

# Learning to Comply: Workflow-Grounded Environment Generation for Training Procedurally Compliant Agents

Seungwon Lim<sup>1,3\*†</sup>, Minu Kim<sup>2,3\*†</sup>, Seokhee Hong<sup>3</sup>  
Jinyoung Yeo<sup>1</sup>, Se-Young Yun<sup>2</sup>, Joonkee Kim<sup>3,4†‡</sup>

<sup>1</sup>Yonsei University   <sup>2</sup>KAIST   <sup>3</sup>LG AI Research   <sup>4</sup>Upstage

sngwon@yonsei.ac.kr, minu.kim@kaist.ac.kr, joonkee0701@gmail.com

## Abstract

Recent advances in AI agents have expanded their applicability beyond simple task completion to increasingly complex real-world settings. In such settings, the challenge extends beyond solving difficult tasks; it also requires following the behavioral policies or operational procedures that govern how those tasks should be carried out. However, how to systematically train agents for such procedural compliance remains underexplored, especially along two dimensions: **(1) how to construct environments for compliance training**, and **(2) how to provide learning signals** that reward compliant behavior. In this work, we introduce a scalable, automated, and verifiable pipeline for jointly generating executable tool-use environments and tasks, each paired with a ground-truth execution path. We further investigate diverse reward design strategies inspired by sequence-matching algorithms to incentivize procedurally compliant behavior, with rewards defined at both the step and trajectory levels. In experiments across diverse benchmarks requiring procedural compliance in tool-use tasks, we find that step-level rewards improve credit assignment by enabling finer-grained advantage estimation in PPO, while trajectory-level rewards, as used in GRPO, yield strong group-relative learning signals when coupled with our environment-specific reward design. Collectively, our results establish that scalable environment construction and compliance-aware reward design are both essential for training agents that are procedurally grounded.

## 1 Introduction

Most agents are designed with a primary objective: accurately executing user-requested tasks (Zhou et al., 2023; Xie et al., 2024; Patil et al., 2025). However, regarding successful execution of user-requested tasks as the sole criterion for evaluation fails to capture the realities of real-world agent tasks (Cao et al., 2026; Zhang et al., 2024; Qi et al., 2025). Imagine a customer requesting a room cancellation; instead of immediately canceling the reservation, the agent should act in accordance with the cancellation policy by carrying out the policy-defined steps, such as checking the reservation status, assessing whether the request meets the policy requirements, and determining whether the cancellation can proceed.

This example shows that, in real-world settings, the correct final task success alone is not enough; the agent must also follow the procedure prescribed by the policy, which we refer to as *procedural compliance*. Motivated by this shift, recent studies have emphasized importance of evaluating agents’ compliance in their deployment domains with policies and behavioral procedures (Cao et al., 2026; Barres et al., 2025; Li et al., 2025; Nandi et al., 2025). Yet, how to *train* agents for procedural compliance remains largely underexplored.

\*Equal contribution.

†Work done at LG AI Research.

‡Corresponding author.

Two challenges remain central: (1) scalable construction of environments with verifiable policy-prescribed procedures, and (2) learning signals that explicitly reward procedure following. Prior work on environment generation (Sullivan et al., 2025; Song et al., 2026; Gandhi et al., 2026; Froger et al., 2025) focuses mainly on general tool use rather than explicitly modeling and verifying procedural structure, while recent RL-based approaches for long-horizon agent tasks (Wang et al., 2025; Zeng et al., 2025; Qian et al., 2025) still primarily optimize for task success or tool-call correctness, rewarding rather than optimizing for procedural compliance.

To address these questions, we introduce a scalable, automated pipeline that generates executable tool-use environments and tasks from domain policies, supported by a verifiable procedural structure. Specifically, we treat a domain policy not simply as a document of rules, but as a specification of valid next actions that induces the procedural structure an agent should follow. We make this structure explicit as a *workflow graph*, which serves as the backbone for deriving the policy document, executable environment, and task instances. This workflow-grounded construction enables scalable environment generation and provides a principled basis for evaluating and training procedural compliance.

More importantly, design of workflow graph offers a unified notion of procedural correctness that extends naturally from environment construction to reward design. This unified formalization supports procedure-aware learning signals, from step-level credit assignment to graded trajectory-level matching, supporting diverse training approaches.

Using environments built from workflow graphs, we show that such rewards improve procedural compliance beyond end-task success alone. SFT benefits from path-grounded demonstrations but shows limited generalization; PPO is most effective with step-level rewards, whereas GRPO performs best with graded rewards rather than raw objective. We hope this framework provides a useful foundation for future research on training reliable and controllable agents for procedural compliance in diverse agentic tasks.

## 2 Preliminaries

### 2.1 POMDP and Tool-using Task

We formulate the tool-using agentic task as a partially observable Markov decision process (POMDP), defined by a tuple  $(\mathcal{S}, \mathcal{A}, T, \mathcal{O}, R)$ , where  $\mathcal{S}$  is the latent state space,  $\mathcal{A}$  is the action space,  $T$  is the state transition function,  $\mathcal{O}$  is the observation space, and  $R$  is the reward function. At each step, the agent takes an action  $a_t \in \mathcal{A}$  based on its interaction history and receives a new observation  $o_{t+1} \in \mathcal{O}$  from the environment.

Concretely, we consider a multi-turn dialogue setting in which a tool-using large language model (LLM) agent  $\pi$  interacts with a user and the surrounding tool environment to solve a task specified by the user. The latent state space  $\mathcal{S}$  consists of both the tool-calling environment and the user’s internal state, where the user is implemented as an LLM-based simulator that provides the task and responds to the agent. The action space of agent  $\mathcal{A}$  is given by  $\mathcal{A} = \mathcal{A}_{\text{msg}} \cup \mathcal{A}_{\text{tool}}$ , where  $\mathcal{A}_{\text{msg}}$  denotes natural-language messages to the user and  $\mathcal{A}_{\text{tool}}$  denotes tool call. The observation space is given by  $\mathcal{O} = \mathcal{O}_{\text{user}} \cup \mathcal{O}_{\text{tool}}$ , where  $\mathcal{O}_{\text{user}}$  and  $\mathcal{O}_{\text{tool}}$  denote observations returned after actions in  $\mathcal{A}_{\text{msg}}$  and  $\mathcal{A}_{\text{tool}}$ , respectively. We denote a trajectory as the resulting action–observation sequence  $\tau = (o_0, a_0, o_1, a_1, \dots, o_T) \sim \pi$ . This interaction provides partial observability for the agent, since neither the full environment state  $\mathcal{S}$  nor the information available to the user is directly accessible to it. Instead, the agent must infer such information indirectly through tool use and dialogue.

### 2.2 Domain Policy and Procedural Compliance

In real-world deployments, agents often operate under domain<sup>1</sup> policies expressed through manuals, guidelines, or operating instructions. Accordingly, agent success depends on satis-

<sup>1</sup>We refer domain as an operational setting defined by a shared set of policies, constraints, and procedures. It is broader than an individual task instance, but narrower than a broad industry category.

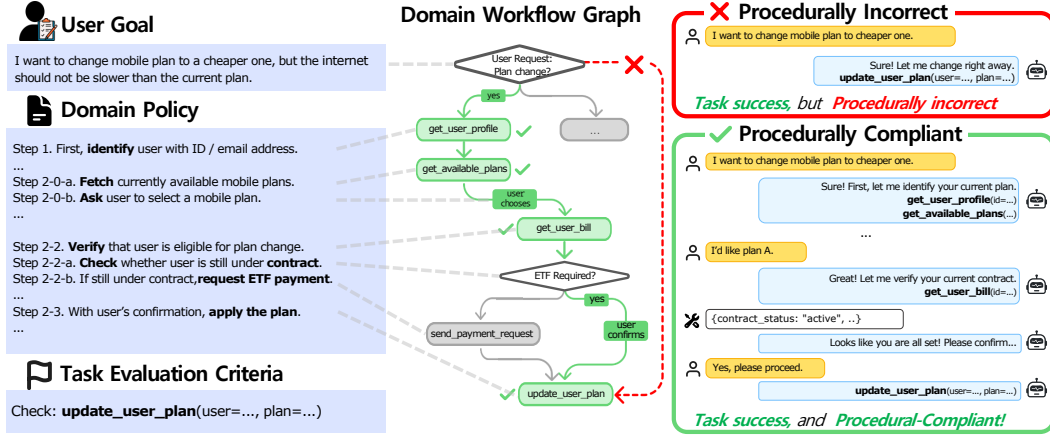


Figure 1: Example of policy-compliant tool use scenario. A domain policy induces a workflow graph and distinguishes procedural compliance from task-level success in a policy-constrained tool-use task.

following the user’s request while following the appropriate intermediate procedure, including which actions are allowed, when they should be taken, and what checks or confirmations must precede them.

These policies define more than a set of isolated rules: they induce a *procedure* that governs the interaction as it unfolds. As illustrated in Fig. 1, the policy is realized as a sequence of required actions and decision points that determine whether and when the agent may proceed. A trajectory may therefore achieve the desired task success yet still be procedurally incorrect if it bypasses required checks or confirmations. This motivates a formal representation of domain procedures as *workflows* over agent actions and observation-conditioned transitions, which we introduce in Sec. 3.

### 3 Defining Procedural Compliance over Workflow Graphs

Let  $p$  denote the domain policy that specifies valid procedures in a given domain. The agent is given a natural-language specification of  $p$  (e.g., in the system prompt). Meanwhile, the same policy determines an *oracle action* at each step:

$$a^* = f_p(s_t, o_{\leq t}, a_{< t}) \in \mathcal{A}, \quad (1)$$

which represents the single action prescribed by the domain policy given the current latent state  $s_t$  and the interaction history. The oracle function  $f_p$  thus encodes the precise procedural path that a compliant agent must follow — at each step, exactly one action is canonically correct. In practice, however, tool-use domains admit certain actions — such as read-only lookups or idempotent status checks — that are operationally neutral and not prohibited by the domain policy; they neither advance nor violate the procedure, and are therefore not represented in the workflow itself. To account for this, we additionally define:

$$F_p(s_t, o_{\leq t}, a_{< t}) \subseteq \mathcal{A}, \quad (2)$$

which augments the oracle action  $a^*$  with such procedurally inert operations that do not constitute violations.

To make the procedural structure of  $p$  explicit, we represent the structure induced by  $p$  as a directed acyclic graph (DAG)  $G = (V, E)$ , which we refer to as the *workflow graph*. Each node  $v \in V$  denotes a policy-relevant agent action in  $F_p$  and each edge  $(u, c, v) \in E$  denotes that  $f_p$  prescribes  $v$  as the oracle action after  $u$  when branch condition  $c$  is satisfied by the next observation. Concretely,  $G$  is constructed so that every root-to-leaf path corresponds to a valid policy-relevant procedural scenario permitted by  $p$ . In tool-using scenarios, these branch conditions are typically determined by tool outputs or user responses. See Fig. 1 for an illustrative example of how the graph  $G$  is instantiated from a textual policy  $p$ .

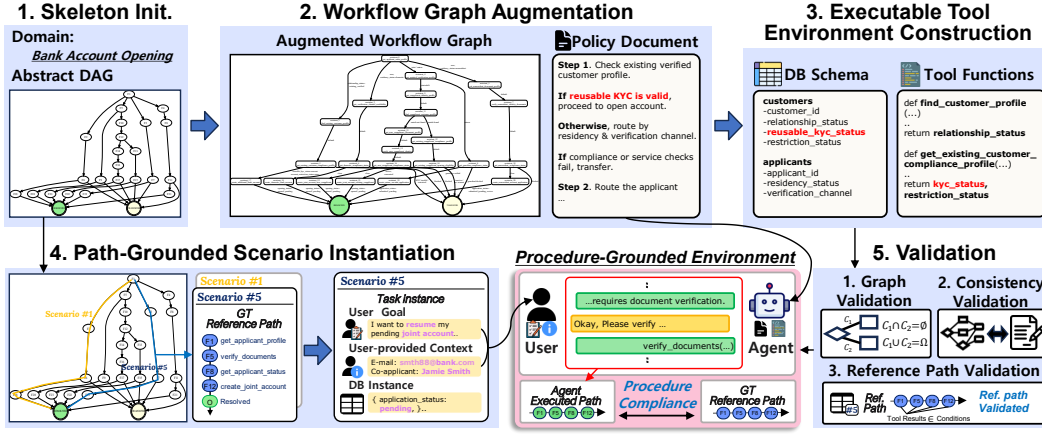


Figure 2: Overview of dataset construction pipeline.

The workflow graph captures the procedural structure shared across a domain, reflecting the domain-wide logic governing valid procedures. For a concrete task instance, however, only one sequence of branches becomes valid, depending on the latent environment state and the user-specific context. This selects a **single root-to-leaf path** in  $G$ ,

$$\rho_x^* = (v_0^*, \dots, v_T^*) \quad (3)$$

where  $x$  denotes the task instance. We call  $\rho_x^*$  the **reference path** for a task  $x$  of underlying domain. Intuitively, each root-to-leaf path corresponds to a distinct valid scenario in the domain, and a concrete task instance determines which of these scenarios is realized (e.g., product return request in a retail domain). Building upon this notation, we now formally define *procedural compliance*:

**Definition 1** (Procedural Compliance). Let  $\hat{\tau} = (\hat{\delta}_0, \hat{a}_0, \hat{\delta}_1, \hat{a}_1, \dots, \hat{\delta}_{\hat{T}})$  be an executed trajectory of an agent. We say that  $\hat{\tau}$  is *procedural-compliant* under policy  $p$  if, for every step  $t < \hat{T}$ ,

$$\hat{a}_t \in F_p(s_t, \hat{\delta}_{\leq t}, \hat{a}_{< t}), \quad (4)$$

where  $s_t$  denotes the latent environment state at step  $t$ . Further, given a task instance  $x$  with reference path  $\rho_x^*$ , an executed trajectory  $\hat{\tau}$  is *fully procedural-compliant* if it is procedural-compliant and the reference path  $\rho_x^*$  is an ordered subsequence of the node sequence induced by  $\hat{\tau}$  in  $G$ .

This definition separates local procedural correctness from *full procedural success*. An agent may follow the correct procedure for several steps and deviate only later; such a trajectory is not fully compliant, but it still reflects partial procedural progress. This graded view is particularly important in long-horizon tool-use tasks, where learning and evaluation should distinguish early correct decisions from later procedural failures. Later in Sec. 5, we build on this notion to define learning signals that reflect agreement with the reference path.

## 4 Dataset Construction

Building on the workflow graph formulation in Sec. 3, we introduce a scalable and automatic pipeline that generates three mutually consistent components from a shared workflow graph: an executable tool environment, a natural-language policy document, and path-grounded task instances. At a high level, the pipeline consists of two parts: (1) constructing a workflow graph-based executable tool environment, and (2) instantiating task instances grounded in individual root-to-leaf paths of the graph, as illustrated in Fig. 2.

### 4.1 Workflow Graph-Based Environment Construction

**Workflow Skeleton Initialization.** Starting from the workflow graph formulation in Sec. 3, we first initialize an empty abstract DAG topology with predefined nodes and edges, while

leaving their semantics (e.g., function names and branch conditions) unspecified. This graph serves as the procedural skeleton of a domain. At this stage, it specifies only the high-level control-flow structure; concrete action semantics and branch logic are augmented later. See Appendix C for details.

**Workflow Graph Augmentation.** We then prompt an LLM to complete the graph by assigning each node an atomic function definition, including its name, input arguments, and return fields, and by specifying the branch logic on each edge. Collectively, these node-level function definitions define the tool interface specifications. We further paraphrase the completed graph into a natural-language policy document that preserves the same procedural order and branching logic, which is presented to the agent at runtime. See Appendix G for detailed prompts.

|                | Min. | Max. | Avg.  |
|----------------|------|------|-------|
| Node           | 20   | 30   | 24.74 |
| Trajectory     | 8    | 30   | 17.18 |
| Length of Traj | 5.2  | 12.2 | 7.68  |

Table 1: Statistics on generated workflow graphs (100 domains).

**Executable Tool Environment Construction.** Given the natural-language policy document, the augmented workflow graph, and the tool interface specification, we construct an executable tool environment by using an LLM to generate Python tool implementations and a database schema. To preserve consistency across representations, we constrain generation so that tool names, argument signatures, response fields, and branch-triggering status values exactly match the workflow-graph specification. The resulting environment is then self-validated through syntax, structural, and runtime smoke checks performed by an LLM using terminal-based Python execution. See Appendix G for detailed prompts.

## 4.2 Path-Grounded Scenario Instantiation

We enumerate all root-to-leaf paths in the workflow graph, each corresponding to a distinct valid scenario. For every path, we instantiate a concrete task by jointly synthesizing a **User Instance**—comprising a *User Goal* and *User-Provided Context* (profile and decision criteria)—and an **Environment Instance**, specified through a database instance. These components are generated jointly using an LLM so that the enumerated path is realized as the ground-truth reference path of the task. Full prompts are provided at Appendix G.

## 4.3 Validating Generated Environments and Tasks

After constructing the dataset following the procedures in Sec. 4.1 and 4.2, the generated environments and tasks were then validated and filtered through the following steps.

**Completeness & Exclusivity of Workflow Graph.** At each decision node, branch conditions must be complete (cover all possible next observations) and exclusive (make only one branch applicable). Otherwise, a root-to-leaf path may fail to induce a unique ground-truth trajectory, either because some observations fall outside the workflow or because multiple continuations remain valid. We verified that neither case occurs, so each sampled reference path uniquely determines the ground-truth trajectory for its scenario.

| Criterion      | #    | Precision | Recall | EM   |
|----------------|------|-----------|--------|------|
| Node           | 2274 | 98.9      | 99.9   | 97.8 |
| Workflow Graph | 100  | 70        | 98     | 70   |
| Task instance  | 1326 | 1142      | 1309   | 1142 |

Table 2: Document-graph semantic consistency check. Node reports percentage; Workflow Graph and Task instance report counts of cases where all nodes meet the metric.

**Consistency Validation of Reference Path.** For each instantiated reference path, we replay the corresponding tool sequence under the generated task instances, and verify at each step that the observed tool result or user response satisfies the edge condition leading to the next node on the reference path. Any instance for which the realized observations induce a different transition is discarded. This ensures that the reference path is faithfully reconstructed as the ground-truth trajectory. Only scenarios whose instantiated task setups faithfully reconstructed their corresponding reference paths were retained after validation (1,326/1,718 task instance, 77.2%).

**Validating Document–Graph Consistency.** At inference time the agent follows the natural-language workflow document, while evaluation is performed against the workflow graph. To verify that the document preserves the graph’s transition logic, we apply a node-level QA test: for each node, an LLM judge (GPT 5.4) is given the document and asked to predict the set of valid next nodes (prompt in Table 27). Predictions are compared against the graph-defined next nodes using precision, recall, and exact match (Table 2). We retain only task instances where every node on the ground-truth reference path achieves exact match (1,142/1,326 instances, 86.1%). See Table 6 for an overview of the full filtering process.

## 5 Training LLM Agents for Procedural Compliance

The dataset construction pipeline yields more than executable tasks: it also provides the reference paths needed to provide supervision for learning. In this section, we turn those paths into training signals. The central idea is to use each validated workflow path both as a demonstration for supervised fine tuning and as a source of procedural reward for reinforcement learning.

### 5.1 Procedural Supervision from Workflow Paths

For each task  $x$ , we define the target tool-call sequence as a reference path  $\rho_x^*$ . In addition,  $\rho_x^*$  provides a natural basis for defining a *step-level procedural reward*:

$$r_t^{\text{proc}} = \begin{cases} \frac{1}{|\rho_x^*|} & \text{if } \hat{a}_t = f_p(s_t, 0 \leq t, a_{<t}) \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Note that this reward credits not the full admissible set  $F_p$ , but only the single oracle action  $a^* = f_p(\cdot)$ : admissible yet procedurally inert actions (e.g., read-only lookups) do not advance the workflow, so crediting only  $a^*$  keeps the learning signal tied to procedural progress and prevents redundant calls from inflating the return. Unlike task-level success signals, it enables credit assignment to intermediate policy-compliant progress, which is crucial in long-horizon settings where an agent may make several correct decisions before eventually deviating (Sutton, 1988). At the trajectory level, this stepwise reward naturally induces an aggregate procedural score:

$$R_{\text{proc}}(\hat{\tau} \mid \rho_x^*) := \sum_{t=0}^{\hat{\tau}-1} r_t^{\text{proc}}. \quad (6)$$

Notice that this score is bounded by 1 and reaches 1 exactly when the entire reference path is followed, *i.e.*, when it is *fully procedural-compliant*. We refer to this *trajectory-level reward* equivalently as **Subsequence Recall** (SR), since it measures the fraction of the reference path recovered in the correct order. Beyond this basic trajectory-level reward, we also consider two additional variants for procedural trajectory matching: (1) **Exact Subsequence Match** (ESM), defined as  $\mathbb{1}[R_{\text{proc}} = 1]$ ; (2) **LCS-F1**, defined from the longest common subsequence (LCS) between the generated node path  $\hat{\rho}$  induced from  $\hat{\tau}$  and the reference path  $\rho_x^*$ :

$$\text{LCS-F1} = \frac{2PR}{P+R}, \quad \text{where } \ell = |\text{LCS}(\hat{\rho}, \rho_x^*)|, \quad P = \frac{\ell}{|\hat{\rho}|}, \quad R = \frac{\ell}{|\rho_x^*|}. \quad (7)$$

Here,  $\ell$  denotes the length of the longest common subsequence between  $\hat{\rho}$  and  $\rho_x^*$ , so that LCS-F1 penalizes both missing relevant actions and generating redundant ones. Note that Exact Subsequence Match coincides with the traditional binary terminal reward when task success is defined as full procedural compliance. For SR and LCS-F1, we additionally consider an exponentially weighted variant, denoted as  $_{\text{exp}}$  (e.g.,  $\text{SR}_{\text{exp}}$ ,  $\text{LCS-F1}_{\text{exp}}$ ) that assigns smaller rewards to matches occurring at earlier turns and larger rewards to matches occurring at later turns. See Sec. 6.3 and Appendix B.1 for ablation study and detailed explanation of each reward functions.

| Training Setup           |                     | Held-out Testset    | Tau2-Retail         | Tau2-Airline        | SOPBench            | FlowBench          |
|--------------------------|---------------------|---------------------|---------------------|---------------------|---------------------|--------------------|
| <b>Qwen3 4B Instruct</b> |                     | 23.0                | 41.2                | 24.0                | 32.2                | 80.0               |
| SFT                      | <i>Distill</i>      | 47.7 (+24.7)        | 47.7 (+6.5)         | 28.2 (+4.2)         | 24.8 (-7.4)         | 66.0 (-14.0)       |
|                          | <i>Self-Distill</i> | 27.2 (+4.2)         | 43.2 (+2.0)         | 28.0 (+4.0)         | 20.7 (-11.5)        | 77.9 (-2.1)        |
| PPO                      | <i>Step-Reward</i>  | 47.4 (+24.4)        | <b>51.1 (+9.9)</b>  | 29.0 (+5.0)         | 45.7 (+13.5)        | <b>84.5 (+4.5)</b> |
|                          | <i>Traj-Reward</i>  | 24.4 (+1.4)         | 22.2 (-19.0)        | 13.0 (-11.0)        | 11.6 (-20.6)        | 77.9 (-2.1)        |
| GRPO                     | <i>Traj-Reward</i>  | <b>54.5 (+31.5)</b> | 49.8 (+8.6)         | <b>35.0 (+11.0)</b> | <b>48.5 (+16.3)</b> | 81.3 (+1.3)        |
| <b>Qwen3 8B</b>          |                     | 20.5                | 31.4                | 17.0                | 5.8                 | 61.7               |
| SFT                      | <i>Distill</i>      | <b>42.9 (+22.4)</b> | 37.1 (+5.7)         | 19.5 (+2.5)         | 13.6 (+7.8)         | 63.2 (+1.5)        |
|                          | <i>Self-Distill</i> | 15.5 (-5.0)         | 32.7 (+1.3)         | 23.0 (+6.0)         | 7.6 (+1.8)          | 66.2 (+4.5)        |
| PPO                      | <i>Step-Reward</i>  | 24.3 (+3.8)         | 33.8 (+2.4)         | 21.0 (+4.0)         | 7.5 (+1.7)          | 61.0 (-0.7)        |
|                          | <i>Traj-Reward</i>  | 15.9 (-4.6)         | 25.5 (-5.9)         | 15.5 (-1.5)         | 2.3 (-3.5)          | 51.6 (-10.1)       |
| GRPO                     | <i>Traj-Reward</i>  | 27.4 (+6.9)         | <b>45.6 (+14.2)</b> | <b>24.0 (+7.0)</b>  | <b>20.9 (+15.1)</b> | <b>67.4 (+5.7)</b> |

Table 3: Performance comparison across the held-out testset and external benchmarks.

## 5.2 Training Methods and Details

We used 100 environments constructed through the data generation pipeline described in Sec. 4. We further applied success-based task filtering, retaining only tasks that were successfully completed by DeepSeek V3.2 in Thinking mode, a strong model for complex tool-using task, across four independent runs (1,081/1,142 task instances, 94.7%). This filtering improves training data quality for SFT and removes unstable instances that could hinder RL training. Detailed training settings are provided in Appendix B.

**Supervised Fine Tuning.** We explored two variants: **(1) Distillation**, which fine-tunes on trajectories which generated during success-based filtering, where the thinking part of the teacher’s output was excluded from training; and **(2) Self-distillation**, which fine-tunes on the model’s own trajectories. For both variants, the training dataset comprised the full trajectories collected from a single rollout.

**Reinforcement Learning.** We explored two widely used variants: **(1) PPO** (Schulman et al., 2017; Ouyang et al., 2022), a representative actor–critic method, and **(2) GRPO** (Shao et al., 2024), a representative critic-free method. Our workflow-graph-based framework provides a reference action at every step, enabling rewards at two granularities. *Step-level rewards* assign partial credit  $r_t^{\text{proc}}$  for each correct action in the ground-truth sequence (i.e., procedure reward); we apply these in PPO, where per-token advantage estimation via GAE can directly exploit such signals (Schulman et al., 2015). *Trajectory-level rewards* score the entire trajectory against the reference path using sequence-matching metrics defined in Sec. 5.1; we apply these in GRPO, where group-relative comparisons operate on trajectory-level scores. Among several candidates, we select LCS-F1<sub>exp</sub> based on preliminary experiments (Sec. 6.3). We defer further details to Appendix E.

## 5.3 Benchmarks

We evaluate on four benchmarks spanning procedural compliance in tool-using tasks: (1) a **Held-out Testset** constructed from our pipeline with domains selected to minimize overlap with training data, validated via automatic verification (Sec. 4.3) and human annotation (Appendix F.1 and G). (2) **Tau2-bench** (Barres et al., 2025), which evaluates policy compliance in customer-support settings<sup>2</sup>; (3) **SOPBench** (Li et al., 2025), which tests adherence to domain-specific standard operating procedures and constraints; and (4) **FlowBench** (Xiao et al., 2024), which measures workflow adherence and correct tool invocation.

<sup>2</sup>We exclude Telecom domain, as it involves users’ tool calls (separate from the agent’s tool call scope) and is thus out of scope for our work.

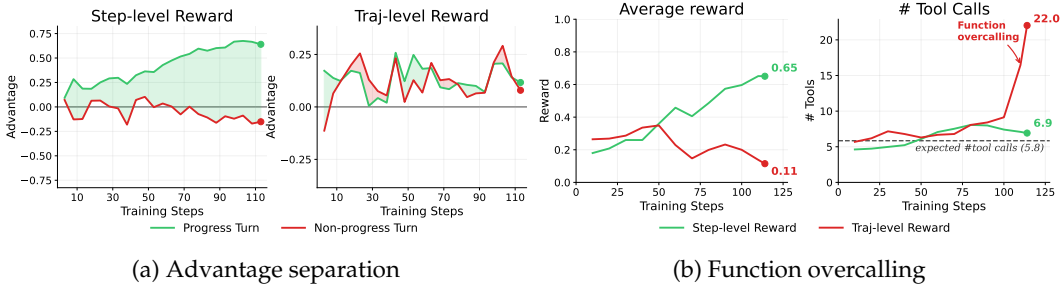


Figure 3: Comparison of PPO with step-level and trajectory-level rewards during training. (a) Advantage values of progressive and non-progressive tool calls, where the y-axis shows GAE advantage values from each rollout during training. Progress Turn denote agent function calls with  $r_t^{\text{proc}} > 0$ , while Non-progress Turn denote function calls with  $r_t^{\text{proc}} = 0$ . Step-level rewards yield a clearer separation between the two, indicating more precise credit assignment. (b) Trajectory-level rewards induce excessive tool calling despite lower reward, whereas step-level rewards avoid this degenerate tool-overcalling and keep the number of tool calls closer to the expected level.

#### 5.4 Overall Results

Table 3 summarizes the results for each training method and its corresponding variants. Across all benchmarks, we make the following observations: **(1) SFT** generally improves procedural compliance on in-distribution settings (i.e., held-out test set), with teacher distillation providing further improvements. However, these gains do not consistently transfer to external benchmarks, where performance sometimes degrades. This suggests that SFT alone has limited capacity to generalize the acquired compliance skills to unseen domains and policy structures. **(2) PPO** benefits most from step-level procedural reward, substantially outperforming its trajectory-level counterpart and the demonstration-based learning setting of SFT. **(3) GRPO** eliminates the need for a value function by generating multiple trajectories per task and computing advantages from their relative rewards within the group. This enables diverse trajectory exploration per scenario, and combined with the effectiveness of our designed reward function, GRPO mostly outperforms PPO across our experiments. We discuss these results in more detail in the next section.

## 6 Discussion on Results

### 6.1 SFT with Rejection Sampling

**Rejection Sampling Fine-tuning.** Because our pipeline provides both trajectories and procedural compliance scores, we perform rejection-sampling SFT by retaining only trajectories with a procedural score of at least 0.5. We therefore ran SFT on both the unfiltered training set and its filtered counterpart. Table 4 shows that filtering hurts performance when applied without rollout augmentation. However, when applied to augmented rollout trajectories at a larger training scale, rejection sampling improves performance.

| RS | Dataset<br>Rows (Tokens) | Inhouse<br>Testset | Tau2-<br>Retail |
|----|--------------------------|--------------------|-----------------|
| X  | 1081 (4.7M)              | 26.7               | 43.2            |
| O  | 481 (2.1M)               | 24.4               | 38.4            |
| O  | 1081 (4.9M)              | 33.0               | 47.5            |

Table 4: Performance comparison under different filtering settings. RS denotes rejection sampling.

### 6.2 Step-Level Credit Assignment Matters

A key advantage of our workflow graph-based framework is that it naturally provides step-level procedural signals  $r_t^{\text{proc}}$ , enabling credit assignment to individual procedure-advancing actions. PPO consumes these signals directly as dense per-step rewards, whereas GRPO

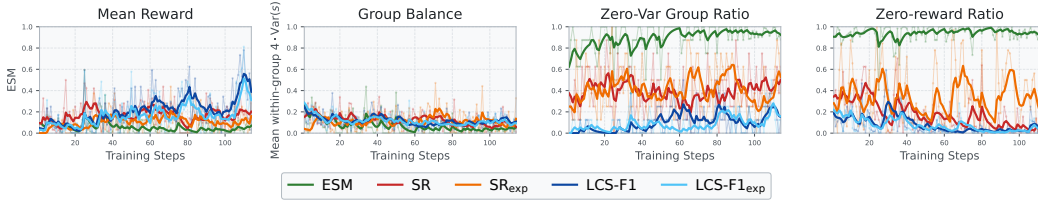


Figure 4: Training dynamics under different reward functions. (a): mean reward. (b): mean within-group reward variance. (c): proportion of groups with same rewards across all rollouts. (d): proportion of rollouts that receive zero reward.

uses trajectory-level sequence-matching scores, such as SR and LCS-F1, that aggregate the same step-level procedural alignment. Here, we begin with their direct use in PPO, where they support fine-grained advantage estimation.

**Step-Level Rewards Improve Advantage Separation.** With step-level rewards, PPO learns a sharper separation between progressive and non-progressive tool calls in advantage space. Under trajectory-level rewards (LCS-F1<sub>exp</sub>), this distinction remains blurred, indicating persistently imprecise credit assignment (Fig. 3 (a)). This result highlights that, in long-horizon procedural tasks, denser supervision from the workflow structure enables the critic to attribute value more accurately to individual decisions (Ng et al., 1999).

**Trajectory-Level Rewards Induce Degenerate Strategies.** When PPO receives only a trajectory-level reward, the weak selectivity in advantage estimates leads to a degenerate strategy: rather than identifying which tool calls are procedurally necessary, the policy increases the overall volume of calls, resulting in severe function overcalling (Fig. 3 (b)). This pathology does not arise under step-level rewards, where each correct action is individually reinforced and extraneous calls receive no credit.

### 6.3 Task-Level Success Alone Is Insufficient as a Reward Signal

Having analyzed the effect of directly exposing workflow-derived step-level rewards in PPO, we now turn to GRPO, where the same step-level procedural alignment is aggregated into a scalar trajectory reward for group-relative comparison. The relevant question is therefore not whether GRPO uses procedure-aware credit, but how this credit should be aggregated so that the resulting trajectory score preserves informative differences among rollouts. The most direct choice is to use the task objective itself as the reward—in our case, ESM. We find, however, that this seemingly natural choice leads to the weakest GRPO performance (Table 5). Because ESM is nearly binary, most groups contain only zero-reward trajectories in early training, leaving GRPO without informative within-group comparisons.

In contrast, partial sequence-matching rewards—SR and LCS-F1—provide graded scores that distinguish trajectories by how much of the reference path they recover. LCS-F1 is particularly effective as it penalizes both missing actions (via recall) and redundant ones (via precision). As shown in Fig. 4, LCS-F1-based rewards yield lower proportions of groups with no reward variation and favorable reward dynamics throughout training. The exponential weighting variant (exp) further improves performance by upweighting matches at later turns, which are reachable only after earlier prerequisites are satisfied, thereby increasing reward separation between partial and full compliance.

| Reward Function       | Inhouse Testset | Tau2-Retail |
|-----------------------|-----------------|-------------|
| ESM                   | 25.7            | 40.7        |
| SR                    | 48.8            | 45.0        |
| LCS-F1                | 52.0            | 46.5        |
| SR <sub>exp</sub>     | 45.9            | 45.6        |
| LCS-F1 <sub>exp</sub> | <b>54.5</b>     | <b>49.8</b> |

Table 5: GRPO results using different rewards.

### 6.4 Scaling Performance with the Number of Training Domains

Fig. 5 shows benchmark scores as we scale the number of training domains generated by our pipeline, using Qwen3 4B Instruct with GRPO. Performance improves monotonically

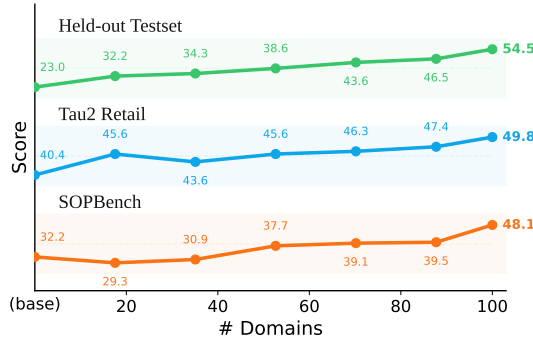


Figure 5: Effect of the number of training domains on performance.

on all three benchmarks, suggesting that generating more environments is an effective way to improve procedural generalization and that further scaling may yield additional gains.

## 7 Related Work

### 7.1 Workflow and Policy Compliance

As LLM agents are increasingly deployed in regulated settings, a growing body of work evaluates whether they follow prescribed procedures and rules. Early efforts study workflow knowledge itself, formalizing its representations or scoring generated workflows (Xiao et al., 2024; Qiao et al., 2024). More recent benchmarks evaluate adherence to customer-service policies and operating procedures in interactive environments (Barres et al., 2025; Li et al., 2025; Nandi et al., 2025), and parallel work extends this to constraint-heavy instructions, out-of-workflow requests, and long policy documents (Qi et al., 2025; Shi et al., 2025; Gupta et al., 2026; Liu et al., 2025). Our work differs from these evaluation-focused efforts in a key respect: we make the workflow explicit as both a graph and a natural-language policy document, so compliance can be measured over concrete state-action transitions rather than only end-task success, generated plans, or generic instruction following. More importantly, we use this same structure not only to *evaluate* compliance but to *train* it, deriving supervision and reward signals directly from the graph.

### 7.2 Scalable Environment Generation for Tool-Using Agents

Recent work synthesizes diverse, executable environments for scaling agent training: database-backed worlds for agentic RL (Wang et al., 2026; Song et al., 2026), procedural tool-use environments (Sullivan et al., 2025), terminal environments (Gandhi et al., 2026), trajectories mined from text (Xu et al., 2026), coding problems recast as RL tasks (Du et al., 2026), and general environment platforms (Froger et al., 2025). These efforts demonstrate the value of scalable construction, but none centers an explicit workflow graph as the source from which environment logic, user scenarios, and reward signals are jointly derived. Our pipeline instead derives all of them from a single workflow DAG, which yields three key properties: broad path coverage through systematic enumeration, deterministic instantiation of jointly synthesized environments, and—critically for process supervision—tight alignment between the policy the agent reads and the verifier at evaluation time.

## 8 Conclusion

We presented a scalable pipeline that jointly generates executable tool-use environments, natural-language policies, and path-grounded tasks from shared workflow graphs, enabling both evaluation and training of procedural compliance. Our experiments show that step-level rewards improve credit assignment in PPO, while carefully designed trajectory-level rewards yield effective group-relative signals for GRPO. We hope our framework encourages further research on training procedurally compliant agents.

## Acknowledgments

This work was supported by LG AI Research.

## References

- Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan.  $\tau^2$ -bench: Evaluating conversational agents in a dual-control environment, 2025. URL <https://arxiv.org/abs/2506.07982>.
- Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, et al. Nestful: A benchmark for evaluating llms on nested sequences of api calls. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 33526–33535, 2025.
- Hongliu Cao, Ilias Driouich, and Eoin Thomas. Beyond task completion: Revealing corrupt success in llm agents through procedure-aware evaluation. *arXiv preprint arXiv:2603.03116*, 2026.
- Weihua Du, Zhan Ling, Kang Liu, Lingfeng Shen, Xuesong Yao, Yufei Xu, Dingyuan Shi, Yiming Yang, and Jiecao Chen. Generalizable end-to-end tool-use rl with synthetic codegym. In *International Conference on Learning Representations*, volume 2026, pp. 17733–17756, 2026.
- Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, et al. Are: Scaling up agent environments and evaluations. *arXiv preprint arXiv:2509.17158*, 2025.
- Kanishk Gandhi, Shivam Garg, Noah D Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.
- Xiaojie Guo, Yang Zhang, Bing Zhang, Ryo Kawahara, Mikio Takeuchi, and Yada Zhu. Unitoolbench: A benchmark for tool-augmented llms in cross-domain, universal task automation. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 4726–4736, 2026.
- Lakshya Gupta, Litao Li, Yizhe Liu, Sriram Ganapathi Subramanian, Kaheer Suleman, Zichen Zhang, Haoye Lu, and Sumit Pasupalak. World of workflows: a benchmark for bringing world models to enterprise systems. *arXiv preprint arXiv:2601.22130*, 2026.
- Han Han, Tong Zhu, Xiang Zhang, Mengsong Wu, Xiong Hao, and Wenliang Chen. Nestools: A dataset for evaluating nested tool learning abilities of large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 9824–9844, 2025.
- Zekun Li, Shinda Huang, Jiangtian Wang, Nathan Zhang, Antonis Antoniadis, Wenyue Hua, Kaijie Zhu, Sirui Zeng, Chi Wang, William Yang Wang, et al. Sopbench: Evaluating language agents at following standard operating procedures and constraints. *arXiv preprint arXiv:2503.08669*, 2025.
- Jiateng Liu, Zhenhailong Wang, Xiaojian Huang, Yingjie Li, Xing Fan, Xiang Li, Chenlei Guo, Ruhi Sarikaya, and Heng Ji. Analyzing and internalizing complex policy documents for llm agents. *arXiv preprint arXiv:2510.11588*, 2025.
- Hyunji Min, Sangwon Jung, Junyoung Sung, Dosung Lee, Leekyeung Han, and Paul Hong-suck Seo. Goat: A training framework for goal-oriented agent with tools. *arXiv preprint arXiv:2510.12218*, 2025.
- Subhrangshu Nandi, Arghya Datta, Nikhil Vichare, Indranil Bhattacharya, Huzefa Raja, Jing Xu, Shayan Ray, Giuseppe Carenini, Abhi Srivastava, Aaron Chan, et al. Sop-bench: Complex industrial sops for evaluating llm agents. *arXiv preprint arXiv:2506.08119*, 2025.

- Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- 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.
- Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. Agentif: Benchmarking instruction following of large language models in agentic scenarios. *arXiv preprint arXiv:2505.16944*, 2025.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.
- Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. *arXiv preprint arXiv:2410.07869*, 2024.
- John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- 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.
- Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. *Advances in Neural Information Processing Systems*, 37:4540–4574, 2024.
- Yuchen Shi, Siqi Cai, Zihan Xu, Yuei Qin, Gang Li, Hang Shao, Jiawei Chen, Deqing Yang, Ke Li, and Xing Sun. Flowagent: Achieving compliance and flexibility for workflow agents. *arXiv preprint arXiv:2502.14345*, 2025.
- Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Zhicheng Dou, and Ji-Rong Wen. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. *arXiv preprint arXiv:2601.05808*, 2026.
- Michael Sullivan, Mareike Hartmann, and Alexander Koller. Procedural environment generation for tool-use agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18555–18573, 2025.
- Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.
- 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*, 2026.
- 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*, 2025.

- Ruixuan Xiao, Wentao Ma, Ke Wang, Yuchuan Wu, Junbo Zhao, Haobo Wang, Fei Huang, and Yongbin Li. Flowbench: Revisiting and benchmarking workflow-guided planning for llm-based agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10883–10900, 2024.
- 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. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.
- Zhihao Xu, Rumei Li, Jiahuan Li, Rongxiang Weng, Jingang Wang, Xunliang Cai, and Xiting Wang. Unlocking implicit experience: Synthesizing tool-use trajectories from text. *arXiv preprint arXiv:2601.10355*, 2026.
- Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, Yang Katie Zhao, and Mingyi Hong. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. In *ICML 2025 Workshop on Computer Use Agents*, 2025.
- Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024.
- 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. *arXiv preprint arXiv:2307.13854*, 2023.

## A Limitations

Although our workflow-grounded pipeline enables scalable and verifiable construction of training environments across diverse domains, our setup remains fundamentally synthetic: The executable environments, policy documents, tool interfaces, and task instances are all generated from a shared workflow graph, which may not fully capture the messiness of real deployments, such as undocumented exceptions, evolving operational practices, noisy records, and organization-specific conventions. Closing this gap, e.g., by seeding generation from real operating procedures and logs, is a natural direction for future work. In addition, both training and evaluation rely on an LLM-based user simulator rather than real human users; while we find the simulator to be largely faithful overall, residual errors remain and become more concentrated in later turns of long interactions. Because procedural compliance in our setting depends on both tool execution and multi-turn user interaction, these two factors together limit how directly our results can be interpreted as evidence of real-world performance, and motivate future evaluation in more naturalistic environments with human users.

## B Experimental Details

For the two models considered in this work, Qwen3 4B Instruct and Qwen3 8B, all training and evaluation was conducted exclusively in the no-think mode, as the complexity of managing reasoning across multi-turn interactions introduces non-trivial implementation challenges that are orthogonal to the focus of this work. Also, we use DeepSeek V3.2 (no-think mode) as the user simulator; see Appendix D for the selection rationale and a detailed analysis of user simulation faithfulness.

We note here that the validation procedure in Sec. 4.3 ensures environment correctness, but some scenarios may still be unsolvable for the model under training, yielding no positive reward signal and potentially destabilizing RL training. To mitigate this, we apply an additional filtering step using a strong teacher model (DeepSeek V3.2 Thinking mode) as a difficulty proxy, excluding scenarios that the teacher fails to solve in all four independent rollouts. As a result, out of the 1,142 task instances that remain after the validation process in Sec. 4.3, 1,081 are retained, and we use all 1,081 for SFT. For RL, we split them into 915 training tasks and 166 validation instances.

| Process                                         | # Instances |
|-------------------------------------------------|-------------|
| Initial task instances                          | 1,718       |
| → Sec. 4.3 Reference-path validation            | 1,326       |
| → Sec. 4.3 Document-graph consistency filtering | 1,142       |
| → Sec. 5.2 pass@4 by DeepSeek V3.2              | 1,081       |
| → Train/Valid split for RL                      | 915/166     |

Table 6: Task-instance validation and filtering pipeline.

### B.1 Reward Design Details

**Definition of GRPO reward functions.** Table 7 shows the definitions and representative behaviors of the three trajectory-level reward metrics used in our analysis. ESM is the strictest metric, assigning credit only when the full reference trajectory is recovered in the correct order, whereas SR provides recall-style partial credit for ordered recovery even when the rollout is incomplete. LCS-F1 further accounts for unnecessary calls by combining precision and recall, making it more sensitive to both missing and redundant actions. Together, these metrics capture complementary aspects of trajectory quality.

**Exponential weighting.** To place more emphasis on late-stage progress, we also consider exponentially weighted variants of the ordered-recovery metrics. Let the reference trajectory

| Metric | Definition                                                                                                    | Formula                                                                      | Examples: $\hat{\rho}$ vs. $r = \text{ABCDEF}$                                                             |
|--------|---------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| ESM    | Returns 1 iff the full reference trajectory appears in the generated trajectory as an ordered subsequence.    | $\text{ESM}(\hat{\rho}, r) = \mathbb{1}[L =  r ]$                            | Exact ABCDEF: 1<br>Extra calls ABXXCDEFXY: 1<br>Incomplete ABCD: 0<br>Misordered ADBECF: 0                 |
| SR     | Recall-style partial credit based on the fraction of the reference trajectory recovered in the correct order. | $\text{SR}(\hat{\rho}, r) = \frac{L}{ r }$                                   | Exact ABCDEF: 1.000<br>Extra calls ABXXCDEFXY: 1.000<br>Incomplete ABCD: 0.667<br>Misordered ADBECF: 0.667 |
| LCS-F1 | LCS-based F1 that rewards ordered recovery while penalizing both missing steps and redundant calls.           | $P = \frac{L}{ \hat{\rho} }, R = \frac{L}{ r }, \text{F1} = \frac{2PR}{P+R}$ | Exact ABCDEF: 1.000<br>Extra calls ABXXCDEFXY: 0.750<br>Incomplete ABCD: 0.800<br>Misordered ADBECF: 0.667 |

Table 7: Trajectory-level reward metrics and illustrative scores. Here  $\hat{\rho}$  denotes the generated trajectory,  $r$  the reference (GT) trajectory, and  $L = |\text{LCS}(\hat{\rho}, r)|$ . In the Examples column, each typewriter string is the generated trajectory  $\hat{\rho}$  being scored against  $r = \text{ABCDEF}$ .

be  $r = (r_1, \dots, r_{|r|})$ , and assign step-wise weights

$$w_i = \frac{\exp(\beta i / |r|) - \exp(\beta(i-1) / |r|)}{\exp(\beta) - 1}, \quad i = 1, \dots, |r|, \quad \beta > 0.$$

These weights sum to 1 and increase with  $i$ , so later reference steps receive larger credit.

For any ordered alignment between  $\hat{\rho}$  and  $r$  that matches reference positions  $i_1 < \dots < i_m$ , its weighted score is

$$\sum_{j=1}^m w_{i_j}.$$

Let  $\tilde{R}(\hat{\rho}, r)$  denote the maximum of this quantity over all valid ordered alignments. We then define the exponentially weighted version of SR as

$$\text{SR}_{\text{exp}}(\hat{\rho}, r) = \tilde{R}(\hat{\rho}, r).$$

Similarly, keeping the same precision term as in LCS-F1,

$$P = \frac{L}{|\hat{\rho}|}, \quad \text{LCS-F1}_{\text{exp}}(\hat{\rho}, r) = \frac{2P \tilde{R}}{P + \tilde{R}}.$$

When  $\beta \rightarrow 0$ , the weights become uniform and  $\text{SR}_{\text{exp}}$  reduces to SR. Larger  $\beta$  puts more emphasis on recovering later steps in the reference trajectory. We keep ESM unchanged.

## B.2 Training Configurations

**SFT.** For both distillation and self-distillation, we fine-tune on full multi-turn trajectories using a standard causal language modeling objective. Table 8 lists the hyperparameters used.

**PPO.** We train with PPO using the standard actor-critic setup with GAE for advantage estimation. Table 9 lists the hyperparameters used.

**GRPO.** We train with GRPO using trajectory-level rewards computed over groups of rollouts for the same task instance. Based on the reward function analysis in Sec. 6.3 and Appendix B.1, we use  $\text{LCS-F1}_{\text{exp}}$  as the default reward function. Table 10 lists the hyperparameters used.

| Parameter           | Value              | Parameter         | Value  | Parameter        | Value        |
|---------------------|--------------------|-------------------|--------|------------------|--------------|
| Learning rate       | $1 \times 10^{-6}$ | Optimizer         | AdamW  | Adam betas       | (0.9, 0.999) |
| Weight decay        | 0.01               | LR scheduler      | cosine | Warmup ratio     | 0.03         |
| Gradient clipping   | 1.0                | Global batch size | 8      | Micro-batch size | 1            |
| Max sequence length | 16384              | Training epochs   | 3      |                  |              |

Table 8: SFT hyperparameters.

| Parameter                           | Value              | Parameter                | Value        | Parameter                | Value              |
|-------------------------------------|--------------------|--------------------------|--------------|--------------------------|--------------------|
| Train batch size                    | 8                  | Actor micro-batch size   | 1            | Critic micro-batch size  | 1                  |
| Rollouts per prompt                 | 4                  | PPO epochs               | 1            | Training epochs          | 2                  |
| Advantage estimator                 | gae                | Optimizer                | AdamW        | Actor learning rate      | $1 \times 10^{-6}$ |
| Critic learning rate                | $1 \times 10^{-5}$ | Adam betas               | (0.9, 0.999) | Weight decay             | 0.01               |
| LR scheduler                        | constant           | Warmup ratio             | 0.0          |                          |                    |
| Max prompt length                   | 16384              | Max response length      | 8192         | Max tool response length | 500                |
| Max assistant turns                 | 30                 | Max user turns           | 12           | KL loss                  | disabled           |
| KL coefficient                      | 0.0                | Discount factor $\gamma$ | 1.0          | GAE $\lambda$            | 1.0                |
| Policy clip ratio                   | 0.2                | Value clip range         | 0.5          | Entropy coefficient      | 0                  |
| Exponential weighting param $\beta$ | 3.0                |                          |              |                          |                    |

Table 9: PPO hyperparameters.

| Parameter                | Value | Parameter                           | Value              | Parameter           | Value        |
|--------------------------|-------|-------------------------------------|--------------------|---------------------|--------------|
| Optimizer                | AdamW | Actor learning rate                 | $1 \times 10^{-6}$ | Adam betas          | (0.9, 0.999) |
| Weight decay             | 0.01  | LR scheduler                        | constant           | Warmup ratio        | 0.0          |
| Train batch size         | 8     | Rollouts per prompt                 | 8                  | PPO mini-batch size | 1            |
| Micro-batch size         | 1     | Max prompt length                   | 16384              | Max response length | 8192         |
| Max tool response length | 500   | Max assistant turns                 | 30                 | Max user turns      | 12           |
| Gradient checkpointing   | True  | KL loss                             | disabled           | KL coefficient      | 0.0          |
| Discount factor $\gamma$ | 1.0   | GAE / return $\lambda$              | 1.0                | Policy clip ratio   | 0.2          |
| Entropy coefficient      | 0     | Exponential weighting param $\beta$ | 3.0                | Training epochs     | 1            |

Table 10: GRPO hyperparameters.

### B.3 Hyperparameter Sensitivity

Three values are fixed in Appendix B.2: the rejection-sampling threshold, the exponential weighting strength  $\beta$ , and the GRPO group size. We vary each one and report the effect on the held-out test set and on Tau2-Retail (Table 11).

**Rejection-sampling threshold.** We perform rejection-sampling SFT by keeping only trajectories whose procedural score is at least the threshold. Table 4 reports SFT at thresholds 0 and 0.5 (threshold 0 is  $RS = X$ , i.e., no filtering); we additionally run SFT at 0.7, oversampling so that all three settings train on 1,081 trajectories. With the same dataset size, a stricter threshold leads to better training.

**Exponential weighting.** We train with  $\beta = 5$  in addition to  $\beta = 3$  ( $\beta = 0$  is substituted by the unweighted SR and LCS-F1 runs). Raising  $\beta$  from 3 to 5 lowers both reward functions, and the drop is much larger for  $LCS-F1_{exp}$  than for  $SR_{exp}$ . GRPO learns from how rollouts rank within a group, not from their absolute reward. Increasing  $\beta$  packs the rewards for early and middle steps into a narrow range while sharply enlarging those for the last few steps (Fig. 6); at  $\beta = 5$  only the final one to three steps of a trajectory receive more reward than at  $\beta = 3$ , while all earlier steps receive several times less. Because  $SR_{exp}$  counts matched reference actions in order, rollouts that recovered more of the reference path keep their relative order within the group even under this compression. However,  $LCS-F1_{exp}$  penalizes non-progressive tool calls. At  $\beta = 5$  the score gap between rollouts that differ in their early or middle steps narrows, so a few such calls can flip the ranking within a group.

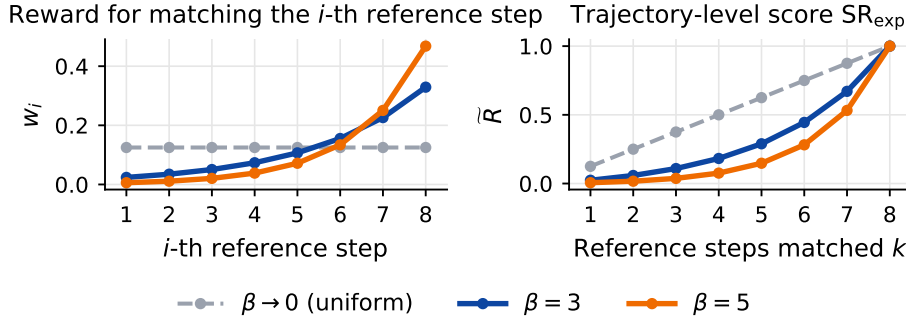


Figure 6: Exponential weighting for an example reference path of length 8. Left: reward  $w_i$  added by matching the  $i$ -th reference step, which is also the score gap GRPO ranks rollouts by. Right: the resulting trajectory-level score ( $SR_{exp}$ ).

**Group size.** GRPO uses a group size of 8; training with 4 is worse on both benchmarks. We could not train with 16 under our compute budget, but during group-size-8 training we sampled 16 rollouts without backpropagation and measured the zero-variance group ratio: it falls from 0.181 to 0.172, a negligible difference, so we keep 8.

## B.4 Experimental Setup

All training experiments were conducted on  $8 \times A6000$  GPUs. SFT training for 3 epochs typically takes approximately 6 hours, while RL training for 1 epoch typically takes approximately 12 hours. All evaluation experiments were conducted on  $4 \times A100$  GPUs.

## C Detailed Explanation for Workflow Graph and Environment

### C.1 Diversity of Generated Workflow Graphs

The topological skeleton of each workflow graph is generated before any domain semantics are assigned. Given a target size  $n$  (between 20 and 30), the algorithm proceeds in three steps:

- 1. Random labeled rooted tree.** A uniformly random labeled rooted tree on  $n - 2$  nodes is sampled using the NetworkX implementation of Prüfer-sequence-based generation (`random_labeled_rooted_tree`), which produces a distinct tree topology for each random seed.

| Setting                                                                    | Held-out Testset | Tau2-Retail  |
|----------------------------------------------------------------------------|------------------|--------------|
| <b>Rejection-sampling threshold</b>                                        |                  |              |
| <b>0.5</b>                                                                 | 33.0             | 47.5         |
| 0                                                                          | 26.7 (−6.3)      | 43.2 (−4.3)  |
| 0.7                                                                        | 40.8 (+7.8)      | 49.1 (+1.6)  |
| <b>Exponential weighting <math>\beta</math>, <math>SR_{exp}</math></b>     |                  |              |
| <b>3</b>                                                                   | 45.9             | 45.6         |
| 0                                                                          | 48.8 (+2.9)      | 45.0 (−0.6)  |
| 5                                                                          | 43.6 (−2.3)      | 43.9 (−1.7)  |
| <b>Exponential weighting <math>\beta</math>, <math>LCS-F1_{exp}</math></b> |                  |              |
| <b>3</b>                                                                   | 54.5             | 49.8         |
| 0                                                                          | 52.0 (−2.5)      | 46.5 (−3.3)  |
| 5                                                                          | 48.5 (−6.0)      | 36.8 (−13.0) |
| <b>GRPO group size</b>                                                     |                  |              |
| <b>8</b>                                                                   | 54.5             | 49.8         |
| 4                                                                          | 50.9 (−3.6)      | 47.4 (−2.4)  |

Table 11: Hyperparameter sensitivity. Bold values are the settings chosen for the main experiments.

**2. Directed orientation.** The undirected tree is converted into a directed acyclic graph (DAG) by orienting every edge away from the root via a breadth-first traversal, yielding a tree-shaped DAG whose internal branching pattern is determined by the random topology.

**3. Terminal node attachment.** Two terminal nodes, RESOLVED and TRANSFER, are appended to the graph: every leaf of the oriented tree receives outgoing edges to both terminals. This ensures that every root-to-leaf path ends at one of the two canonical outcomes.

Because the initial tree is sampled uniformly at random from the space of all labeled rooted trees on  $n - 2$  nodes, the resulting DAGs exhibit substantial topological diversity: they vary in depth, branching factor, the number and length of root-to-leaf paths, and the positions at which branching occurs. Fig. 7 shows three representative empty DAG skeletons generated with different random seeds, illustrating this diversity.

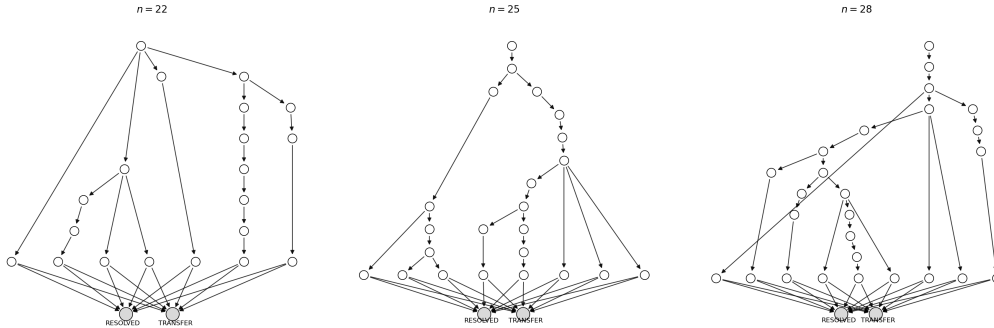


Figure 7: Three empty DAG skeletons generated by our workflow graph initialization algorithm with different random seeds ( $n = 22, 25, 28$ ). The graphs differ in depth, branching factor, and the number of root-to-leaf paths, demonstrating the topological diversity of the generated structures. Terminal nodes (RESOLVED, TRANSFER) are shown in gray.

Table 1 in the main text summarizes the resulting statistics across our 100 generated domains: the number of nodes ranges from 20 to 30 (avg. 24.74), the number of distinct root-to-leaf trajectories ranges from 8 to 30 (avg. 17.18), and the average trajectory length ranges from 5.2 to 12.2 steps (avg. 7.68). This variation ensures that the procedural complexity of the

generated environments is not uniform, exposing trained agents to a broad spectrum of workflow structures during learning.

We treat this pre-sized skeleton as a controllable scaffold rather than a model of any specific real-world workflow: fixing the topology in advance makes procedural difficulty—graph size, branching factor, and path length—configurable and decoupled from LLM stochasticity, while the LLM supplies only domain semantics on top.

## C.2 Reproducibility of the Generation Pipeline

To assess the reproducibility of the generated environments, we regenerate all 100 training domains with the same prompts under a different random seed. On 20 randomly sampled domains, annotators compare each regenerated environment to its original and mark the fraction of tools with matching functionality and of DB attributes with matching characteristics, at 60.5% and 79.8% respectively (Table 12). Task difficulty is likewise stable: pass@1 of four solver models tracks closely between the original and regenerated environments (Table 13), as task difficulty is driven not only by the domain semantics but in large part by the deterministic graph-generation stage.

| Environment Component | Overlap (%) |
|-----------------------|-------------|
| Tool                  | 60.5        |
| DB attribute          | 79.8        |

Table 12: Human-verified component overlap between original and regenerated environments, on 20 sampled domains.

| Model             | Pass@1 (Original) | Pass@1 (Regenerated) |
|-------------------|-------------------|----------------------|
| Qwen3 4B Instruct | 21.8              | 19.9                 |
| Qwen3 8B          | 18.9              | 20.3                 |
| GPT 5.4           | 43.8              | 46.5                 |
| DeepSeek V3.2     | 45.2              | 44.0                 |

Table 13: Pass@1 of four solver models on tasks from the original and regenerated environments.

## C.3 Difference between Workflow Graphs and Tool Dependency Graphs

Several prior works construct graph structures over tools to generate multi-step tool-use scenarios. TaskBench (Shen et al., 2024) introduces a *Tool Graph* that captures resource dependencies (the output type of one tool matches the input type of another) and temporal dependencies (a prescribed ordering between tools) to sample subgraphs from which user instructions are back-generated. GOAT (Min et al., 2025) builds an *API Dependency Graph* in which a directed edge from tool  $t_i$  to  $t_j$  indicates that the output of  $t_i$  can populate a specific input parameter of  $t_j$ ; connected subgraphs are then sampled and executed to produce goal-oriented training data. NesTools (Han et al., 2025) and NESTful (Basu et al., 2025) similarly model nested tool-call sequences as DAGs where edges denote that one call’s return value is passed as an argument to a subsequent call. UniToolBench (Guo et al., 2026) formulates tool graph construction as a link-prediction problem over input–output type compatibility to scale across diverse API collections.

While our workflow graph shares the high-level idea of representing tool interactions as a directed graph, it differs from these tool dependency graphs in two fundamental respects.

**Policy-awareness versus data-flow wiring.** Tool dependency graphs are *data-flow* graphs: an edge exists whenever the output of one tool can serve as input to another, regardless of whether that transition is prescribed or even desirable. The graph captures what is *technically possible* given input–output type compatibility. In contrast, our workflow graph

encodes what is *procedurally required* by the domain policy. Edges represent not mere data compatibility but policy-mandated transitions: a branch from node  $u$  to node  $v$  means that the domain procedure prescribes action  $v$  after action  $u$  when a specific condition is met. As a result, the workflow graph captures the normative structure of a domain—which actions *should* be taken in which order—rather than the space of technically feasible tool compositions. This distinction is crucial for our setting, where the goal is not to chain tools into a working pipeline, but to ensure that the agent follows the correct operational procedure.

**Semantic branching versus structural connectivity.** In tool dependency graphs, edges are determined by structural properties of the tool interface: parameter names, types, and return fields. Two tools are connected if their signatures are compatible, and the graph is typically constructed through type matching (Shen et al., 2024), embedding similarity (Min et al., 2025), or link prediction (Guo et al., 2026). The branching structure of these graphs reflects the combinatorial space of possible tool compositions, not domain-specific decision logic. By contrast, edges in our workflow graph carry explicit *semantic branch conditions* tied to the domain context—for example, “if the user’s account is eligible for cancellation” or “if the requested plan change exceeds the spending limit.” These conditions are determined by observation-dependent factors such as tool return values and user responses, and they partition the set of possible continuations at each node into mutually exclusive and exhaustive cases. This semantic branching means that each root-to-leaf path corresponds to a distinct, meaningful domain scenario (e.g., “eligible cancellation with confirmation” versus “ineligible cancellation requiring transfer”), rather than merely a feasible sequence of API calls.

#### C.4 Example of a Generated Environment and Task

Fig. 10 illustrates the full output of our pipeline for the vehicle diagnostics health reporting domain, showing each artifact produced at successive stages of the construction process.

**(a) Empty Workflow Graph.** The pipeline begins by initializing an abstract DAG skeleton that defines the procedural control flow—node count, branching structure, and terminal nodes—without any domain semantics. At this stage, nodes and edges carry only structural identifiers.

**(b) Augmented Workflow Graph.** An LLM then populates the skeleton with domain-specific semantics: each node is assigned a concrete function name, input arguments, and return fields, while each edge is annotated with the branch condition (e.g., a specific tool-output value) that triggers the corresponding transition. The resulting graph fully specifies all valid procedural scenarios in the domain.

**(c) Policy Document.** The augmented workflow graph is paraphrased into a natural-language policy document that preserves the same procedural ordering and branching logic. This document serves as the agent’s instruction at inference time, describing the required steps, decision criteria, and permissible actions in prose form.

**(d) Tool Implementation.** A coding agent generates executable Python implementations for each tool defined in the workflow graph. Function signatures, argument names, and return fields are constrained to exactly match the graph specification, ensuring consistency between the procedural representation and the runtime environment.

**(e) DB Schema Implementation.** Alongside the tool implementations, the pipeline produces a database schema that defines the tables and fields required by the tools. This schema provides the backing data store against which tool calls are executed during both validation and agent interaction.

**(f) Task Setup.** Finally, for a specific root-to-leaf path in the workflow graph, the pipeline instantiates a concrete task by jointly generating a *user instance*—comprising a user goal and user-provided context—and an *environment instance*—comprising a database populated with specific records. These are synthesized so that the designated path is realized as the ground-truth reference path under the given user context and database state.

## C.5 Domain Statistics

| Category                              | Domains                                                                                                                                                                                                |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| E-commerce / Commerce                 | 1. online marketplace order processing, 2. product catalog management<br>3. shopping cart checkout, 4. ecommerce returns refunds<br>5. product review rating system                                    |
| Healthcare / Medical Services         | 6. patient appointment scheduling, 7. electronic medical records management<br>8. prescription processing pharmacy, 9. lab test ordering results delivery<br>10. telemedicine video consultation       |
| Finance / Banking / Fintech           | 11. bank account opening kyc verification, 12. loan application approval process<br>13. money transfer p2p payments, 14. credit card application management<br>15. investment portfolio management     |
| Travel / Reservations                 | 16. hotel room booking reservation, 17. flight booking eticketing<br>18. travel itinerary planning, 19. rental car booking pickup<br>20. tour package customization                                    |
| Telecom Services                      | 21. mobile plan subscription upgrade, 22. network issue troubleshooting<br>23. telecom bill payment invoicing, 24. sim card activation number porting<br>25. data usage monitoring alerts              |
| Foodtech / Restaurant Operations      | 26. food delivery order processing, 27. restaurant table reservation<br>28. menu inventory management, 29. kitchen order display system<br>30. restaurant loyalty rewards program                      |
| Smart Home / IoT                      | 31. home automation control system, 32. smart security system monitoring<br>33. home energy management optimization, 34. iot device pairing setup<br>35. smart home routine schedule management        |
| Customer Support / CX                 | 36. support ticket creation routing, 37. customer complaint resolution<br>38. faq knowledge base search, 39. live chat support system<br>40. customer feedback collection survey                       |
| HR / People Operations                | 41. employee onboarding process, 42. leave attendance management<br>43. performance review evaluation, 44. payroll processing tax calculation<br>45. recruitment job application processing            |
| Logistics / Supply Chain              | 46. shipment tracking status update, 47. warehouse inventory management<br>48. delivery route optimization, 49. last mile delivery coordination<br>50. supplier order procurement                      |
| Education / EdTech                    | 51. course enrollment registration, 52. assignment submission grading<br>53. online exam proctoring system, 54. student progress tracking analytics<br>55. certificate issuance verification           |
| Real Estate / Property Management     | 56. property listing search platform, 57. virtual property tour booking<br>58. rental application processing, 59. property maintenance request system<br>60. lease agreement management                |
| Insurance                             | 61. insurance policy quote generation, 62. insurance claim filing processing<br>63. policy renewal modification, 64. beneficiary information management<br>65. insurance risk assessment               |
| Media / Entertainment Platforms       | 66. video streaming playback service, 67. music playlist management<br>68. event ticket booking seat selection, 69. content recommendation engine<br>70. subscription plan management                  |
| Manufacturing / Production Operations | 71. production line monitoring control, 72. quality control inspection<br>73. equipment maintenance scheduling, 74. bill of materials management<br>75. work order processing tracking                 |
| Utilities / Energy                    | 76. utility meter reading billing, 77. power outage reporting tracking<br>78. energy consumption analytics dashboard, 79. utility service connection request<br>80. renewable energy monitoring system |
| LegalTech / Legal Operations          | 81. legal case management system, 82. document review ediscovery<br>83. contract lifecycle management, 84. legal billing time tracking<br>85. court filing scheduling                                  |
| Marketing / Growth                    | 86. marketing campaign management, 87. lead generation nurturing workflow<br>88. email marketing automation, 89. social media content scheduling<br>90. marketing analytics roi reporting              |
| Retail Operations / POS               | 91. point of sale transaction processing, 92. retail inventory replenishment<br>93. customer loyalty program management, 94. price promotion management<br>95. store operations staff scheduling       |
| Public Administration / GovTech       | 96. permit license application processing, 97. tax filing payment system<br>98. public records request system, 99. voter registration election management<br>100. social welfare benefits enrollment   |

Table 14: Full list of the 100 domains used in our training dataset, grouped into 20 higher-level categories.

Our pipeline generated 100 environments for training and 20 for the held-out testset. The domains span a wide range of industries and operational settings, including customer service, healthcare, finance, logistics, and technical support, among others. The full list of domains is provided in Table 14 and Table 15.

| Category                                    | Domains                                                                                                                                                                                                                     |
|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Smart Agriculture &amp; AgriTech</b>     | 1. crop planning seeding management, 2. smart irrigation fertilization control<br>3. livestock health feeding tracking, 4. farm equipment maintenance usage<br>5. harvest yield forecasting traceability                    |
| <b>Construction &amp; Site Operations</b>   | 6. construction project schedule management, 7. site safety incident reporting<br>8. building material delivery tracking, 9. subcontractor workforce coordination<br>10. site inspection punchlist management               |
| <b>Mobility &amp; Fleet Management</b>      | 11. vehicle service maintenance booking, 12. fleet telematics monitoring dispatch<br>13. ev charging session management, 14. roadside assistance request dispatch<br>15. vehicle diagnostics health reporting               |
| <b>Laboratory &amp; Research Operations</b> | 16. experiment protocol workflow management, 17. sample tracking chain of custody<br>18. laboratory equipment booking calibration, 19. research data collection analysis pipeline<br>20. lab compliance audit documentation |

Table 15: Full list of the 20 domains used in our held-out testset, grouped into 4 higher-level categories.

| Model              | Task Entailment (%) | Info Entailment (%) |
|--------------------|---------------------|---------------------|
| DeepSeek V3.2 (NT) | 96.8                | 95.4                |
| GPT 4.1            | 98.2                | 97.6                |
| Qwen3 30B A3B      | 86.3                | 88.7                |

Table 16: Faithfulness of user simulators by each LLM, measured by Task Entailment and Info Entailment. Scores are computed over all user utterances in trajectories generated from task instances across 100 training environments, with the assistant fixed to DeepSeek V3.2 (Think). NT denotes non-thinking mode.

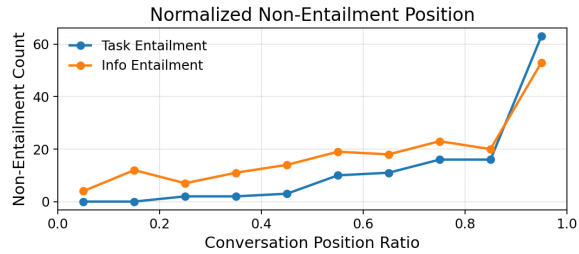


Figure 8: Distribution of non-entailment errors over the course of a conversation. Errors are rare in early turns and concentrate toward the end of the dialogue, indicating that most failures arise from late-stage drift.

## D Evaluating User Simulator Faithfulness

To ensure that the user simulator behaves faithfully throughout multi-turn interactions, we evaluate each simulated user utterance with an LLM judge. The judge is given the scenario specification—including the user goal, user-side context, predefined response options, and dialogue history—and is instructed to assess whether the simulator remains consistent with its assigned role.

The evaluation proceeds in two stages. First, the judge identifies whether an utterance contains a task request, information provision, or both. Task requests correspond to instructions to perform, continue, or modify a task, while information provision refers to user-side facts, preferences, constraints, or answers to clarification questions. Second, the judge performs entailment checks conditioned on the utterance type. For utterances containing task requests, it evaluates **Task Entailment**, i.e., whether the request is supported by the original user goal. For utterances containing information provision, it evaluates **Info Entailment**, i.e., whether the provided content is supported by the predefined user profile and response constraints. Utterances that contain neither are excluded from the entailment step. See Table 28 and 29 for prompts used in these two stages.

We aggregate these utterance-level judgments across trajectories to compare candidate LLMs as user simulators. As shown in Table 16, GPT 4.1 achieves the highest scores on both Task Entailment and Info Entailment, while DeepSeek V3.2 (NT) also maintains strong faithfulness. We nevertheless use DeepSeek V3.2 (NT) as the user simulator because it provides a substantially better cost-quality trade-off for large-scale data collection and evaluation. In contrast, Qwen3 30B A3B performs notably worse on both metrics. Fig. 8 further shows that non-entailment errors are rare in early turns and become concentrated toward the end of the conversation. Info-related deviations appear slightly earlier and are

somewhat more broadly distributed, but both error types peak sharply in the final portion of the dialogue, indicating that the remaining failures are primarily due to late-stage drift rather than persistent inconsistency.

## E More Discussion

### E.1 Performance Gap Between Qwen3 4B Instruct and Qwen3 8B

As shown in Table 3 (gray rows), Qwen3 8B scores below Qwen3 4B Instruct both before and after training. The two models differ in their post-training lineage: the 4B model is the Qwen3-Instruct-2507 variant<sup>3</sup>, which received an additional round of downstream post-training<sup>4</sup>, whereas Qwen3 8B is the original Qwen3 release without this update<sup>5</sup>. This initial gap is present before any of our training and carries over to the trained models.

Training does not close this gap; it widens it. We train both models with the LCS-F1 reward and evaluate the checkpoints before and after training on two metrics (Table 17). The first is LCS-F1 itself, the graded reward used for training; the second is ESM (Exact Subsequence Match), a strict metric of whether the reference procedure is followed in full. Both models improve on the reward they are trained on, by 51.9 points for Qwen3 4B Instruct and 37.9 for Qwen3 8B. On ESM, however, the two diverge further, by 48.6 and 29.0 points respectively. Qwen3 4B Instruct converts its gain in partial credit into full procedural compliance, whereas Qwen3 8B learns the reward but converts less of it.

| Model             | Metric: LCS-F1 |       |          | Metric: ESM |       |          |
|-------------------|----------------|-------|----------|-------------|-------|----------|
|                   | Initial        | Final | $\Delta$ | Initial     | Final | $\Delta$ |
| Qwen3 4B Instruct | 35.2           | 87.1  | +51.9    | 18.9        | 67.5  | +48.6    |
| Qwen3 8B          | 27.5           | 65.4  | +37.9    | 11.4        | 40.4  | +29.0    |

Table 17: Average LCS-F1 and ESM on the validation set at the start and end of GRPO training with the LCS-F1 reward.

### E.2 Outcome-Based Does Not Always Lead to Task Success

To isolate the effect of reward design, we compare two families of trajectory-level rewards for GRPO (Fig. 9). *Procedure-aware* rewards score how faithfully a rollout reproduces the reference action path, using the sequence-matching rewards introduced in Sec. 5.1 (LCS-F1 and SR). *Outcome-based* rewards instead judge only the final result of a trajectory, ignoring the intermediate procedure. We consider two such signals: **Outcome.DB** (Database), which rewards a trajectory when its final database state exactly matches that of the reference, and **Outcome.LA** (Last Action), which rewards a trajectory when its last action matches the final action of the reference trajectory.

As shown in Fig. 9, optimizing for end outcomes alone does not reliably translate into actual task success. The procedure-aware reward variants consistently outperform the outcome-based baselines on both SR (Subsequence Recall) and the stricter ESM (Exact Subsequence Match), and both improve substantially over the course of training, recovering a larger portion of the reference workflow and more often converting that partial progress into full successful execution. Together, these results show that outcome supervision alone is not sufficient to induce procedural compliance, which requires a compliance-aware learning signal grounded in the reference path.

<sup>3</sup><https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507>

<sup>4</sup><https://github.com/QwenLM/qwen3>

<sup>5</sup><https://huggingface.co/Qwen/Qwen3-8B>

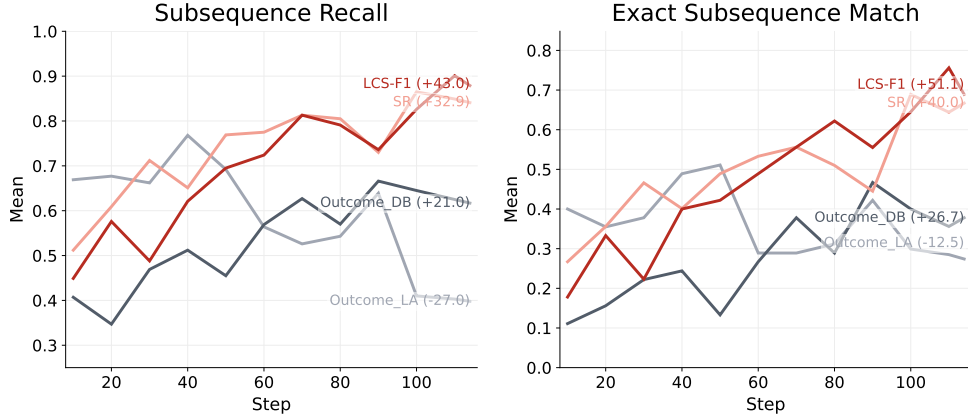


Figure 9: Comparison of procedure-aware and outcome-based reward designs during training, measured on the validation set. Procedure-aware rewards consistently achieve higher SR (Subsequence Recall) and ESM (Exact Subsequence Match) than outcome-based rewards, showing that optimizing only for end outcomes does not reliably induce successful execution of the full reference procedure.

## F Evaluation Benchmarks

### F.1 Benchmark Details

**Held-out Testset.** The held-out benchmark consists of 202 tasks drawn from 20 environments in our dataset generation pipeline. To minimize domain similarity with the training set, the test domains are selected to share as little domain knowledge as possible with those used for training. Each task is validated through both an automatic verification process (Sec. 4.3) and human annotation. For human annotation, annotators were provided with the policy document for each domain and asked to identify the sequential function-call sequence required to satisfy the user goal for each task instance. Because this was intended as a logical verification procedure, matching function arguments was not required; instead, annotators only needed to select the function names. A task instance was included in the held-out test set only if the annotator successfully executed all tool calls in the reference trajectory; otherwise, it was discarded. Through this process, 202 task instances were retained out of the 235 generated trajectories. Fig. 11 shows the annotation interface used for human validation. We report the average success rate over three independent trials (Avg@3).

**Tau2-bench.** Tau2-bench is a benchmark for evaluating whether an agent complies with policies and rules in real-world customer-support settings. We use two subsets: Retail (114 tasks) and Airline (50 tasks). We report the average score over four independent trials (Avg@4)<sup>6</sup>.

**SOPBench.** SOPBench is a benchmark for evaluating whether an agent adheres to domain-specific standard operating procedures (SOPs), policies, and constraints during tool use. We use 830 tasks spanning seven domains. SOPBench provides two success rate metrics: Macro, the average success rate across domains, and Micro, the success rate over all tasks from all domains. We report the mean of Macro Success Rate over three independent trials (Avg@3).

**FlowBench.** FlowBench is a benchmark for evaluating whether an agent correctly follows the intended workflow and invokes tools accurately at each step. We use the session-level single-scenario subset, comprising 228 sessions across 16 domains. We report the average of success rate over three independent trial (Avg@3).

<sup>6</sup>We follow the evaluation protocol of the original Tau2-bench paper, which reports results over four independent trials.

| Training Setup           |                     | Tau2-Retail               |             |     | Tau2-Airline              |             |      |
|--------------------------|---------------------|---------------------------|-------------|-----|---------------------------|-------------|------|
|                          |                     | Runs                      | Avg         | Std | Runs                      | Avg         | Std  |
| <b>Qwen3 4B Instruct</b> |                     | 37.7 / 40.4 / 41.2 / 45.6 | 41.2        | 2.8 | 26.0 / 18.0 / 26.0 / 26.0 | 24.0        | 3.5  |
| SFT                      | <i>Distill</i>      | 52.6 / 47.4 / 49.1 / 41.6 | 47.7        | 4.0 | 40.0 / 36.0 / 24.0 / 12.8 | 28.2        | 10.7 |
|                          | <i>Self-Distill</i> | 50.9 / 45.6 / 36.8 / 39.5 | 43.2        | 5.5 | 24.0 / 28.0 / 24.0 / 36.0 | 28.0        | 4.9  |
| PPO                      | <i>Step-Reward</i>  | 50.0 / 50.0 / 52.6 / 51.8 | <b>51.1</b> | 1.1 | 30.0 / 26.0 / 24.0 / 36.0 | 29.0        | 4.6  |
|                          | <i>Traj-Reward</i>  | 24.6 / 21.1 / 21.1 / 22.1 | 22.2        | 1.4 | 20.0 / 8.0 / 20.0 / 4.0   | 13.0        | 7.1  |
| GRPO                     | <i>Traj-Reward</i>  | 50.0 / 51.8 / 51.8 / 45.6 | 49.8        | 2.5 | 36.0 / 30.0 / 40.0 / 34.0 | <b>35.0</b> | 3.6  |
| <b>Qwen3 8B</b>          |                     | 32.5 / 30.7 / 28.9 / 33.3 | 31.4        | 1.7 | 16.0 / 16.0 / 20.0 / 16.0 | 17.0        | 1.7  |
| SFT                      | <i>Distill</i>      | 37.7 / 37.7 / 32.5 / 40.4 | 37.1        | 2.9 | 20.0 / 18.0 / 16.0 / 24.0 | 19.5        | 3.0  |
|                          | <i>Self-Distill</i> | 34.2 / 31.6 / 31.6 / 33.3 | 32.7        | 1.1 | 24.0 / 20.0 / 24.0 / 24.0 | 23.0        | 1.7  |
| PPO                      | <i>Step-Reward</i>  | 34.2 / 32.5 / 34.2 / 34.2 | 33.8        | 0.7 | 24.0 / 18.0 / 20.0 / 22.0 | 21.0        | 2.2  |
|                          | <i>Traj-Reward</i>  | 28.1 / 26.3 / 24.5 / 22.8 | 25.4        | 2.0 | 18.0 / 14.0 / 14.0 / 16.0 | 15.5        | 1.7  |
| GRPO                     | <i>Traj-Reward</i>  | 41.2 / 49.1 / 43.0 / 49.1 | 45.6        | 3.6 | 22.0 / 24.0 / 24.0 / 26.0 | 24.0        | 1.4  |

Table 18: Per-run results on Tau2-Retail and Tau2-Airline across four runs.

| Training Setup           |                     | Held-out Testset   |      |     | SOPBench           |      |     | FlowBench          |      |     |
|--------------------------|---------------------|--------------------|------|-----|--------------------|------|-----|--------------------|------|-----|
|                          |                     | Runs               | Avg  | Std | Runs               | Avg  | Std | Runs               | Avg  | Std |
| <b>Qwen3 4B Instruct</b> |                     | 20.9 / 24.3 / 23.9 | 23.0 | 1.9 | 31.8 / 32.3 / 32.4 | 32.2 | 0.3 | 80.7 / 77.2 / 82.0 | 80.0 | 2.5 |
| SFT                      | <i>Distill</i>      | 47.5 / 47.5 / 48.0 | 47.7 | 0.3 | 24.7 / 24.8 / 24.8 | 24.8 | 0.1 | 61.4 / 65.8 / 70.6 | 65.9 | 4.6 |
|                          | <i>Self-Distill</i> | 27.4 / 27.4 / 26.7 | 27.2 | 0.4 | 20.7 / 20.6 / 20.8 | 20.7 | 0.1 | 77.2 / 79.4 / 77.2 | 77.9 | 1.3 |
| PPO                      | <i>Step-Reward</i>  | 49.5 / 47.5 / 45.3 | 47.4 | 2.1 | 45.5 / 45.9 / 45.8 | 45.7 | 0.2 | 88.2 / 85.5 / 79.8 | 84.5 | 4.3 |
|                          | <i>Traj-Reward</i>  | 24.0 / 23.9 / 25.4 | 24.4 | 0.8 | 11.1 / 11.5 / 12.1 | 11.6 | 0.5 | 78.9 / 78.9 / 75.9 | 77.9 | 1.8 |
| GRPO                     | <i>Traj-Reward</i>  | 55.0 / 54.5 / 54.2 | 54.5 | 0.4 | 48.1 / 48.5 / 48.7 | 48.5 | 0.3 | 78.9 / 81.6 / 83.3 | 81.3 | 2.2 |
| <b>Qwen3 8B</b>          |                     | 18.8 / 21.8 / 20.8 | 20.5 | 1.5 | 6.0 / 5.9 / 5.3    | 5.8  | 0.4 | 61.0 / 58.3 / 65.8 | 61.7 | 3.8 |
| SFT                      | <i>Distill</i>      | 42.1 / 42.1 / 44.6 | 42.9 | 1.4 | 13.6 / 13.3 / 13.8 | 13.6 | 0.2 | 63.2 / 59.6 / 66.7 | 63.2 | 3.5 |
|                          | <i>Self-Distill</i> | 16.3 / 16.3 / 13.9 | 15.5 | 1.4 | 5.3 / 8.9 / 8.4    | 7.6  | 1.9 | 65.4 / 65.4 / 68.0 | 66.2 | 1.5 |
| PPO                      | <i>Step-Reward</i>  | 23.3 / 23.3 / 26.2 | 24.3 | 1.7 | 7.5 / 7.6 / 7.3    | 7.5  | 0.1 | 64.5 / 57.0 / 61.4 | 61.0 | 3.7 |
|                          | <i>Traj-Reward</i>  | 15.3 / 16.2 / 16.1 | 15.9 | 0.5 | 2.3 / 2.3 / 2.3    | 2.3  | 0.0 | 47.3 / 51.9 / 55.7 | 51.6 | 4.2 |
| GRPO                     | <i>Traj-Reward</i>  | 24.8 / 31.2 / 26.2 | 27.4 | 3.4 | 20.4 / 21.0 / 21.5 | 20.9 | 0.6 | 64.9 / 71.1 / 66.2 | 67.4 | 3.2 |

Table 19: Per-run results on Held-out Testset, SOPBench, and FlowBench.

## F.2 Full Evaluation Results

Tables 18 and 19 provide the complete evaluation results across all benchmarks and training configurations, supplementing Table 3.

## F.3 Detailed Metrics for Each Benchmark

### F.3.1 Held-out Testset.

Because the held-out data are generated from workflow instances that are already separated by terminal outcome, each example belongs by construction to either the RESOLVED subset or the TRANSFER subset. In other words, this partition is inherited directly from the data generation process rather than introduced post hoc at evaluation time. Accordingly, in Table 20, we supplement the overall held-out success rate with RESOLVE SR and TRANSFER SR, which measure success rate on the corresponding subsets.

Several observations emerge from the detailed results. First, on the in-distribution held-out testset, our best trained variant (Qwen3 4B Instruct + GRPO with trajectory-level reward) achieves a 54.5% overall success rate, which is comparable to frontier models such as GPT 5.4 (53.5%), DeepSeek V3.2 Thinking (54.1%), and GLM-5 (53.6%).

| External Models           |              |            |             | Our Trained Models       |              |             |             |
|---------------------------|--------------|------------|-------------|--------------------------|--------------|-------------|-------------|
| Model                     | Success Rate | RESOLVE SR | TRANSFER SR | Model                    | Success Rate | RESOLVE SR  | TRANSFER SR |
| <b>Proprietary Models</b> |              |            |             | <b>Qwen3 4B Instruct</b> | <b>23.0</b>  | <b>21.9</b> | <b>24.1</b> |
| GPT 5.4                   | 53.5         | 57.6       | 49.5        | + SFT (Distill)          | 47.7         | 46.4        | 48.9        |
| Claude Sonnet 4.6         | 50.7         | 47.5       | 53.9        | + SFT (Self-Distill)     | 27.2         | 26.9        | 27.4        |
| Grok-4                    | 51.1         | 47.1       | 55.1        | + PPO (Step-Reward)      | 47.4         | 50.5        | 44.5        |
| Gemini 2.5 Pro            | 32.7         | 29.4       | 36.2        | + PPO (Traj-Reward)      | 24.4         | 25.3        | 23.5        |
| <b>Open-Source Models</b> |              |            |             | + GRPO (Traj-Reward)     | 54.5         | 56.2        | 52.9        |
| DeepSeek V3.2             | 54.1         | 54.8       | 53.3        | <b>Qwen3 8B</b>          | <b>20.5</b>  | <b>22.2</b> | <b>18.8</b> |
| GLM-5                     | 53.6         | 55.2       | 52.1        | + SFT (Distill)          | 42.9         | 43.8        | 42.1        |
| Qwen3.5 397B A17B         | 50.0         | 45.2       | 54.8        | + SFT (Self-Distill)     | 15.5         | 12.8        | 18.1        |
| Minstral 14B              | 30.1         | 30.5       | 29.9        | + PPO (Step-Reward)      | 24.3         | 27.9        | 20.7        |
| Kimi K2 Thinking          | 28.6         | 32.8       | 24.2        | + PPO (Traj-Reward)      | 15.9         | 16.9        | 12.0        |
| Llama 4 Maverick          | 13.1         | 8.9        | 16.9        | + GRPO (Traj-Reward)     | 27.4         | 27.3        | 27.5        |
| Llama 3.1 8B Instruct     | 0.0          | 0.0        | 0.0         |                          |              |             |             |

Table 20: Detailed held-out benchmark results. The left table reports external models, split into proprietary and open-source groups. The right table reports our trained variants, where gray-colored rows indicate the corresponding base models. All values are reported as percentages. We report the overall success rate on the full held-out set, as well as RESOLVE SR and TRANSFER SR on the held-out subsets corresponding to the RESOLVED and TRANSFER branches, respectively. For reasoning models, reasoning was enabled using the default reasoning effort.

| Training Setup           |                     | Macro Success Rate  | Micro Success Rate  | Procedure Compliance Rate | Constraint Compliance Rate |
|--------------------------|---------------------|---------------------|---------------------|---------------------------|----------------------------|
| <b>Qwen3 4B Instruct</b> |                     | <b>32.2</b>         | <b>29.6</b>         | <b>42.3</b>               | <b>63.6</b>                |
| SFT                      | <i>Distill</i>      | 24.8 (-7.4)         | 24.9 (-4.7)         | 41.0 (-1.3)               | 53.9 (-9.7)                |
|                          | <i>Self-Distill</i> | 20.7 (-11.5)        | 20.9 (-8.7)         | 31.2 (-11.1)              | 51.8 (-11.8)               |
| PPO                      | <i>Step-Reward</i>  | 45.7 (+13.5)        | 44.5 (+14.9)        | 63.2 (+20.9)              | 72.3 (+8.7)                |
|                          | <i>Traj-Reward</i>  | 11.6 (-20.6)        | 9.9 (-19.7)         | 19.6 (-22.7)              | 42.9 (-20.7)               |
| GRPO                     | <i>Traj-Reward</i>  | <b>48.5 (+16.3)</b> | <b>47.4 (+17.8)</b> | <b>66.3 (+24.0)</b>       | <b>73.9 (+10.3)</b>        |
| <b>Qwen3 8B</b>          |                     | <b>5.8</b>          | <b>4.9</b>          | <b>10.0</b>               | <b>30.9</b>                |
| SFT                      | <i>Distill</i>      | 13.6 (+7.8)         | 11.2 (+6.3)         | 19.2 (+9.2)               | 37.3 (+6.4)                |
|                          | <i>Self-Distill</i> | 7.6 (+1.8)          | 6.1 (+1.2)          | 11.2 (+1.2)               | 32.6 (+1.7)                |
| PPO                      | <i>Step-Reward</i>  | 7.5 (+1.7)          | 6.2 (+1.3)          | 11.8 (+1.8)               | 31.8 (+0.9)                |
|                          | <i>Traj-Reward</i>  | 2.3 (-3.5)          | 2.2 (-2.7)          | 9.9 (-0.1)                | 26.8 (-4.1)                |
| GRPO                     | <i>Traj-Reward</i>  | <b>20.9 (+15.1)</b> | <b>15.9 (+11.0)</b> | <b>25.6 (+15.6)</b>       | <b>40.5 (+9.6)</b>         |

Table 21: Detailed results on SOPBench.

Second, training with our pipeline yields balanced improvements across both RESOLVE and TRANSFER subsets, indicating that the learned procedural compliance generalizes across both outcome types rather than being biased toward one terminal branch. Third, among the external baselines, we observe notable variation in the balance between RESOLVE SR and TRANSFER SR—for instance, Claude Sonnet 4.6 performs better on TRANSFER cases (53.9%) than on RESOLVE cases (47.5%), whereas DeepSeek V3.2 Thinking is relatively balanced. This suggests that different models exhibit different procedural biases, and that evaluating both subsets separately provides a more informative picture of compliance behavior than the aggregate score alone.

### F.3.2 SOPBench.

Table 21 breaks down SOPBench performance into four complementary metrics. Macro Success Rate is the same metric reported in the main table, namely the domain-wise representative score averaged across domains. Micro Success Rate is the fraction of interactions

that are finally marked successful. A trajectory is counted as successful only when all evaluation conditions are satisfied, including the absence of tool-call errors, constraint satisfaction, database-state correctness, correct action invocation, and directed-graph consistency. Procedure Compliance Rate isolates whether the required procedural order is followed, i.e., whether prerequisite actions are satisfied in the directed action graph before later actions are taken. Constraint Compliance Rate measures whether the model avoids violating environment and tool-level constraints.

Taken together, the detailed SOPBench results show a consistent pattern across both models: effective training improves not only final success rates, but also the stronger procedural correctness and constraint adherence. In addition to gains in Macro Success Rate and Micro Success Rate, we also observe corresponding improvements in Procedure Compliance Rate and Constraint Compliance Rate. This indicates that the benefit of our training is not limited to increasing the chance of task completion, but extends to following the required procedural order more faithfully and respecting environment and tool-level constraints more reliably. In this sense, the detailed breakdown provides supporting evidence that our training improves genuine SOP-following behavior, rather than merely boosting end-task success in a superficial way.

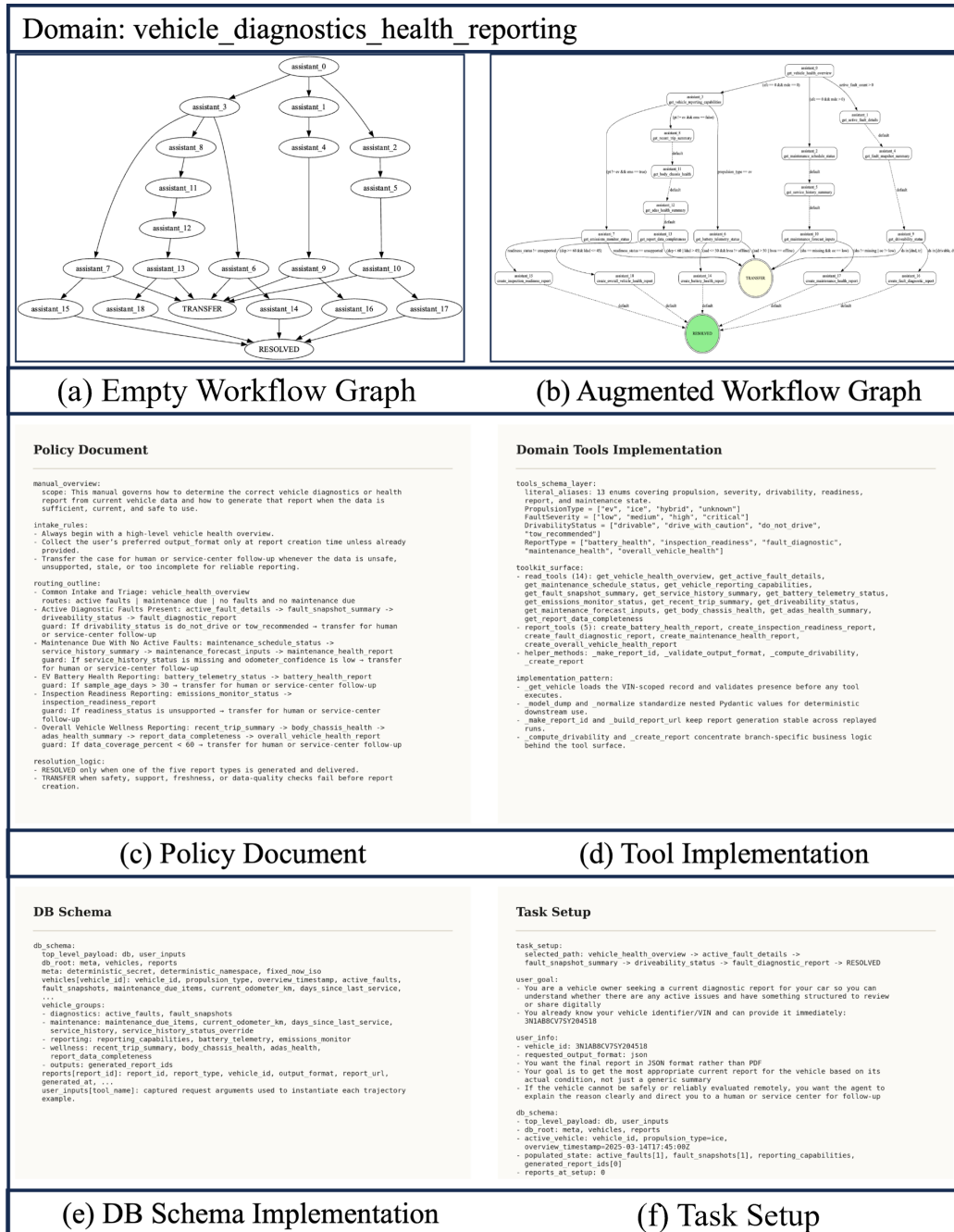


Figure 10: End-to-end example of a generated environment and task for the “vehicle diagnostics health reporting” domain. **(a)** Empty workflow graph skeleton with abstract topology. **(b)** Augmented workflow graph with domain-specific function definitions and branch conditions. **(c)** Natural-language policy document paraphrased from the augmented graph. **(d)** Executable Python tool implementations. **(e)** Database schema backing the tool environment. **(f)** A concrete task instance, including user goal, user context, and database records, grounded in a specific reference path.

## G Prompts and Human Annotation Interfaces

```

# Task: Generate a Complete SOP (Standard Operating Procedure) Graph

## Overview
You are an AI workflow designer. Your task is to transform a Skeleton Workflow Graph into a fully annotated SOP Graph that an AI agent can follow to resolve diverse user intents.

## Input: Skeleton Workflow Graph
You will receive a Skeleton Workflow Graph in JSON format. Nodes are named assistant_n, edges represent control flow, and terminal nodes are RESOLVED or TRANSFER. A node that branches to TRANSFER should retrieve domain data, and the edge condition should express the business rule in terms of that data.

## Your Assignment: SOP Graph Completion
Transform the skeleton graph into a complete SOP by annotating each node with: (1) a well-designed Tool Name, (2) properly defined Input Arguments, and (3) a meaningful Response Schema with branching conditions.

## Step 1: Design Tool Names
Define clear, descriptive names using the verb.noun pattern. Prefer READ operations first and then WRITE operations. Do not design assistant_0 as a request-classification or intent-routing function; start with a meaningful READ operation that gathers actual system state.

## Step 2: Define Input Arguments
For each argument, define the argument name, data type, source type, required flag, and optional default value.

### Supported Data Types
{SUPPORTED_DATA_TYPES_TABLE}

### Source Type Definitions
{SOURCE_TYPE_DEFINITION_TABLE}

### Reference Path Syntax (for context type)
{CONTEXT_REFERENCE_PATH_TABLE}

## Step 3: Design Response Schema and Edge Conditions
The response schema determines control flow and data flow. When a node has only one outgoing edge, use condition = "default". When a node has multiple outgoing edges, conditions must be mutually exclusive and exhaustive. A tool's response schema should describe domain data returned by the system; the edge condition is where the business rule is applied.

### Supported Comparison and Logical Operators
{SUPPORTED_COMPARISON_OPERATORS_TABLE}
{LOGICAL_OPERATORS_TABLE}

## Output Format
Return the final SOP Graph in NetworkX node-link JSON format using nodes and edges. Create a node entry for every assistant_n node in the skeleton workflow graph. No justification. No meta-reasoning. Output valid JSON only.
{OUTPUT_JSON_SCHEMA_TEMPLATE}

## Input Data
### Domain
{DOMAIN}

### Workflow Graph Structure
'''json
{WORKFLOW_SKELETON}
'''

```

Table 22: Prompt for workflow graph augmentation.

```

# Task
Given {DOMAIN} and {ANNOTATED_WORKFLOW}, derive a self-contained Markdown manual/policy pair. The workflow is structurally and
semantically defined, so role transitions, branching, convergence, and termination must be inferred from it rather than invented.
The annotation provides domain intent, tool descriptions, arguments, response fields, and transition conditions; do not invent
concrete APIs, databases, or other implementation details beyond it.

## Output Rules
Return only two wrapped blocks, manual and policy, with no extra text. The output must not mention graphs, nodes, edges, steps, or
the conversion process, and it must not include meta commentary or justification.

## Stage 1: Manual Generation
Write a standalone operator-facing resolution manual that explains what to ask, verify, and do; what information is required; how
decision branches route; what counts as successful resolution; and when the case must be transferred or escalated.

## Stage 2: Policy Generation
Rewrite the manual as a role-oriented internal policy for an autonomous assistant. Preserve the same recoverable workflow logic,
but express it as role identity, obligations, decision principles, and domain constraints rather than step-based troubleshooting.

## Required Policy Content
Define the assistant's mission, authority boundaries, core functions, confirmation before persistent state changes,
non-fabrication, one-action-at-a-time discipline, policy enforcement, and human handoff conditions. Also encode domain entities,
attributes, lifecycle states, time-based logic, and any baseline record structure implied by the workflow. For each function,
specify preconditions, required information, blocking conditions, transfer criteria, post-action effects, and branching,
validation, exception, irreversible-action, and safety/compliance rules in abstract policy language.

## Final Instruction
Generate a realistic, domain-consistent manual and policy that jointly encode both operational responsibilities and the implied
control flow. Return only the wrapped manual and policy blocks.

```

Table 23: Prompt for document generation.

```

# Task: Generate tools.py and database schema
Read {WORKFLOW_MANUAL}, the annotated workflow {ANNOTATED_WORKFLOW}, and the sample format at {SAMPLE_TOOLS_PATH}; produce
exactly one file, /mnt/data/tools.py, in the exact style of sample_tools.py. Use code interpreter to read and write files, and
do not emit any additional files.

## Authoritative Inputs
The workflow graph is authoritative for the tool surface: implement every and only the tools appearing in nodes[*].tool.name;
method names, argument names, required-vs-optional status, defaults, and response fields must match the graph exactly. Terminal
states are not implemented as tools.

## Implementation Requirements
Define Database inheriting BaseData and Toolkit inheriting BaseToolkit. Decorate each public tool with @tool(...) and a logical
ToolType. WRITE tools must mutate self.db; validations, eligibility rules, refusals, and state transitions must follow the
policy. Raise ValueError for invalid inputs, missing entities, or policy violations.

## Schema Discipline
Map graph types to compatible Python types. Return JSON-serializable values consistent with each tool's response schema, and
always include fields referenced by edge conditions. Use explicit Literal[...] values for closed, branch-critical categories and
lifecycle states rather than free-form strings or booleans.

## Determinism
Keep normal DB-driven logic, but replace inherently non-deterministic IDs, tokens, references, and time with deterministic
helpers derived from stable arguments, DB identifiers, and metadata such as deterministic_secret, deterministic_namespace, or
fixed_now_iso. Do not use wall-clock time, UUID randomness, or global RNG state for branch-relevant outputs.

## Tool Surface Quality
Each tool docstring must include Description, Args, Returns, and Raises in structured form. Private helper methods are allowed.
Do not reference workflow generation, NetworkX, or meta-reasoning in code comments or docstrings.

## Required Validation
After writing /mnt/data/tools.py, repeatedly run: (1) python -m py_compile for syntax, (2) AST checks confirming that Database
and Toolkit exist with correct inheritance, decorators, and signatures for every workflow tool, and (3) runtime smoke tests
using stubbed environment.tools modules so the file imports without project dependencies. Fix the file until all checks pass.

```

Table 24: Prompt for tool implementation and database schema synthesis.

```

# Task: Design deterministic db and user_inputs for one scenario
Given the environment code {TOOLS_PY} and one expected execution trajectory {TRAJECTORY}, produce a single JSON object that
deterministically reproduces that scenario in the verification environment.
## Source of Truth
Use only the provided tools/schema code and the scenario JSON. Scenario expected_user_inputs values are placeholders that show
shape and type only; do not copy them. Generate new coherent values that satisfy tool logic and remain consistent with the database.
## Analysis Requirements
Inspect the Database schema and the tools used in the trajectory to determine which DB fields are read, written, and returned,
which values are sourced from user_input versus context(ref), and which branch conditions must hold at each step.
## Critical Branch Control
Identify the decisive step whose response determines whether the path ends in TRANSFER or continues to RESOLVED. Determine the
exact returned field and value that control this branch, then set the DB so that the tool returns the value required by the
scenario terminal. This branch-setting field is the highest-priority design constraint.
## Output Object
Return one valid JSON object with exactly two top-level fields: db, the minimal deterministic initial database state conforming to
Database; and user_inputs, containing every scenario input marked as user_input. The DB must be just large enough for all prior
tool calls to produce the referenced context(ref) values and satisfy every branch condition.
## Determinism and Validity
Do not invent unsupported fields, use randomness, or rely on current time. If the schema exposes deterministic metadata such as
deterministic_secret, deterministic_namespace, or fixed_now_iso, populate them with stable values sufficient for deterministic ID
or time generation. All field names and types must match Database, and the JSON must be strictly valid.
## Final Output Rule
Return only a JSON code block of the form {"db": {...}, "user_inputs": {...}} with no explanation text before or after.

```

Table 25: Prompt for deterministic database instance and user-provided context generation.

```

# Task: Generate a user-simulator role description
Given assistant policy {POLICY}, an expected resolution flow {RESOLUTION_FLOW}, advance user knowledge {KNOWN_INFORMATION}, and
relevant tool descriptions {TOOL_DESCRIPTION}, write a role description for the user simulator so that, when the simulator follows
that role and the agent follows policy, the conversation naturally proceeds along the intended path.
## Role Framing
Write in second person as instructions to the simulator, not as a message sent to the agent. Describe who the user is, the
situation they are in, what outcome they want, and what information they already possess and can provide.
## Compatibility Constraints
The role must be compatible with the policy and naturally induce the target resolution flow, but it must not mention tool names,
tool calls, policies, schemas, internal mechanics, or workflow steps. The user should appear unaware of the agent's internal
process.
## Required Content
Reflect all necessary known information as facts the user already knows, such as personal details, IDs, dates, preferences,
constraints, or supporting evidence. Describe the user's situation and goal only at the natural-language level, and include
decision preferences for alternatives or fallback handling, including preference for escalation to a human when unresolved.
## Output Rule
Return exactly one JSON dictionary of the form {"user_task_description": "<USER_TASK_DESCRIPTION>"} and nothing else.

```

Table 26: Prompt for user goal generation.

```

# Task: Predict all immediate next functions or terminal states from a workflow manual
System instruction: you are a strict workflow QA evaluator; answer only with JSON, with no markdown and no explanation.
## Inputs
Given workflow manual {WORKFLOW_MANUAL}, function catalog {FUNCTION_CATALOG} serialized in OpenAI tools format, current
function name {CURRENT_FUNCTION_NAME}, and current function description {CURRENT_FUNCTION_DESCRIPTION}, infer the set of all
possible immediate next functions or terminal states for the current node.
## Inference Rules
Base the answer only on the workflow manual, not on workflow edges or ground-truth transitions. The predicted next_function must
be either a function name that appears in {FUNCTION_CATALOG} or a terminal state RESOLVED/TRANSFER. Also provide a short
routing_condition describing when that next step is taken.
## Evaluation Note
The script records both next_function and routing_condition, but automatic scoring uses only the predicted next-function set;
routing conditions are retained for human inspection.
## Output Format
Return exactly one JSON object of the form {"next_steps": [{"next_function": "function_or_terminal_state", "routing_condition":
"condition"}, ...]} and nothing else.

```

Table 27: Prompt for next node prediction in document-graph consistency validation.

```

# Task: Ground one user utterance against the original task and user-known information
Given the original task description {USER_TASK_DESCRIPTION}, the flattened candidate reference facts {USER_INFO_CANDIDATES},
and the current user utterance {CURRENT_USER_UTTERANCE}, classify the utterance using any applicable labels from task_request,
info_provision, and none.
## Label Semantics
task_request means the user is asking the assistant to do something, continue the workflow, produce an output, take an action,
or otherwise advance the task. info_provision means the utterance provides task-relevant information already known to the user
and supported by the given user_info. none covers acknowledgements, social talk, filler, emotion, or content unrelated to the
task or reference information.
## Decision Rules
Judge using only the initial user task, the provided user-info candidates, and the single current user utterance; do not use
previous or future turns. Both positive labels may be assigned when the utterance simultaneously contains a request and
supported user-known information. Only assign info_provision when the content matches or paraphrases {USER_INFO_CANDIDATES}.
Use none only when neither positive label applies.
## Evidence and Output

Return one JSON object with field labels and optional fields evidence_task_request and evidence_info_provision. Evidence fields
should contain short quoted snippets and should appear only when the corresponding label is present.

```

Table 28: Prompt for utterance type identification.

```

# Task: Verify whether grounded evidence remains in scope of the original task/reference information
Step 2 is dispatched after the grounding stage and uses one of two prompt variants depending on which Step 1 label was assigned.
Both variants return a single JSON object with entailment in {"entailed", "not_entailed"} and a short reason.
## Variant A: task_request entailment
Given {USER_TASK_DESCRIPTION} and {USER_UTTERANCE}, decide whether the instruction requested in the utterance is included in,
closely aligned with, or reasonably inferred from the original task description without introducing a materially different
request. Judge only against the original task description, focus on intended scope rather than exact wording, and mark
not_entailed when the user adds a new request or meaningfully changes the task. Although the implementation passes extracted
evidence items into this stage, this prompt variant does not actually expose them to the model.
## Variant B: info_provision entailment
Given {USER_TASK_DESCRIPTION}, {USER_INFO_CANDIDATES}, the immediately preceding assistant utterance {ASSISTANT_UTTERANCE},
the current user utterance {USER_UTTERANCE}, and the extracted evidence bundle {EVIDENCE_ITEMS}, decide whether the combined
evidence is supported by the original reference information. Mark entailed when the evidence matches, paraphrases, or can be
directly inferred from the provided user-info candidates, optionally supported by the task description; mark not_entailed when
any material part of the evidence introduces unsupported information.
## Decision Rules

Variant A judges against only the original task description. Variant B judges against only the provided user-info candidates and
task description, with the one-turn assistant-user dialogue included as local conversational context. In both cases, the model
must return exactly one JSON object and no extra explanation text.

```

Table 29: Prompt for entailment evaluation.

**Workflow Annotation** wtf / trajs 1/7 User Manual

**AVAILABLE TOOLS**

- analyze\_backend\_connectivity**  
Diagnose why clients still cannot us...
- apply\_connectivity\_fix**  
Apply the proposed network-policy ...
- assess\_pod\_readiness**  
Determine whether unhealthy pods ...
- check\_selector\_alignment**  
Evaluate whether the current Servic...
- diagnose\_dns\_connectivity**  
Run DNS and in-cluster reachability...
- find\_mysql\_workload**  
Search the namespace for the most...
- get\_mysql\_service**  
Retrieve the authoritative Kubernetes...
- get\_service\_endpoints**  
Inspect the current endpoint state a...
- inspect\_mysql\_listener\_config**  
Inspect a representative selected M...
- inspect\_network\_policies**  
Inspect namespace NetworkPolicy...
- inspect\_workload\_rollout**  
Inspect the workload owner and sel...
- list\_selected\_pods**  
List the pods currently matched by L...
- patch\_service\_selector**  
Update only the Service selector so...

**User Task**

You are the application engineer on call for the payments system. An internal Kubernetes MySQL service used by the ledger component is unavailable, and you want help restoring service connectivity. Your concern is service reachability only, not MySQL data, queries, or schema contents. You know the target identifiers and can provide them when asked: the namespace is payments-core and the Service name is ledger-mysql. You do not know the current selector, pod labels, workload details, or exact root cause. You want the agent to investigate and lead the diagnosis once it has the identifiers. You prefer the least invasive, evidence-based fix first and do not want broad, destructive, or speculative cluster changes. You are willing to approve a narrowly scoped change if the agent explains why it is needed and what it will modify. If a single targeted fix does not restore the service, you want the case escalated or handed off to a human platform/database engineer rather than continuing with unrelated changes. You communicate concisely, answer direct questions, and only provide information you actually know.

**get\_mysql\_service** Step 1

Arguments

Result

```

{
  "cluster_ip": "10.96.64.12",
  "exists": true,
  "namespace": "payments-core",
  "ports": [
    {
      "name": "mysql",
      "port": 3306,
      "protocol": "TCP",
      "target_port": 3306
    }
  ],
  "selector": {},
  "service_name": "ledger-mysql",
  "service_type": "ClusterIP"
}

```

Step 2: What is the next tool to call?

Type a tool name.

Submit

Correct! Step 1/7 complete.

Figure 11: Annotation interface used for human validation of held-out test set tasks.