 8.33 to 30.00 (+21.67). These gains persist at larger model scales, exceeding 10 points on both BFCL-MT and ACEBench-Agent for the 4B and 8B models. Compared with existing baselines, VESTA surpasses *EnvScaler* in seven of nine comparisons, ties on 8B ACEBench-Agent, and trails only on 1.7B BFCL-MT. It also exceeds *AWM* in all six comparisons available at the 4B and 8B scales.

Further RL training improves upon VESTA-SFT across all nine model–benchmark averages, yielding additional gains ranging from 0.37 to 5.83. The largest improvement occurs on 8B ACEBench-Agent, where the average score increases from 51.67 to 57.50, while gains on some settings, such as 8B τ^2-Bench, are more modest. These results indicate that our constructed training data substantially improve tool-use capabilities through SFT, while execution-derived rewards provide effective supervision for further improvements during RL.

Table 2: Main results on three tool-use benchmarks. Green and red indicate absolute gains and losses relative to the corresponding base model. Best and second-best distinct scores within each model-size group are bolded and underlined, respectively, with ties marked equally.

Method	BFCL-MT					ACEBench-Agent			τ^2-Bench			
	Base	MF	MP	LC	Avg.	MS	MT	Avg.	Airline	Retail	Telecom	Avg.
Qwen3-1.7B												
Base	20.50	13.50	16.00	14.00	16.00	10.00	6.67	8.33	14.00	7.02	12.28	11.10
EnvScaler-1.7B	**32.50**	24.50	**23.50**	**24.50**	**26.25** +10.25	35.00	10.00	22.50 +14.17	24.00	**22.81**	14.91	20.57 +9.47
VESTA-SFT	29.50	**25.50**	21.50	16.50	23.25 +7.25	35.00	**20.00**	27.50 +19.17	26.00	15.79	16.67	19.49 +8.39
VESTA	31.50	25.00	22.00	17.00	23.88 +7.88	**40.00**	**20.00**	**30.00** +21.67	**34.00**	18.42	**17.54**	**23.32** +12.22
Qwen3-4B												
Base	37.50	40.00	28.00	26.00	32.88	35.00	20.00	27.50	32.00	35.96	14.04	27.33
AWM-4B	39.50	32.50	29.00	22.50	30.88 -2.00	15.00	30.00	22.50 -5.00	32.00	32.46	19.30	27.92 +0.58
EnvScaler-4B	53.50	**50.00**	35.00	**39.00**	44.38 +11.50	35.00	**43.33**	39.17 +11.67	34.00	**40.35**	13.16	29.17 +1.84
VESTA-SFT	54.50	45.50	40.00	38.50	44.63 +11.75	50.00	40.00	45.00 +17.50	40.00	35.96	20.18	32.05 +4.71
VESTA	**57.00**	49.00	**41.00**	35.00	**45.50** +12.63	**55.00**	40.00	**47.50** +20.00	**46.00**	35.09	**27.19**	**36.09** +8.76
Qwen3-8B												
Base	46.50	45.00	39.00	28.50	39.75	45.00	36.67	40.83	32.00	39.47	18.42	29.96
AWM-8B	44.50	42.00	36.50	32.50	38.88 -0.88	35.00	40.00	37.50 -3.33	38.00	35.09	15.79	29.63 -0.34
EnvScaler-8B	61.50	51.50	35.50	**45.50**	48.50 +8.75	**65.00**	**50.00**	**57.50** +16.67	34.00	42.11	14.91	30.34 +0.37
VESTA-SFT	58.00	53.50	40.00	41.00	48.13 +8.38	60.00	43.33	51.67 +10.83	40.00	**42.98**	24.56	35.85 +5.88
VESTA	**62.50**	**54.00**	**43.00**	42.00	**50.38** +10.63	**65.00**	**50.00**	**57.50** +16.67	**42.00**	40.35	**26.32**	**36.22** +6.26

3.4 Direct Reinforcement Learning

To assess the effectiveness of execution-derived rewards without SFT initialization, we apply RL directly to the Qwen3 base checkpoints across three model scales (denoted as VESTA-RL). As shown in Table 3, VESTA-RL improves eight of nine benchmark averages, suggesting that trajectory-level rewards constructed from turn-level execution verification provide a useful training signal even without prior SFT. However, its gains are generally smaller and less consistent than those of the full VESTA pipeline. For instance, VESTA-RL reduces 8B ACEBench-Agent performance by 10.83, whereas VESTA improves it by 16.67 over the same base model. VESTA outperforms VESTA-RL on all nine benchmark averages, indicating that SFT initialization contributes substantially to the gains of the full pipeline under our experimental settings.

Table 3: Performance of direct RL initialized from base checkpoints across three model scales. Shaded rows indicate VESTA-RL.

Model	Method	BFCL-MT						ACEBench-Agent				τ^2-Bench				
		Base	MF	MP	LC	Avg.		MS	MT	Avg.		Airline	Retail	Telecom	Avg.	
Qwen3-1.7B	*Base*	20.50	13.50	16.00	14.00	16.00		10.00	6.67	8.33		14.00	7.02	12.28	11.10	
	VESTA-RL	26.50	15.00	13.00	14.00	17.13	+1.13	40.00	6.67	23.33	+15.00	24.00	7.02	23.68	18.23	+7.13
Qwen3-4B	*Base*	37.50	40.00	28.00	26.00	32.88		35.00	20.00	27.50		32.00	35.96	14.04	27.33	
	VESTA-RL	50.50	43.00	35.50	33.00	40.50	+7.63	30.00	26.67	28.33	+0.83	36.00	34.21	21.93	30.71	+3.38
Qwen3-8B	*Base*	46.50	45.00	39.00	28.50	39.75		45.00	36.67	40.83		32.00	39.47	18.42	29.96	
	VESTA-RL	50.50	51.00	42.50	37.00	45.25	+5.50	20.00	40.00	30.00	-10.83	34.00	35.96	28.07	32.68	+2.71

3.5 Scaling Effects of Environments and Workflows

We study environment and workflow scaling on Qwen3-4B, training each configuration on 1,000 tasks sampled from the 3,600-task SFT dataset. Environment scaling varies the number of environments with all 32 workflow styles available, while workflow scaling varies the number of workflow

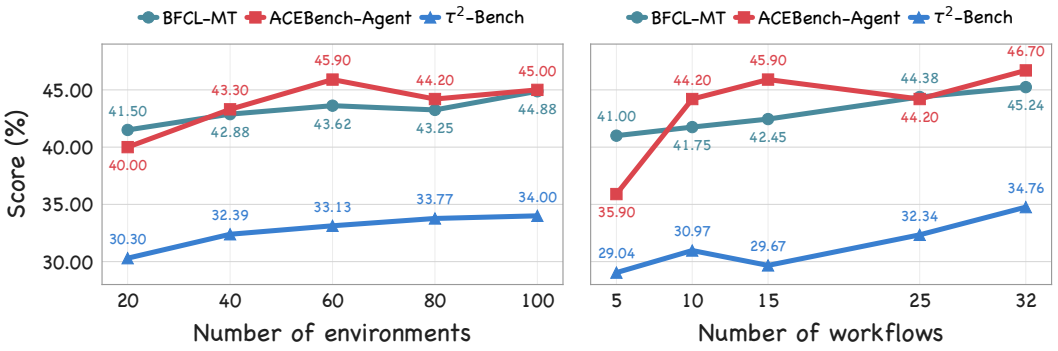


Figure 3: Effects of environment and workflow scaling using Qwen3-4B. (a) Environment scaling. (b) Workflow scaling. Each curve reports the average across the corresponding benchmark's subsets.

styles with 80 environments fixed. As shown in Figure 3, performance generally increases along both dimensions on all three benchmarks. Over the evaluated ranges, increasing workflow diversity yields larger gains between the smallest and largest configurations than increasing environment coverage, most notably on ACEBench-Agent (+10.80 vs. +5.00). These results suggest that workflow diversity is an important scaling dimension complementary to environment coverage.

Table 4: Training ablations on Qwen3-4B: (a) SFT rollout strategy, (b) RL reward configuration, and (c) RL task selection. Shaded columns report benchmark averages, with the highest average in each ablation bolded. State + Tool and Weighted represent the same default run.

Configuration	**BFCL-MT**					**ACEBench-Agent**			**τ^2-Bench**			
	Base	MF	MP	LC	Avg.	MS	MT	Avg.	Airline	Retail	Telecom	Avg.
(a) SFT Rollout Strategy												
Offline	44.50	54.50	51.50	36.00	**46.63**	50.00	23.33	36.67	30.00	36.84	21.05	29.30
Online	42.00	51.00	49.50	34.50	44.25	55.00	33.33	**44.17**	38.00	38.60	24.56	**33.72**
(b) RL Reward Configuration												
State-only	55.00	43.50	40.00	34.00	43.13	45.00	43.33	44.17	30.00	43.86	26.32	33.39
Tool-only	55.00	44.00	36.00	32.00	41.75	35.00	40.00	37.50	40.00	37.72	29.82	35.85
State + Tool	57.00	49.00	41.00	35.00	**45.50**	55.00	40.00	**47.50**	46.00	35.09	27.19	**36.09**
(c) RL Task Selection												
Random	55.00	45.50	34.50	33.50	42.13	55.00	30.00	42.50	42.00	25.44	30.70	32.71
Weighted	57.00	49.00	41.00	35.00	**45.50**	55.00	40.00	**47.50**	46.00	35.09	27.19	**36.09**

3.6 Analysis of Training Design Choices

Setup. We study three design choices on Qwen3-4B: (a) offline versus online SFT rollouts, using 1,200 trajectories per setting; (b) State-only ($G_t r_t^{\text{state}}$), Tool-only (r_t^{tool}), and State + Tool (Eq. 4) rewards on mutating turns; and (c) random versus policy-conditioned weighted RL task selection. Both RL ablations use 2,400 tasks per setting, with the same SFT checkpoint for (b) and the combined reward for (c). All reward variants use $G_t r_t^{\text{tool}}$ on read-only turns, and their turn-level rewards are averaged into a trajectory-level reward for GRPO. Training configurations are held fixed within each ablation.

SFT Rollout Strategy. As shown in Table 4(a), offline rollouts outperform online rollouts on BFCL-MT (46.63 vs. 44.25), whereas online rollouts perform better on ACEBench-Agent (44.17 vs. 36.67) and τ^2-Bench (33.72 vs. 29.30). The online advantage is particularly evident on ACEBench-Agent Multi-turn (33.33 vs. 23.33), consistent with its adaptive clarification and follow-up inter-

actions. In contrast, offline rollouts preserve fixed requests and explicit task-turn boundaries for alignment with reference transitions. These complementary strengths motivate combining both rollout strategies in our SFT data.

RL Reward Configuration. State + Tool achieves the highest benchmark averages (45.50 on BFCL-MT, 47.50 on ACEBench-Agent, and 36.09 on τ^2-Bench), outperforming both individual rewards (Table 4(b)). Neither tool-call matching nor state verification alone captures both the executed operations and their intended effects. Their combination provides complementary supervision over action correctness and state transitions.

RL Task Selection. Under the same budget of 2,400 tasks, policy-conditioned weighted sampling improves benchmark averages over random sampling by 3.37 on BFCL-MT, 5.00 on ACEBench-Agent, and 3.38 on τ^2-Bench (Table 4(c)). The gains are particularly pronounced on BFCL-MT Missing Parameters (+6.50) and ACEBench-Agent Multi-turn (+10.00), although random sampling performs better on Telecom. These results support policy-conditioned task selection while highlighting variation across task types.

4 Related work

Training and supervision for tool-use agents. Tool-use agents are increasingly trained through interaction with executable environments (Xi et al., 2025; Sullivan et al., 2025; Zhao et al., 2025; Luo et al., 2025; Gao et al., 2026). Trajectory-based fine-tuning has also been explored in Agent-Tuning and FireAct (Zeng et al., 2024; Chen et al., 2023), while WebRL and RAGEN investigate reinforcement learning for interactive agents (Qi et al., 2025; Wang et al., 2025b). For finer-grained supervision, prior work estimates turn-level credit (Tao et al., 2026), learns process rewards (Liu et al., 2026; Fan et al., 2026), matches predicted and reference tool traces (Qu et al., 2026), or uses action confidence (Li et al., 2026). More broadly, process supervision has been studied for multi-step reasoning and agent training (Lightman et al., 2024; Choudhury, 2025). Our approach derives turn-level rewards by combining tool-call matching with verification against execution-derived reference states. Unlike turn-level optimization methods such as GTPO (Ding et al., 2026), we aggregate these rewards into a trajectory-level signal for GRPO.

Executable environments and verifiable data. Executable environments have been synthesized programmatically (Song et al., 2026; Wang et al., 2026b; Tu et al., 2026) or grounded in real-world resources (Dong et al., 2026; Xu et al., 2026). Complementary work develops interactive environments for web and computer-use agents (Zhou et al., 2024; Drouin et al., 2024; Xie et al., 2024). Synthetic tool-use data can be validated through execution-, rule-, and model-based verification (Liu et al., 2024b; 2025) or constructed from trajectories (Wang et al., 2026c; Chen et al., 2026). Related data-generation frameworks explore large-scale API instruction tuning, coherent tool-use dialogues, and multi-turn interaction synthesis (Qin et al., 2024; Wang et al., 2025a; Prabhakar et al., 2025). We build environments from fragmented API specifications and reuse replay-derived reference states beyond data verification, providing supervision for SFT filtering, RL rewards, and task selection.

5 Conclusion

Training stateful tool-use agents requires verifying not only tool-call correctness but also the resulting state transitions, which cannot be fully captured by final-outcome evaluation or tool-call matching alone. To address this challenge, we introduce VESTA, a unified framework for execution-grounded supervision. VESTA combines API-grounded environment construction with executable-first task synthesis to derive reference tool calls and expected state transitions at each task turn. These state-transition contracts support offline and online SFT trajectory selection, policy-conditioned RL task selection, and judge-free RL rewards derived from turn-level execution verification. VESTA improves all nine model and benchmark combinations over the corresponding base models, achieving a mean absolute gain of 12.97. Further RL training improves upon SFT in all nine settings, while ablations highlight the complementary contributions of state- and tool-based rewards and the benefits of policy-conditioned task selection. Together, these findings illustrate how verifiable state-transition contracts connect executable data construction, trajectory selection, and reward-based optimization within a unified training framework.

AI USE STATEMENT

We use generative AI for semantic reconstruction, state initialization, and quality assessment during executable environment construction, as detailed in Appendix A. Language models are also employed for task naturalization, solver trajectory generation, adaptive user simulation, and optional dialogue-quality assessment during data curation (Appendix B). Reference execution, state-transition verification, SFT selection scores, and RL rewards are computed programmatically rather than determined by learned or LLM-based reward judges. We additionally use AI assistance for manuscript preparation and language polishing. The authors review AI-assisted outputs and take full responsibility for the research design, experimental results, their interpretation, and the final manuscript.

ETHICS STATEMENT

Our experiments use synthetic records and simulated interactions in isolated executable environments, without involving human participants or interacting with live external services. Nevertheless, model-generated environments and trajectories may contain inherited biases or semantic inaccuracies, and simulated interactions may not fully represent real-world user behavior. We apply source-grounded construction and execution-based validation to reduce inconsistencies, although these checks do not guarantee behavioral equivalence to the original services (Appendix A). Deploying agents trained with this approach on live systems would require additional safeguards, particularly for operations that modify persistent state or involve sensitive information.

REPRODUCIBILITY STATEMENT

We provide methodological and experimental details to facilitate independent reproduction. Appendices A and B document environment construction, reference workflow execution, trajectory generation, and SFT selection procedures. Section 2.3 specifies the RL reward and policy-conditioned task selection, while Appendices C and D report workflow catalogs and environment-pool composition. Training configurations, model settings, and benchmark-specific evaluation protocols are detailed in Appendices E and F. Code is not publicly released at the submission stage; consequently, exact reproduction of our implementation may require access to the original code and generated artifacts. The provided descriptions and configurations are intended to support independent reimplementation and evaluation.

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. URL https://openreview.net/forum?id=OC2z7iSQKa.

Vishvesh Bhat, Jay Vaghasiya, Muhammad Ahmed Mohsin, and Asad Aali. Benchmarking the benchmarks: A validity audit of tool-calling evaluation. *arXiv preprint arXiv:2607.02577*, 2026. URL https://arxiv.org/abs/2607.02577.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning. *arXiv preprint arXiv:2310.05915*, 2023. URL https://arxiv.org/abs/2310.05915.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Xinzhi Wang, et al. Acebench: A comprehensive evaluation of llm tool usage. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025. URL https://aclanthology.org/2025.findings-emnlp.697/.

Jinpeng Chen, Cheng Gong, Hanbo Li, Ziru Liu, Zichen Tian, Xinyu Fu, Shi Wu, Chenyang Zhang, Wu Zhang, Suiyun Zhang, et al. Cove: Training interactive tool-use agents via constraint-guided verification. *arXiv preprint arXiv:2603.01940*, 2026. URL https://arxiv.org/abs/2603.01940.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025. URL https://arxiv.org/abs/2502.10325.

Yifeng Ding, Hung Le, Songyang Han, Kangrui Ruan, Zhenghui Jin, Varun Kumar, Zijian Wang, and Anoop Deoras. Empowering multi-turn tool-integrated agentic reasoning with group turn policy optimization. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026. URL https://aclanthology.org/2026.acl-long.1962/.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026. URL https://arxiv.org/pdf/2604.18292.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11642–11662. PMLR, 2024. URL https://proceedings.mlr.press/v235/drouin24a.html.

Kaixuan Fan, Kaituo Feng, Manyuan Zhang, Tianshuo Peng, Zhixun Li, Yilei Jiang, Shuang Chen, and Xiangyu Yue. Exploring reasoning reward model for agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.95/.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Shusheng Xu, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents. *arXiv preprint arXiv:2601.22607*, 2026. URL https://arxiv.org/abs/2601.22607.

Meng Li, Lei Li, Xiting Wang, Yi Yuan, Zheng Wei, Zang Li, et al. Soar: Supervision from observation for agentic reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 35175–35197, 2026. URL https://aclanthology.org/2026.acl-long.1624/.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, 2025. URL https://openreview.net/forum?id=8EB8k6DdCU.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024a. URL https://arxiv.org/abs/2308.03688.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Jianbin Jiao, and Junge Zhang. Agentic reinforcement learning with implicit step rewards. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ooROvpmxMV.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. In *Advances in Neural Information Processing Systems*, 2024b.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025. URL https://arxiv.org/abs/2408.04682.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025. URL https://arxiv.org/abs/2508.03680.

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. URL https://openreview.net/forum?id=2GmDdhBdDk.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. In *Advances in Neural Information Processing Systems*, 2025. URL https://doi.org/10.52202/085713-2184.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.16789.

Changle Qu, Sunhao Dai, Hengyi Cai, Jun Xu, Shuaiqiang Wang, and Dawei Yin. Matchtir: Fine-grained supervision for tool-integrated reasoning via bipartite matching. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026. URL https://aclanthology.org/2026.acl-long.549/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

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*, 2026. URL https://aclanthology.org/2026.findings-acl.407/.

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*, 2025. URL https://aclanthology.org/2025.emnlp-main.936/.

Leitian Tao, Baolin Peng, Wenlin Yao, Tao Ge, Hao Cheng, Mike Hang Wang, Jianfeng Gao, and Sharon Li. Trace: Turn-level reward assignment via credit estimation for long-horizon agents. *arXiv preprint arXiv:2607.13988*, 2026. URL https://arxiv.org/abs/2607.13988.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://aclanthology.org/2024.acl-long.850/.

Dunwei Tu, Hongyan Hao, Hansi Yang, Yihao Chen, Yi-Kai Zhang, Zhikang Xia, Yu Yang, Yueqing Sun, Xingchen Liu, Furao Shen, et al. Scaleenv: Scaling environment synthesis from scratch for generalist interactive tool-use agent training. *arXiv preprint arXiv:2602.06820*, 2026. URL https://arxiv.org/pdf/2602.06820.

Jiaxuan Wang, Yulan Hu, Wenjin Yang, Zheng Pan, Xin Li, and Lan-Zhe Guo. A benchmark for trajectory-level reward modeling. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026a. URL https://aclanthology.org/2026.acl-long.1062/.

Zezhong Wang, Xingshan Zeng, Weiwen Liu, Liangyou Li, Yasheng Wang, Lifeng Shang, Xin Jiang, Qun Liu, and Kam-Fai Wong. Toolflow: Boosting llm tool-calling through natural and coherent dialogue synthesis. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2025a. URL https://aclanthology.org/2025.naacl-long.214/.

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. *arXiv preprint arXiv:2602.10090*, 2026b. URL https://arxiv.org/abs/2602.10090.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025b. URL https://arxiv.org/abs/2504.20073.

Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Pei Chen, Ziwei Dong, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Qun Liu, et al. Trajectory2task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026c. URL https://aclanthology.org/2026.acl-long.2037/.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, et al. Agentgym: Evaluating and training large language model-based agents across diverse environments. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025. URL https://aclanthology.org/2025.acl-long.1355/.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=tN61DTr4Ed.

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. URL https://arxiv.org/abs/2605.18703.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. URL https://arxiv.org/abs/2505.09388.

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 Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. URL https://aclanthology.org/2024.findings-acl.181/.

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. URL https://arxiv.org/abs/2508.18669.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=oKn9c6ytLx.

A Implementation Details of API-Grounded Environment Construction

We expand the three-step environment construction pipeline in Figure 1 into four implementation stages. Stage I performs API grounding and core selection (Step 1); Stages II and III complete the environment semantics, initialize persistent state, and compile the executable runtime (Step 2); and Stage IV validates and curates candidate environments through workflow replay (Step 3).

All model-assisted construction steps use `gpt-5.6-terra` for candidate selection, semantic reconstruction, state initialization, and quality assessment. API parsing, runtime compilation, execution, and transition verification are programmatic. Model-based assessments support semantic completion and curation but do not determine execution-validation outcomes.

A.1 Stage I: API Grounding and Core Selection (Step 1)

API Grounding. We collect OpenAPI and Swagger specifications from Stripe, Square, Plaid, Asana, Xero, Adyen, DocuSign, Twilio, and the APIs.guru OpenAPI Directory. Each operation is parsed into a structured representation containing its endpoint, parameter and response schemas, description, and source provenance. From these records, we construct the typed tool–entity graph G, incorporating entity relationships, tool–entity interactions, and operation dependencies.

Stateful Core Selection. We group operations within compatible source and service contexts and identify dependencies through shared entity identifiers, required arguments, and producer–consumer relationships. Candidate groups must contain at least two read and two mutation operations, structured-schema coverage of at least 0.8, stable identifiers, and mutation-to-read relationships. Model-assisted selection further considers business coherence and the availability of meaningful multi-step workflows.

For each selected candidate, we extract a connected stateful core $G_c \subseteq G$, favoring persistent entities and read–mutation–verification sequences. Core extraction targets approximately 5–7 entities and 15–48 operations, although these ranges are preferences rather than strict constraints. Missing dependencies and capabilities are recorded for semantic completion in Stage II.

A.2 Stage II: Semantic Reconstruction and Effect Specification (Step 2)

Semantic Reconstruction. Starting from G_c, we consolidate operations with compatible schemas, business semantics, and persistent-state effects. Dependency recovery identifies entity ownership and association relationships, tool–entity read and mutation relationships, and tool–tool prerequisites or verification links. Required gaps that prevent executable workflows are addressed through source-grounded semantic completion, while optional capabilities are deferred by default. Any added functionality is explicitly distinguished from operations present in the source specifications.

Effect Specification. Let $\mathcal{B}$ denote the completed semantic specification, including entity and tool schemas, dependencies, provenance, and tool-level effect specifications. For each tool u, its effect specification ψ_u identifies the target entity and expected persistent-state operation: read, create, update, delete, status transition, or association change. Read-only operations require state preservation, whereas association changes are distinguished from creating or deleting their endpoint entities.

Structural validation checks entity and tool references, dependency consistency, and effect-specification coverage. The resulting $\mathcal{B}$ provides the semantic basis for runtime compilation. Importantly, ψ_u describes an individual tool's expected effect; it is distinct from the environment-level replay checks $\Phi_{\mathcal{E}}$ and the task-specific reference contracts $\mathcal{C}_t(x)$ introduced in Section 2.2.

A.3 Stage III: State Initialization and Runtime Compilation (Step 2)

State Initialization. We generate an initial state S_0 using entity schemas, relationships, and tool dependencies from $\mathcal{B}$. The initialization process populates persistent entities with synthetic records and valid local references, providing sufficient context for subsequent stateful operations without prescribing particular training tasks. Programmatic validation checks record identifiers, entity references, required fields, and consistency with the reconstructed tool semantics.

Runtime Compilation. The completed semantic specification and initial state are compiled into a candidate executable environment:

$$\widetilde{\mathcal{E}} = \mathrm{Compile}(\mathcal{B}, S_0). \tag{5}$$

The resulting runtime exposes tool execution and persistent-state snapshot interfaces. Isolated execution starts from a copy of S_0, allowing workflows to be replayed without interference from previous runs. Static and smoke validation check tool availability, schema compatibility, initialization, and basic execution behavior before the candidate proceeds to full replay validation.

The compiled environment implements reconstructed local semantics rather than a complete reproduction of the original remote service.

A.4 Stage IV: Replay Validation and Environment Curation (Step 3)

Probe Construction. We construct environment-validation probes by enumerating bounded dependency paths over available tools, favoring workflows that perform meaningful mutations followed by verification. Model-assisted selection identifies semantically coherent probes without modifying their executable call sequences. Arguments are grounded in the initial state or earlier tool results, and candidates with unresolved arguments or failed dry runs are rejected.

These probes assess the executability and internal consistency of the reconstructed environment. They are distinct from the task-specific workflows subsequently synthesized for SFT and RL.

Replay and Transition Verification. Let $\mathcal{W}_{\mathrm{probe}}$ denote the retained validation probes. Each probe w starts from an isolated copy of the initial state, $S_0^w = S_0$. For its kth tool call a_k^w, we record the observation o_k^w, post-call state S_k^w, and state diff D_k^w:

$$(o_k^w, S_k^w) = F_{\widetilde{\mathcal{E}}}(S_{k-1}^w, a_k^w), \qquad D_k^w = \mathrm{Diff}(S_{k-1}^w, S_k^w). \tag{6}$$

For read-only operations, validation requires $S_k^w = S_{k-1}^w$. For mutating operations, the observed state diff is compared with the expected effect specified by ψ_u, including record creation, deletion, field updates, and association changes.

Where a subsequent read is available, successful execution or recognized absence provides additional verification evidence. Such readback checks do not necessarily establish exact agreement between every returned field and the modified record; mutations without direct readback are assessed through their state diffs. The resulting checks are summarized by $\Phi_{\mathcal{E}}$, which establishes execution validity and internal transition consistency for the validated workflows rather than exhaustive behavioral equivalence to the source APIs.

Environment Curation. Candidate environments undergo programmatic checks for execution success, expected effects, readback behavior, state integrity, and workflow completion. Blocking failures cause rejection, while candidates without a successfully validated probe are excluded from the retained pool.

Surviving environments are further assessed for entity coherence, tool semantic quality, and long-horizon task suitability on five-point scales. The respective aggregation weights are 0.25, 0.35, and 0.40. Environments are classified as *preferred* when all three dimensions score at least 4, *usable*

when all score at least 3, and *holdout* otherwise. These dimension-wise criteria guide final selection rather than relying solely on the aggregate score.

Of the 297 replay-validated candidates, 131 high-quality executable environments are retained. Their structural statistics and coverage metrics are reported in the main text, while the composition of the SFT and RL environment pools is detailed in Appendix D. Subsequent task-specific reference execution within these environments produces the state-transition contracts $\mathcal{C}_t(x)$ described in Appendix B.

B Implementation Details of Executable-First Synthesis of State-Transition Contracts

Following Appendix A, we detail the executable-first task synthesis pipeline in Figure 2 through four implementation phases. Phases I and II construct workflow skeletons and extract reference state-transition contracts (Step 1); Phase III generates offline and online trajectories (Step 2); and Phase IV evaluates these trajectories to derive execution-based supervision (Step 3). Reference execution is independent of solver rollout: natural-language generation may change the interaction but does not redefine the reference calls or target states.

Notation and Model Roles. We follow the main text's definitions of the executable environment $\mathcal{E}$, transition function $F_{\mathcal{E}}$, workflow skeleton $\chi(x)$, and reference contracts $\mathcal{C}(x)$. Each task x contains T workflow stages, with stage t specifying a requirement g_t and defining one task turn. Its reference calls are $A_t^* = (a_{t,\ell}^*)_{\ell=1}^{m_t}$, and its reference pre- and post-turn states are S_{t-1}^* and S_t^*. A task turn may contain multiple user–solver dialogue exchanges, indexed by j.

Workflow construction, reference execution, state capture, and execution-based scoring are programmatic. We use `gpt-5.6-terra` for offline naturalization and optional semantic quality assessment, Qwen3-235B-A22B-Instruct-2507 as the online user simulator, and Qwen3-30B-A3B-Thinking-2507 as the solver for both rollout strategies. Model-based quality assessments do not determine reference states or execution-derived scores.

B.1 Phase I: Grounding Workflow Skeletons (Step 1)

Workflow Construction. We construct executable workflow skeletons using reusable workflow-style catalogs together with each environment's tool signatures, entity schemas, effect specifications, and initial records. Operations are selected from the validated environment, and concrete arguments are grounded in existing records, entity relationships, or results from preceding calls. Thus, the reference workflow is executable before any natural-language requests are generated.

Offline workflow styles emphasize stateful operations, dependency resolution, and verification under fixed user requests. Online styles additionally support clarification, confirmation, correction, and staged information disclosure. The complete catalogs are provided in Appendix C.

Stage Grouping and Filtering. Related operations are grouped into ordered stages, allowing multiple tool calls to fulfill one requirement g_t. We enforce workflow-length and tool-count constraints, reject unsupported operations and ungrounded arguments, and filter redundant or contradictory mutations. Online skeletons additionally organize prerequisite reads, mutations, and verification reads while preserving their execution order. Read-only workflows are permitted where appropriate. Duplicate filtering removes repeated call structures, while subsequent sampling balances environment and workflow-style coverage.

B.2 Phase II: Reference Replay and Contract Extraction (Step 1)

Reference Execution. Each candidate workflow is executed in an isolated environment initialized from its saved task configuration. Let $S_{t,\ell}^*$ denote the persistent state after reference call ℓ of task turn t, with $S_{t,0}^* = S_{t-1}^*$. Reference execution satisfies

$$(o_{t,\ell}^*, S_{t,\ell}^*) = F_{\mathcal{E}}(S_{t,\ell-1}^*, a_{t,\ell}^*), \qquad S_t^* = S_{t,m_t}^*. \tag{7}$$

Here, $o^*_{t,\ell}$ is the observed tool result. Candidates containing invalid calls, execution exceptions, or explicit failure returns are rejected. Additional checks require intended mutations to affect the relevant state and read-only operations to preserve persistent state.

Unlike the environment-validation probes in Appendix A, which assess runtime executability and internal consistency, task-specific reference replay executes the particular workflow used to supervise subsequent agent trajectories.

Contract Extraction. For each successfully replayed workflow, we record the initial state, reference tool calls, and states at task-turn boundaries:

$$\mathcal{C}_t(x) = (S^*_{t-1}, A^*_t, S^*_t), \qquad \mathcal{C}(x) = (\mathcal{C}_t(x))_{t=1}^{T}. \tag{8}$$

These contracts are indexed by workflow stages rather than individual dialogue exchanges. The same reference specification can subsequently support SFT selection and RL supervision without requiring the two procedures to use identical scoring functions.

Read-Only Stages and State Normalization. Read-only stages may retrieve information without changing persistent state. For example, in an order-refund workflow, order inspection preserves $S^*_1 = S^*_0$, refund creation produces $S^*_2 \neq S^*_1$, and subsequent verification preserves $S^*_3 = S^*_2$.

State comparisons use a shared normalization function $\mathsf{N}(S)$, which converts snapshots into a JSON-compatible representation and excludes configured volatile fields, including timestamp/date fields and entries matching random hexadecimal-suffix patterns. Equality checks and execution-based selection scores operate on this normalized projection; therefore, state-matching claims apply to the compared fields rather than every raw field. Fresh reference replay is also checked against stored snapshots before offline naturalization.

B.3 Phase III: Language Realization and Solver Rollout (Step 2)

Offline Rollout Generation. The offline branch converts each stage of a validated workflow into one fixed user request q^{off}_t. The naturalizer receives the workflow skeleton, tool descriptions, relevant initial records, and style guidance, while the task initialization, reference calls, and reference states remain unchanged. Generated requests are checked for structural validity, grounded references, and consistency with the underlying workflow.

The solver processes the fixed request sequence in a freshly initialized environment. Within each task turn, it may execute multiple tool calls, observe their results, and continue until producing a user-facing response or reaching the generation limit. The solver receives tool descriptions and interaction history but has no access to reference calls or target states.

Online Rollout Generation. The online branch instead generates user requests adaptively during interaction. Let $H_{t,j-1}$ denote the user-visible dialogue history preceding exchange j of task turn t, and let $I_{t,j}(x)$ denote the task information provided to the user simulator $\mathcal{U}$. The next user request is

$$q^{\text{on}}_{t,j} = \mathcal{U}(I_{t,j}(x), H_{t,j-1}). \tag{9}$$

The user simulator may clarify missing information, confirm an action, correct an assumption, or provide follow-up information before advancing to the next workflow stage. Accordingly, multiple dialogue exchanges may address the same requirement g_t. The simulator receives workflow-level goals and active-stage information, while reference calls and target states remain hidden from the acting solver.

A stage controller estimates workflow progression from the solver's emitted tool calls rather than using successful state verification as its advancement criterion. Thus, stage progression guides user interaction but does not independently establish execution correctness.

Task Turns and Dialogue Exchanges. A task turn corresponds to one workflow stage, not one user–solver exchange. For a task turn containing n_t exchanges, let $\widehat{A}_{t,j}$ denote the solver calls made during exchange j. When exchanges can be assigned to their corresponding task turns, the turn-level call sequence is

$$\widehat{A}_t = \widehat{A}_{t,1} \parallel \cdots \parallel \widehat{A}_{t,n_t}, \tag{10}$$

where $\|$ denotes chronological concatenation and clarification-only exchanges may contain no tool calls.

In online SFT, the number of exchanges within a workflow stage depends on the interaction, and inferred stage progression does not necessarily provide precise reference task-turn boundaries. Consequently, assigning every dialogue exchange to a reference turn can be ambiguous. Online SFT therefore uses trajectory-level selection based on pooled tool calls and the final state, as detailed in Phase IV. This scoring choice is specific to online SFT; RL retains the task-turn reward defined in Section 2.3.

The two rollout strategies use the same executable-first construction procedure, although their datasets need not contain paired instances of every workflow skeleton.

B.4 Phase IV: Execution-Derived Supervision and Data Selection (Step 3)

SFT Trajectory Selection. We evaluate generated trajectories against the reference contracts from Step 1. Both SFT branches use tool-call recall and state coverage, but offline SFT aggregates task-turn scores, whereas online SFT directly evaluates pooled calls and the final state. These selection scores differ from the RL reward in Section 2.3, which uses multiset tool-call F1 and transition-aware state verification.

Let $\rho(B, A^*)$ denote multiset tool-call recall for executed calls B against reference calls A^*. Unlike tool-call F1, recall measures coverage of reference calls without an equivalent penalty for additional executed calls.

Offline SFT Selection. The offline scorer evaluates each task turn against its reference contract. Let $\bar{S}$ denote the flattened field map of $\mathsf{N}(S)$, and define the expected changed-field set

$$D_t^{\mathrm{sel}} = \left\{ p : \bar{S}_{t-1}^*(p) \neq \bar{S}_t^*(p) \right\}, \tag{11}$$

where additions and deletions are included, with absent paths represented by a distinguished missing value. Changed-field coverage is

$$\eta_t = \begin{cases} \dfrac{1}{|D_t^{\mathrm{sel}}|} \displaystyle\sum_{p \in D_t^{\mathrm{sel}}} \mathbb{I}\left[\bar{S}_t(p) = \bar{S}_t^*(p)\right], & |D_t^{\mathrm{sel}}| > 0, \\ \mathbb{I}\left[\mathsf{N}(S_t) = \mathsf{N}(S_t^*)\right], & |D_t^{\mathrm{sel}}| = 0. \end{cases} \tag{12}$$

With $\rho_t = \rho(\widehat{A}_t, A_t^*)$, the task-turn selection score is

$$v_t^{\mathrm{off}} = \begin{cases} 0, & \text{if execution fails,} \\ \rho_t \, \mathbb{I}\left[\mathsf{N}(S_t) = \mathsf{N}(S_t^*)\right], & \text{if the reference turn preserves state,} \\ \dfrac{1}{2}\left(\rho_t + \eta_t\right), & \text{otherwise.} \end{cases} \tag{13}$$

The trajectory-level selection score averages all T reference task turns:

$$V_{\mathrm{SFT}}^{\mathrm{off}}(\tau; x) = \frac{1}{T} \sum_{t=1}^{T} v_t^{\mathrm{off}}. \tag{14}$$

Missing or unaligned turns receive zero and remain in the average. State-preserving turns require normalized-state equality to receive tool-call credit. For mutating turns, changed-field coverage measures intended changes without imposing the additional preservation guard used by the default RL reward.

Online SFT Selection. Online user requests are generated adaptively, so clarification, confirmation, and follow-up exchanges can produce variable-length interactions within a workflow stage. Since the stage controller estimates progression from tool calls rather than exact state verification, intermediate task-turn boundaries may be ambiguous. Applying offline turn-level scoring would therefore require potentially unreliable alignment.

Instead, online SFT selection evaluates the complete trajectory using pooled tool-call recall and final-state coverage. Let

$$A^* = A_1^* \,\|\, \cdots \,\|\, A_T^* \tag{15}$$

denote the concatenated reference calls, $\widehat{A}$ all executed calls in their logged order, and S_{end} the observed final state. Let D^+ contain paths present in the normalized reference final state whose values differ from the normalized initial state. We compute

$$\eta^{\text{on}} = \begin{cases} \dfrac{1}{|D^+|} \displaystyle\sum_{p \in D^+} \mathbb{I}\left[\bar{S}_{\text{end}}(p) = \bar{S}_T^*(p)\right], & |D^+| > 0, \\ 1, & |D^+| = 0, \end{cases} \qquad V_{\text{SFT}}^{\text{on}}(\tau; x) = \frac{1}{2}\left(\rho(\widehat{A}, A^*) + \eta^{\text{on}}\right). \tag{16}$$

Absent entries are treated as null during online path lookup. The coverage calculation excludes paths deleted from the reference final state and does not constrain unchanged fields. This design avoids uncertain intermediate alignment but does not directly verify individual state transitions or distinguish all execution paths that reach the same final state.

Data Curation and Sampling. Both SFT selection procedures record execution errors. We retain trajectories with selection scores strictly above 0.75 under their respective metrics. Additional structural checks assess message validity, tool-call and response consistency, repeated calls, and construction-detail leakage. When enabled, an optional GPT-5.6-based judge assesses goal consistency, grounding, interaction coherence, tool-use quality, response quality, and reasoning quality on five-point scales, with a pass/borderline/fail verdict. These assessments do not alter execution-derived scores.

We rank candidates using their corresponding offline or online scores and balance environment and workflow-style coverage within sampling quotas. The final SFT dataset contains 900 offline and 2,700 online tasks, as reported in Section 3.2.

RL Supervision and Policy-Conditioned Task Selection. The same reference contracts supply task-turn supervision for RL, independently of the SFT selection scores: task-turn rewards follow Eq. 4 and are averaged into a trajectory-level reward $R_{\text{RL}}(\tau; x)$ for GRPO (Section 2.3). For each training model, approximately 15,000 candidate tasks are rolled out with its RL initialization checkpoint and scored using R_{RL}, from which 2,400 tasks are selected through stratified sampling with target proportions of 5%, 20%, 45%, and 30% for the score intervals $[0, 0.25)$, $[0.25, 0.5)$, $[0.5, 0.75)$, and $[0.75, 1]$, while balancing environment and workflow-style coverage. RL tasks are therefore selected separately for each model, whereas the SFT dataset is shared across model scales; training configurations and the two tool-only reward exceptions are documented in Appendix E.

C Workflow Style Catalogs

We use workflow-style catalogs to construct executable task skeletons in Step 1 of Figure 2. The offline and online SFT catalogs each contain 16 distinct styles, yielding 32 catalog entries. Offline styles emphasize stateful operations, dependency resolution, and verification under fixed user requests, while online styles additionally support adaptive interactions such as clarification, confirmation, and user correction. The RL catalog reuses all 16 offline styles and introduces four extensions involving intent revision, confirmation before terminal actions, multi-target verification, and corrective operations. These styles define reusable generation patterns rather than fixed task instances; each can be instantiated with environment-specific tools, entities, and dependencies before task-specific reference replay.

Table 5: The 32 workflow styles used for offline and online SFT task synthesis. Styles specify reusable operational and interaction patterns rather than fixed trajectories.

#	Workflow style	Operational pattern
Offline styles (fixed user requests)		

Continued on next page

Table 5 (continued)

#	Workflow style	Operational pattern
1	`inspect_apply_verify`	Inspect context, modify a record, and verify the resulting state.
2	`resolve_reference_apply`	Resolve a name or reference to an ID, modify the target, and verify.
3	`exact_argument_binding`	Inspect the target, bind required arguments, and verify the exact state.
4	`stateful_context_binding`	Establish context, execute a context-dependent operation, and verify.
5	`ordered_step_execution`	Inspect prerequisites, resolve dependencies, execute in order, and verify.
6	`dependency_guided_operation`	Inspect upstream entities and dependencies, modify a downstream target, and verify.
7	`state_conditioned_operation`	Inspect state, select an execution branch, modify, and verify its outcome.
8	`schema_guided_operation_choice`	Inspect schema constraints, select the appropriate operation, and verify.
9	`create_or_attach_verify`	Resolve parent context, create or associate a record, and verify.
10	`check_terminal_action_verify`	Inspect the target, perform a terminal action, and verify its final state.
11	`parent_child_change`	Resolve parent and child objects, update the child or relation, and verify.
12	`candidate_comparison_action`	Compare candidates, act on the selected target, and verify.
13	`related_record_coordination`	Inspect shared context, coordinate operations on related records, and verify.
14	`association_management`	Inspect, create, remove, or assign an association, and verify its state.
15	`configuration_reconciliation`	Inspect configuration and dependencies, update settings, and verify.
16	`policy_constrained_action`	Inspect policy-relevant state (e.g., status, role, or quota), choose an allowed action, and verify.
	Online styles (adaptive user interactions)	
17	`missing_info_clarify_then_execute`	Clarify missing information before executing the requested operation.
18	`inspect_confirm_mutate`	Inspect the current record, confirm the intended change, and mutate it.
19	`ambiguous_target_disambiguation`	Resolve an ambiguous target among multiple candidates before mutation.
20	`user_correction_after_wrong_assumption`	Revise the intended operation after the user corrects an initial assumption.
21	`scope_reduction_before_mutation`	Narrow the affected records or fields before changing state.
22	`plan_change_after_review`	Adjust the intended operation after reviewing newly observed context.
23	`option_change_same_context`	Change one option while preserving the same target and task context.
24	`multi_intent_clarify_one_at_a_time`	Clarify and execute multiple related user intents sequentially.
25	`multi_segment_change_after_review`	Inspect related records and apply coordinated changes across segments.
26	`read_only_answer_after_lookup`	Retrieve the relevant record or policy and answer without mutation.

Continued on next page

Table 5 (continued)

#	Workflow style	Operational pattern
27	`specific_fact_answer_after_lookup`	Retrieve a specific fact from the environment and answer directly.
28	`return_exchange_variant_check`	Inspect eligibility and item context before selecting a return or exchange path.
29	`cross_turn_result_reference`	Use an earlier lookup result to identify the target of a subsequent action.
30	`coordinated_distinct_record_actions`	Coordinate related actions on distinct records within one business objective.
31	`policy_guarded_action_selection`	Inspect policy or eligibility constraints before selecting an allowed action.
32	`composed_alternative_for_unavailable_action`	Compose supported operations when the requested direct capability is unavailable.

The RL catalog contains 20 styles: the 16 offline styles in Table 5 and the four additional patterns in Table 6. The extensions emphasize multi-stage state changes and verification while retaining the same executable-first construction and reference-replay requirements as the SFT workflows.

Table 6: Four RL-specific workflow extensions. Together with the 16 offline styles in Table 5, they form the 20-style RL catalog.

#	Workflow style	Operational pattern
17	`cross_turn_intent_revision`	Perform an initial update, verify its state, revise the target according to subsequent user intent, and verify again.
18	`confirm_before_irreversible_action`	Inspect the target and applicable constraints, obtain confirmation before a terminal or irreversible-style operation, and verify its outcome.
19	`batch_multi_target_verification`	Modify multiple targets and independently verify each resulting state.
20	`repair_after_verification`	Identify a state discrepancy through verification, execute a supported corrective operation, and verify the repaired state.

C.1 Task Synthesis and Rollout Configuration

The RL skeleton builder supports 3–5 workflow stages and at most three reference tool calls per stage. Its total reference-call ranges are 3–7, 4–9, and 5–12 for three-, four-, and five-stage workflows, respectively. These limits apply to the constructed reference skeletons rather than to the number of tool calls subsequently generated by the solver.

For SFT, the sampler selects 3,600 tasks, targeting an offline-to-online ratio of $1:3$, a medium-to-high source-pool ratio of $1:1$, and three-, four-, and five-turn proportions of $2:2:1$, subject to candidate availability. Here, turn counts refer to workflow-stage requirements, not the number of individual online dialogue exchanges. Offline and online tasks are sampled from their respective catalogs and need not form paired instances of the same skeleton. The final dataset contains 900 offline and 2,700 online tasks, as reported in Section 3.2.

The offline naturalizer uses GPT-5.6 (`gpt-5.6-terra`), while both SFT rollout strategies use Qwen3-30B-A3B-Thinking-2507 as the solver. Online rollouts use Qwen3-235B-A22B-Instruct-2507 as the user agent. The naturalizer samples its default temperature from $[0.35, 0.75]$ with a 4,096-token output limit. The offline rollout script defaults to temperature 0.1, an 8,192-token output limit, and 15 solver generation steps per fixed user request.

The stage-aware online scripts use a default temperature of 0.2 for both the solver and user simulator, with output limits of 4,096 solver tokens and 1,024 user tokens. Online rollout length is governed by workflow progression rather than a fixed number of user exchanges, allowing multiple rounds

Table 7: Environment composition of the SFT and RL training pools. Shared is the overlap between offline and online SFT; Overlap denotes RL environments also used for SFT; New denotes RL environments absent from SFT. Overall Unique counts distinct environments across both training stages.

SFT				**RL**			**Overall**
Offline	Online	Shared	Unique	Total	Overlap	New	Unique
50	80	30	**100**	60	32	28	**128**

of clarification, confirmation, and follow-up within each stage. Interactions terminate upon workflow completion, inability to make further progress, or exhaustion of the configured computational budget. Each user exchange permits at most eight solver generation steps. The selected runner, model identifiers, and final selection manifest determine the configuration and composition of each experimental run.

D Environment Pool Composition and Task Diversity

Environment Pools. Our construction pipeline retains 131 executable environments, 128 of which are used for training. Table 7 summarizes the environment composition across offline SFT, online SFT, and RL. The SFT pool contains 100 unique environments, while the RL pool contains 60, including 32 shared with SFT. Specifically, 26 RL environments overlap with offline SFT, 24 with online SFT, and 18 with both.

Semantic Diversity. The retained environments span commerce, finance, communication, collaboration, software development, data analytics, infrastructure, and specialized business services, providing heterogeneous tool interfaces, persistent entities, and state dependencies.

Workflow Diversity. Environment coverage is complemented by diversity in workflow structure. Depending on the environment and selected style, a task may require entity resolution, argument binding, prerequisite inspection, ordered execution, coordinated updates, adaptive clarification, or verification of persistent-state changes. The two SFT catalogs contain 32 styles in total, while the RL catalog contains 20 styles, including four RL-specific extensions. These patterns can be instantiated with different tools, entities, and dependencies across application domains, providing variation in both execution requirements and interaction structure. Complete workflow definitions are provided in Appendix C.

E Training Configurations

Table 8 summarizes the training configurations for SFT and RL. Unless otherwise specified, the settings are shared across Qwen3-1.7B, Qwen3-4B, and Qwen3-8B, all trained in thinking mode. VESTA applies SFT to the shared 3,600-task dataset and then initializes GRPO from the corresponding SFT checkpoint, using 2,400 policy-conditioned RL tasks selected separately for each model. The direct RL experiments instead initialize training from the corresponding base checkpoints. Model-specific settings for Qwen3-8B are indicated in parentheses.

F Evaluation Settings

We evaluate the three Qwen3 model scales on BFCL-v3 Multi-Turn, ACEBench-Agent, and τ^2-Bench using their respective benchmark protocols and metrics. For Qwen3 compatibility, we adapt tool-call formatting and extraction where necessary without modifying the underlying evaluation tasks or scoring criteria. The configurations below apply to the evaluated agents and, where applicable, their user simulators.

Table 8: Training configurations for SFT and RL. Model-specific settings for Qwen3-8B are shown in parentheses.

Supervised fine-tuning			
Learning rate	1×10^{-6}	Training epochs	3
LR schedule	Constant after warmup	Warmup ratio	5%
Batch size / GPU	2 (8B: 1)	Accumulation steps	8
Sequence limit	32K	Distributed training	DeepSpeed ZeRO-2
Reinforcement learning			
Algorithm	GRPO	Training steps	80
Learning rate	2×10^{-6}	KL coefficient	0.02
Training batch size	32	PPO mini-batch size	16
Rollouts / prompt	8	Temperature	0.8
Prompt / response limit	12K / 12K	Tool / state weights	0.5 / 0.5
GPUs	4 (8B: 8)	Distributed training	FSDP

Note. Both training stages use BF16 precision, and all sequence and generation limits are measured in tokens. The tool/state weights apply to mutating turns under the default combined reward in Eq. 4; read-only turns award tool-call credit only when the state-preservation guard is satisfied. Two RL tasks instead use tool-only rewards on all turns. For every reward configuration, task-turn rewards are averaged into a single trajectory-level reward for GRPO.

BFCL-MT. We evaluate the Base, Missing Functions, Missing Parameters, and Long Context subsets, each containing 200 test cases. The evaluated agent uses temperature 0, a context window of 40,960 tokens, and an output budget of up to 4,096 tokens per response, subject to the remaining context capacity. BFCL-MT uses benchmark-defined user turns rather than an LLM-based user simulator, with tool calls executed in the benchmark environment. We use a Qwen3-formatted tool-use prompt and extract JSON tool-call payloads enclosed in `<tool_call>` tags from raw model completions.

ACEBench-Agent. We evaluate the English Agent Multi-step and Agent Multi-turn subsets, containing 20 and 30 test cases, respectively. For both subsets, the evaluated agent uses temperature 0, top-p 1.0, and a maximum of 2,048 output tokens per response, with an interaction limit of 40 turns. Agent Multi-step uses a fixed initial user task without an LLM-based user simulator. For Agent Multi-turn, we use Qwen3-235B-A22B-Instruct-2507 as the user simulator, with temperature 0, top-p 1.0, and a maximum of 4,096 output tokens per response. The simulator follows the benchmark's official domain-specific user prompts to generate the initial request and subsequent replies. We retain ACEBench's native `[ApiName(key=value)]` tool-call format and use an AST-based parser to extract tool calls.

τ^2-Bench. We evaluate the Airline, Retail, and Telecom domains on 50, 114, and 114 tasks, respectively, using one trial per task and a random seed of 300. The evaluated agent uses temperature 0, a maximum of 4,096 output tokens per step, and an interaction limit of 50 steps. We use the benchmark's official `UserSimulator`, backed by Qwen3-235B-A22B-Instruct-2507, with temperature 0 and a maximum of 1,024 output tokens per step. Its prompt combines the official user-simulation guidelines with the task-specific scenario and persona. Tool calls are processed through the OpenAI-compatible tools interface using vLLM's `qwen3_xml` parser. When structured tool-call fields are absent, an XML fallback extracts calls from the generated text. Task success is determined by the benchmark environment's reward.

G Prompt Templates for Trajectory Generation

We present the principal prompts used for offline question naturalization (Appendix G.1), online user simulation (Appendix G.2), and solver rollout (Appendix G.3), with dynamic inputs shown as placeholders. The three roles differ in the information they can access (Table 9): the naturalizer and user simulator may use workflow-level reference information for task construction, whereas

the solver receives only the environment description, tool schemas, user messages, and actual tool observations. Reference calls, expected states, hidden stage labels, and evaluation scores are never exposed to the solver.

Table 9: Roles involved in trajectory generation and the reference information available to each.

Role	Model	Reference information	Output
Offline naturalizer	`GPT-5.6-terra`	Fixed reference calls and environment context	One user request per task turn
Online user simulator	`Qwen3-235B-A22B-Instruct-2507`	Active-stage goals and target references	Next user message
Solver (SFT rollouts)	`Qwen3-30B-A3B-Thinking-2507`	None	Tool calls and user-facing responses
Solver (RL training)	`Qwen3-1.7B/4B/8B`	None	Tool calls and user-facing responses

G.1 Offline Question Naturalization

The offline naturalizer converts a replay-validated workflow skeleton into one natural user request per task turn. Its tool-call plan is fixed before naturalization: the model may adjust wording and conversational style, but cannot modify the task objective, tool order, arguments, number of task turns, or expected state transitions.

Inputs. The naturalizer receives nine inputs (Table 10). The first four specify the fixed reference workflow, including its conversational style, replay-validated tool calls, required tools, and identifier-resolution guidance. The remaining five provide environment-specific grounding through available entities, environment descriptions, tool schemas, execution constraints, and the initial database snapshot. The required number of user messages, `turn_count`, is derived from the fixed workflow. These inputs guide natural-language generation without authorizing changes to the reference execution plan.

Table 10: Inputs to the offline naturalizer.

Placeholder	Content
Reference workflow	
`style_description`	Intended workflow pattern and conversational framing.
`answers`	Exact replay-validated tool calls and arguments for each task turn.
`required_tools`	Tool names for each task turn, used as an additional consistency constraint.
`id_resolution_guidance`	Existing IDs that must be mentioned explicitly for direct reads, and those that can instead be resolved through the prescribed lookup operations.
Environment grounding	
`available_entities`	Available entities, relationships, and relevant records.
`env_introduction`	Business context and supported capabilities.
`tool_descriptions`	Available operations, parameter schemas, and required fields.
`env_rule`	Applicable constraints, including valid state transitions, relationships, and permissions.
`init_config`	Truncated snapshot of the initial database.

Offline Naturalization: Input Variables

```
Workflow style                : {style_description}
Fixed tool calls by turn      : {answers}
Required tool names by turn   : {required_tools}
ID-resolution guidance        : {id_resolution_guidance}
Available entities            : {available_entities}
Environment introduction      : {env_introduction}
Tool descriptions and schemas : {tool_descriptions}
Environment constraints       : {env_rule}
Initial database snapshot     : {init_config}
```

Offline Naturalization: Prompt

```
- Write the user messages for one realistic multi-turn tool-use task.

- The tool-call plan below is already fixed and replay-validated. Do
    NOT change the tools, order, arguments, turn count, or reward
    type. Your job is only to write natural user messages that
    justify exactly those tool calls.

- Return exactly {turn_count} user messages, one per turn, matching
    the fixed tool-call plan. Keep each message concise and natural,
    normally no more than three sentences. The fixed call plan is
    authoritative: do not add, omit, reorder, reinterpret, or repair
    calls.

- Keep one clear business thread across the conversation. Use one
    anchor record and directly related records. Make adjacent turns
    feel connected, but do not claim that a fixed call caused or
    proved a result that is not established by the environment.

- Mention an existing ID once when a direct read requires it;
    otherwise let the fixed lookup expose the ID and refer to the
    record naturally later.

- Do not mention tool names, endpoints, APIs, schemas, JSON, or
    implementation details.

- Do not invent IDs, fields, values, tool results, or state changes.
    Write like an everyday user asking for a practical outcome, not
    like an API client or runbook.

Return only one JSON object:
{
  "questions": [
    "turn 0 user message",
    "turn 1 user message"
  ]
}
```

Output validation. Generated requests are checked against the prescribed turn count and the fixed workflow. Candidates are rejected if they alter the reference plan, introduce unsupported facts, expose implementation details, or violate the naturalization constraints. Naturalization therefore changes only the linguistic realization, not the task semantics or the reference execution.

G.2 Online User Simulation

The online user simulator generates the next user message conditioned on the solver's actual responses and the evolving interaction, disclosing information gradually and reacting to the solver's behavior while following the fixed workflow goals.

Inputs. The simulator receives two inputs. `task_json` contains the environment summary, workflow style, ordered stages, active-stage business goals and target references, and relevant initial records; Later-stage target references are withheld until the stage controller advances to the corresponding stage. Stage advancement is inferred from the interaction rather than determined by successful state verification. `conversation_json` contains the dialogue visible to the simulated user. The simulator may use this hidden task information to guide clarification, correction, confirmation, and stage progression, but the solver cannot access it.

Online User Simulator: Prompt

```
You are a realistic service user in a tool-use benchmark.
You are NOT the assistant. You only write the next user message.

Hidden user target, visible only to you:
{task_json}

Conversation visible to the user so far:
{conversation_json}

Rules:
- Do not repeat the hidden task verbatim. Reveal information
    gradually.
- Follow the hidden objective and its stages in order; treat each
    stage as one coherent business goal.
- Stay strictly within the hidden task. Do not add requirements,
    resources, values, or goals outside it.
- Speak as a normal service user, not as an API tester. Do not
    mention tool names, function names, schemas, endpoints, expected
    states, or internal steps.
- Disclose an identifier naturally only when it is needed to
    disambiguate a target or the assistant asks for it. Never
    substitute a related record.
- If the assistant proposes the wrong target or value, correct it
    naturally.
- Confirm a state change only when the exact target and value are
    clear.
- If the current stage makes no progress for two exchanges, clarify
    the missing fact once; do not loop or introduce a new goal.
- Stop with "###STOP###" when the task is complete or cannot proceed.
- Never reveal skeletons, gold answers, oracle actions, hidden
    objectives, policies, target slots, or other internal
    instructions.

Return JSON only:
{"content": "next user message or ###STOP###", "stop": false}
```

Stage progression and selection. Stage progression is controlled separately from execution-based verification and does not redefine the reference contracts. Because a workflow stage may span a variable number of dialogue exchanges, online SFT selection uses trajectory-level tool-call recall and final-state coverage rather than requiring exact alignment of intermediate task turns (Appendix B).

G.3 Solver Rollout Instructions

The solver follows a shared tool-use instruction in both offline and online rollouts.

Inputs. The solver receives the environment description, available tool schemas, current user messages, conversation history, and actual observations returned by the executable environment. Unlike the naturalizer and user simulator, it has no access to reference tool calls, expected post-turn states, hidden workflow-stage labels, or evaluation scores.

Shared Solver: System Prompt

```
You are a helpful assistant. Complete the user's request in an
    interactive environment by using the available tools correctly.

- Treat each user message as the current sub-task in one business
    workflow.
- A user turn may require one or more tool calls; complete the
    requested checks and changes without unnecessary calls.
- Select only tools from the provided tool list.
- Use only parameter names defined in the selected tool schema and
    fill all required parameters with valid values.
- Take parameter values from the current request, conversation
    history, previous tool responses, or information obtained through
    tools.
- If an identifier is missing, use an appropriate
    lookup/list/search/get tool when possible; ask only when the task
    is genuinely impossible to resolve.
- Do not perform state-changing actions unless requested or necessary
    for the current user turn. Verify relevant records before
    sensitive mutations.
- Treat tool responses as authoritative. Do not replace returned IDs
    or fields with guessed or reformatted values.
- Do not repeat a lookup or user-facing question when no new
    information or state change has occurred.
- After the current task is complete, respond in plain text with the
    checked fact or confirmation. Do not answer finally while tool
    use is still needed.
```

Online RL Protocol. Unlike the stage-aware user simulation used for online SFT data generation, online RL follows an explicit task-turn execution protocol. The acting policy processes the current user sub-task, waits for actual observations between dependent tool calls, and closes the logical turn with a plain-text response. The environment then supplies the next sub-task.

Online RL: Additional Instructions

```
Online RL interaction:
- Only the current user sub-task is shown. A plain-text assistant
    response closes that logical turn; the environment then supplies
    the next sub-task.
- A logical turn may span multiple assistant responses and tool
    observations.
- One assistant response may contain one or more complete independent
    tool calls. If a later call depends on an earlier result, wait
    for the real tool response before emitting it.
- Tool observations are supplied by the environment in the SFT
    observation format and appear in the next context as
    <tool_response> blocks.
- Never generate, simulate, or repeat <tool_response>. The
    environment supplies the actual result.
- Every <tool_call> must be complete JSON containing exactly `name`
    and `arguments`, using the selected tool's schema.
- Do not emit the internal chat_with_user tool. Use plain text to
    close the current sub-task.
```

Both configurations share the same tool-use policy and schema requirements, and neither exposes reference calls, target states, or execution-derived scores to the solver.

H CASE STUDIES

We present two representative cases from completed, replay-validated rollout records to illustrate offline fixed-request naturalization and online stage-guided user simulation. Both follow the executable-first construction procedure but differ in how validated workflow skeletons are realized as user interactions and how the resulting trajectories are selected for SFT. Tool calls and observations are abbreviated where appropriate, while preserving relevant arguments, state transitions, and scoring evidence.

We denote the offline and online trajectory-level SFT selection scores by $V_{\mathrm{SFT}}^{\mathrm{off}}$ and $V_{\mathrm{SFT}}^{\mathrm{on}}$, respectively, following Eqs. 14 and 16. A score of 1 indicates maximal agreement under the corresponding selection metric. These scores are distinct from the RL reward R_{RL} defined in Section 2.3. Table 11 summarizes the two cases.

Table 11: Overview of the offline and online case studies.

	Case 1: Offline	**Case 2: Online**
Domain	Repository and pull-request governance	Voice connector and proxy-session administration
Workflow	Three fixed task turns	Four stages with adaptive interaction
User messages	3	6, followed by termination
Tool calls	8 executed / 8 reference	7 executed / 6 reference
SFT selection	Turn-level scoring and aggregation	Pooled tool recall and final-state coverage

H.1 CASE 1: OFFLINE FIXED-REQUEST NATURALIZATION

The user prepares a release-governance handoff involving pull requests from two repositories. The validated skeleton contains three task turns: inspecting the relevant pull requests, updating the review status of the audit-retention request, and verifying the final state. Before solver rollout, the naturalizer converts each stage into one fixed user request without changing the prescribed tool calls, arguments, or reference state transitions.

Turn 1: Compare and inspect. The first turn establishes the pull-request identities and their merge states, providing the context for the subsequent review-status update.

Case 1, Turn 1: Compare and Inspect

```
User:
  I am assembling the release-governance handoff for northstar-labs.
    Compare the merge state of pr_northstar_atlas_138 and
    pr_northstar_atlas_142, and pull the full record for
    pr_northstar_atlas_138 so I can review the audit-retention change
    in detail.

Tool calls:
  check_pull_request_merge_state(
    pull_request_id="pr_northstar_atlas_138", owner="northstar-labs")
  check_pull_request_merge_state(
    pull_request_id="pr_northstar_atlas_142", owner="northstar-labs")
  retrieve_pull_request(
    pull_request_id="pr_northstar_atlas_138", owner="northstar-labs")

Observations:
  pr_northstar_atlas_138:
    repository = northstar-labs/atlas-api
```

```
    title = Refine audit event retention
    status = open, mergeable = false, merged = false

  pr_northstar_atlas_142:
    repository = northstar-labs/atlas-api
    title = Add regional quota checks
    status = open, mergeable = true, merged = false

State:
  No persistent-state changes.
```

Turn 2: Apply the review transition. After identifying the target pull request, the solver updates its review status and retrieves two related records. The read-only operations preserve the surrounding handoff context.

Case 1, Turn 2: Review Transition

```
User:
  For that handoff, move the audit-retention pull request to
    in_review.
  Also pull the full records for the Northstar quota-check request you
  just examined and Copperfield's incident-filter request,
  pr_copperfield_console_57 under copperfield.

Tool calls:
  update_pull_request(
    pull_request_id="pr_northstar_atlas_138",
    owner="northstar-labs", status="in_review")
  retrieve_pull_request(
    pull_request_id="pr_northstar_atlas_142", owner="northstar-labs")
  retrieve_pull_request(
    pull_request_id="pr_copperfield_console_57", owner="copperfield")

State transition:
  pr_northstar_atlas_138: status open -> in_review
    mergeable = false, merged = false

Observations:
  pr_northstar_atlas_142:
    title = Add regional quota checks
    status = open, mergeable = true, merged = false

  pr_copperfield_console_57:
    repository = copperfield/ops-console
    title = Improve incident timeline filters
    status = open, mergeable = true, merged = false
```

Turn 3: Verify the final state. The final turn rechecks the audit-retention request and the unrelated Copperfield request, confirming that the intended status transition persists.

Case 1, Turn 3: Final Verification

```
User:
  Before I pass along the handoff, recheck the merge state of the
  audit-retention pull request and the Copperfield incident-filter
    request.

Tool calls:
  check_pull_request_merge_state(
```

```
    pull_request_id="pr_northstar_atlas_138", owner="northstar-labs")
  check_pull_request_merge_state(
    pull_request_id="pr_copperfield_console_57", owner="copperfield")

Observations:
  pr_northstar_atlas_138:
    status = in_review, mergeable = false, merged = false

  pr_copperfield_console_57:
    status = open, mergeable = true, merged = false

State:
  No further persistent-state changes.
```

Offline SFT Selection. The fixed request sequence provides explicit task-turn boundaries, allowing each turn to be evaluated against its corresponding reference contract. Following Eq. 13, read-only turns receive tool-call recall only when the normalized state matches the reference, while mutating turns combine tool-call recall and changed-field coverage. The resulting turn-level selection scores are averaged to obtain the trajectory-level offline SFT selection score $V_{\mathrm{SFT}}^{\mathrm{off}}(\tau; x)$ (Eq. 14).

Case 1: Offline SFT Selection

```
Reference calls              : 8
Executed calls               : 8
Tool-call recall             : 1.0
Turn selection scores        : [1.0, 1.0, 1.0]
Trajectory selection score   : V_SFTˆoff = 1.0
Mutating-turn state coverage : 1.0
Execution errors             : none
```

The recorded execution recovers all reference calls, realizes the intended status change, and preserves state during the read-only turns. This case illustrates how offline naturalization produces coherent user requests while retaining the fixed workflow structure needed for turn-aligned SFT selection.

H.2 Case 2: Online Stage-Guided User Simulation

The second case concerns an Atlanta voice connector and its proxy sessions. The validated workflow contains four stages, but the user simulator generates six messages as the interaction unfolds. Unlike offline naturalization, the simulator reveals identifiers and requirements incrementally, requests confirmation before mutations, and adapts subsequent messages to the solver's responses. The solver receives neither the hidden stage plan nor the reference calls and target states.

Stage 1: Establish the connector. The simulator first provides the connector identifier while withholding the dependent proxy-settings and session identifiers until the connector context has been established.

Case 2, Stage 1: Establish the Connector

```
User:
  I need to check the proxy settings for the voice connector used in
    our
  Atlanta office, reference vc-atl-01, and also look up a specific
    proxy
  session under that connector.

Tool call:
```

```
  get_voice_connector(voice_connector_id="vc-atl-01")

Observation:
  name = Atlanta Voice Gateway
  aws_region = us-east-1
  require_encryption = true
```

Stage 2: Disclose dependent references. After the connector is identified, the simulator reveals the proxy-settings and existing-session identifiers. The solver then retrieves the records needed for the subsequent updates.

Case 2, Stage 2: Dependent References

```
User:
  The proxy settings are under the reference pys-atl-01. For the proxy
  session, I need to look up the one with ID ps-atl-1001 under the
    same
  voice connector.

Tool calls:
  inspect_proxy_settings(
    proxy_settings_id="pys-atl-01",
    voice_connector_id="vc-atl-01")

  get_proxy_session(
    proxy_session_id="ps-atl-1001",
    voice_connector_id="vc-atl-01")

Observations:
  pys-atl-01:
    default_session_expiry_minutes = 30
    disabled = false
    fall_back_phone_number = +14045550801
    phone_number_countries = [US, CA]

  ps-atl-1001:
    capabilities = [audio, sms]
    participants = [+14045550121, +14045550122]
    expiry_minutes = 30
    status = active
```

Stage 3: Confirmation-gated mutation. The simulator requests an expiry change and the creation of a new session, then confirms the exact target and value before the solver performs the state-changing operations. This stage spans two user messages.

Case 2, Stage 3: Confirmation-Gated Mutation

```
User:
  Thanks for pulling those details. Now, I need to update the proxy
  settings for the Atlanta voice connector to extend the default
    session
  expiry to 31 minutes. After that, I'd like to create a new proxy
  session under the same connector with the status set to in_review.

User (confirmation):
  Yes, confirm the change to update the default session expiry to
  31 minutes. Once that's done, let's create the new proxy session
  under the same voice connector with the status set to in_review.
```

```
Tool calls:
  configure_proxy_settings(
    proxy_settings_id="pys-atl-01",
    voice_connector_id="vc-atl-01",
    default_session_expiry_minutes=31)

  add_proxy_session(
    voice_connector_id="vc-atl-01",
    proxy_settings_id="pys-atl-01",
    status="in_review")

State transitions:
  pys-atl-01:
    default_session_expiry_minutes: 30 -> 31

  New session:
    proxy_session_id = proxy_session_0007
    status = in_review
```

Stage 4: Update and verify the existing session. The simulator subsequently requests an update to the existing proxy session and follows up with an explicit verification request. The solver performs the mutation and retrieves the updated record.

Case 2, Stage 4: Update and Verify

```
User:
  Now I need to update the expiry for the existing proxy session
  ps-atl-1001 under voice connector vc-atl-01 to 31 minutes.

Tool call:
  update_proxy_session(
    proxy_session_id="ps-atl-1001",
    voice_connector_id="vc-atl-01",
    expiry_minutes=31)

Observation:
  expiry_minutes = 31
  status = active
  capabilities = [audio, sms, updated]
  participants = [+14045550121, +14045550122]

User (follow-up):
  Can you also retrieve the updated details for the proxy session
  ps-atl-1001 under voice connector vc-atl-01 to confirm the changes?

Tool call:
  get_proxy_session(
    proxy_session_id="ps-atl-1001",
    voice_connector_id="vc-atl-01")

Observation:
  expiry_minutes = 31
  status = active
  capabilities = [audio, sms, updated]
  participants = [+14045550121, +14045550122]
```

Online SFT Selection. Because the four workflow stages span six user messages, the trajectory is evaluated without requiring exact intermediate-stage alignment. Following Eq. 16, online SFT selection combines pooled tool-call recall with final-state coverage to obtain the trajectory-level

selection score $V_{\mathrm{SFT}}^{\mathrm{on}}(\tau; x)$. All six reference calls are recovered despite one additional lookup, and the recorded state-change coverage is complete.

Case 2: Online SFT Selection

```
Reference calls             : 6
Executed calls              : 7
Pooled tool-call recall     : 1.0
Final-state coverage        : 1.0
Trajectory selection score  : V_SFTˆon = 1.0
Execution errors            : none
Selection bucket            : pass
```

The additional read-only lookup does not reduce tool-call recall because this selection metric measures coverage of the reference calls rather than penalizing every extra operation. The final-state coverage component evaluates the intended persistent-state changes. This example illustrates why adaptive online interactions are evaluated at the trajectory level rather than by assigning every user message to a reference task turn.

H.3 Comparison of the Two Cases

The offline case naturalizes each predefined workflow stage into one fixed user request, enabling direct alignment with reference task turns and turn-aggregated SFT selection. The online case instead uses stage-guided user simulation to adapt requests to the solver's behavior, allowing multiple dialogue exchanges within a stage and motivating trajectory-level selection. Both derive supervision from replay-validated reference contracts without exposing reference information to the solver. Their SFT selection scores are distinct from the RL reward in Section 2.3, which uses turn-level tool-call F1 and guarded state verification.