AREWARD: SYNTHESIZING REAL-WORLD ENVIRONMENTS WITH EXECUTABLE STATE RUBRICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents learn through interaction with stateful environments, but sparse outcome rewards provide limited feedback on intermediate progress. Rubric-based rewards offer finer-grained evaluation, yet their criteria must reflect user requirements and be checked reliably against execution. We introduce **AReward**, which jointly synthesizes transactional environments, tasks, and executable state rubrics through a shared formulation of create, read, update, and delete (CRUD). Tool execution determines state changes, and task requirements specify the conditions that count as completion. Reference solution replay verifies task reachability, while reward auditing examines the consistency of acceptance criteria with user requirements. Evaluating these rubrics after each action exposes progress and reversals in rubric satisfaction. **STG-GRPO**, our state-transition graph variant of Group Relative Policy Optimization, uses this evidence and progress along the reference state chain to refine trajectory comparisons. Combined with supervised initialization, reinforcement learning with a staged curriculum, and long-tail task management, AReward provides a training framework in which reward criteria are tied to user requirements and their evaluation is grounded in actual execution. Code is available at `https://anonymous.4open.science/r/stg-anonymous-248B/`.

1 INTRODUCTION

Humans pursue complex goals by observing the effects of their actions and using this feedback to guide subsequent decisions. Progress depends on understanding both how an environment changes and how those changes relate to the intended outcome. Language agents face a similar need when interacting with tools and users over multiple steps (Yao et al., 2024; Barres et al., 2025; He et al., 2025). Reinforcement learning offers a way to improve such decisions through experience, but sparse outcome rewards, assigned at the end of a trajectory based on task success, provide limited guidance about individual actions (Wang et al., 2025; Feng et al., 2025). Trajectories that satisfy different subsets of a task's requirements may receive the same failure signal, making credit assignment difficult.

Rubric-based rewards address part of this problem by decomposing an objective into separately assessed requirements (Gunjal et al., 2025; Huang et al., 2025; Shao et al., 2025). This structure makes completion criteria explicit and represents partial satisfaction, but the usefulness of the feedback depends on both the criteria and their evaluation. Criteria must reflect user intent, cover necessary constraints, and preserve legitimate alternative solutions. Their evaluation must establish whether the required outcomes actually occurred. Research on rubric generation and reward hacking shows that incomplete or misaligned criteria and inaccurate judgments can produce different forms of unreliable feedback (Shen et al., 2026; Mahmoud et al., 2026). Improving the evaluator alone does not settle what should count as success.

The development of interactive environments makes these questions increasingly actionable. Persistent databases and programmatic tests support reproducible assessment of task outcomes (Trivedi et al., 2024). Automated pipelines now generate executable environments, task instances, and verifiers, while language world models offer a complementary route to scalable interaction (Zhang et al., 2025; Wu et al., 2026; Dong et al., 2026; Zuo et al., 2026). These approaches broaden access to training environments, with different trade-offs in controllability, state access, and execution fi-

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

delity. Within executable environments, a central design problem is to keep user requirements, tool effects, and acceptance criteria consistent as they are generated and used for learning. This places reward specification alongside environment construction.

Our central insight is that environment dynamics and task requirements can share the same state representation. In transactional tool environments, *create, read, update, and delete* (CRUD) operations define transitions between database states. State differences record the effects of tool calls, and executable rubrics check whether the resulting states satisfy the user's requirements.

We introduce **AReward**, which jointly synthesizes databases, tools, tasks, executable state rubrics, and reference solutions (Figure 1). Execution tests check tool behavior, and reference solution replay verifies task reachability. Reward auditing checks that acceptance criteria follow the request, preserve valid alternatives, and avoid credit for initially satisfied conditions. Diagnostic feedback guides revisions before training.

These artifacts provide progress feedback during learning. Evaluating the same rubrics after each call identifies newly satisfied requirements and requirements invalidated by subsequent actions. Replayed reference states provide a complementary progress coordinate. Building on group-relative optimization and state-based trajectory comparison (Shao et al., 2024; Feng et al., 2025; Cheng et al., 2026), we introduce **STG-GRPO**. It groups steps within each task's rollout group by post-action position on the reference state chain and the number of unsatisfied core criteria. Normalized step-level contributions are summed into a trajectory-level advantage and combined with trajectory-level feedback. Isolated environment execution and replay-based scoring connect these signals to the actual tool calls. Training combines supervised initialization, a staged RL curriculum, and long-tail task management.

Experiments with Qwen3.6-27B show improvements in task completion and interaction efficiency. On the 400-task VitaBench, SFT reaches a task success rate of 54.5%, a 9.5 percentage-point gain over Base. The final RL checkpoint achieves 53.8%, remaining 8.8 points above Base. Delivery and In-store each reach 67.0%, improving over Base by 14.0 and 12.0 percentage points, respectively. Delivery also gains 4.0 points over SFT while reducing mean tool calls from 15.4 to 14.5. On the evaluated 489-instance τ-bench suite, the task success rate reaches 73.0%, compared with 67.3% for Base. Telecom shows a joint gain over SFT under matched user-simulator settings: the success rate increases from 81.6% to 86.8%, while mean tool calls decrease from 7.1 to 5.9, a 16.9% reduction. These results demonstrate successful task execution with fewer tool interactions in this domain.

Rubric-level analysis also reveals progress beyond binary task success. On 100 VitaBench Cross-domain tasks, the earlier RL-50 checkpoint satisfies 1,290 of 1,441 rubric criteria, compared with 1,263 for SFT. Both checkpoints solve 44 tasks, but RL-50 satisfies 27 additional criteria, raising the rubric satisfaction rate from 87.65% to 89.52%. This difference makes partial improvements visible even when the number of fully completed tasks is unchanged. State-based case analysis complements these aggregate measures by tracing how agents establish required conditions, retain completed subgoals, and reach the requested final state.

2 AREWARD

AReward synthesizes tool-use environments together with tasks, executable state rubrics, and reference solutions. We focus on transactional domains, where agents query and modify persistent business records such as orders, payments, and reservations. Tool execution and task evaluation share the same database state (Figure 1). CRUD operations define how actions access and change the environment. Rubrics express which resulting conditions satisfy the user's requirements, connecting task semantics to the execution evidence used for reward.

2.1 ENVIRONMENT SYNTHESIS

Environment synthesis starts from scenario seeds that describe real-world service workflows, such as ordering food, reserving accommodation, changing a flight, and managing an account. Each seed abstracts a workflow into business entities, user needs, available operations, and operational constraints. These specifications guide the construction of executable environments and tasks with concrete business goals.

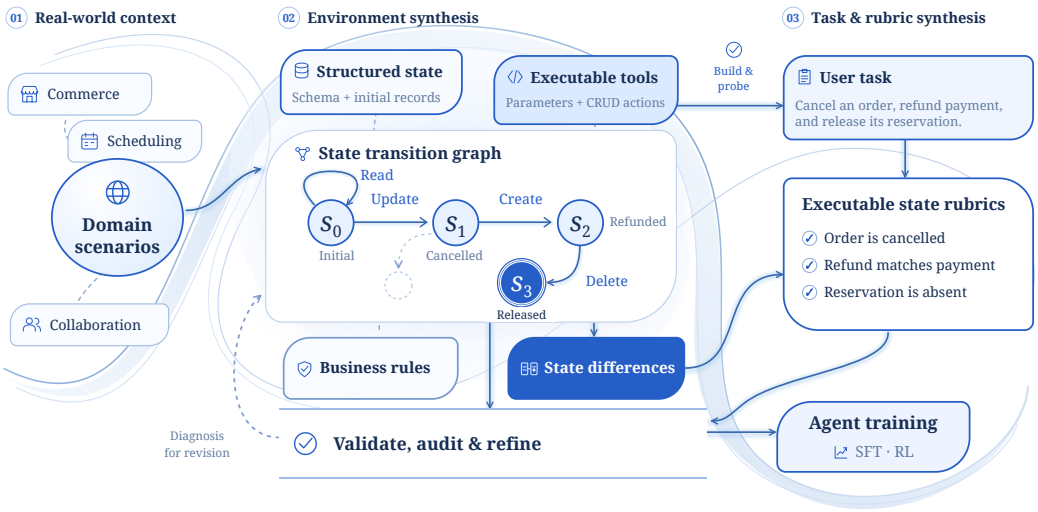


Figure 1: **Overview of AReward.** Scenarios guide environment and task synthesis. In the schematic graph, nodes are database states and edges are tool calls. State rubrics check task requirements, while the feedback loop represents validation, auditing, and refinement before training.

Given a scenario seed, the generator constructs an environment $\mathcal{E} = (\Sigma, s_0, \mathcal{U})$, where Σ is a relational database schema, s_0 is the initial database state, and $\mathcal{U}$ is the set of executable tools. Let $\mathcal{S}_\Sigma$ denote the database states consistent with Σ. Each state $s \in \mathcal{S}_\Sigma$ contains records identified by table and primary key, with fields and relations representing business properties. Initial records supply the objects and alternatives needed for retrieval and decision-making. An environment policy exposes the applicable business rules to the agent.

Each tool implements a parameterized composition of *create, read, update, and delete* (CRUD) operations. Create inserts records, read retrieves records or query results, update changes existing fields, and delete removes records. These operations provide a common execution model across domains while preserving business-level tool interfaces. For example, a refund tool can insert a refund record and update an associated order within one call. Tool definitions specify argument types, preconditions, declared effects, and executable database operations. Preconditions constrain when a call is admissible, and declared effects specify the state conditions expected after successful execution. The database operations of a write call execute within one transaction.

At call index t, the agent selects an action $a_t = (u_t, \mathbf{x}_t)$, where $u_t \in \mathcal{U}$ is a tool and $\mathbf{x}_t$ is its argument assignment. Execution yields a successor state and an observation:

$$s_t = T_{\mathcal{E}}(s_{t-1}, a_t), \qquad \omega_t = O_{\mathcal{E}}(s_{t-1}, a_t, s_t). \tag{1}$$

Here $T_{\mathcal{E}}$ is the deterministic state-transition function defined by the tool implementations, $O_{\mathcal{E}}$ is the observation function, and ω_t is the returned query result, execution feedback, or error. Reads leave $s_t = s_{t-1}$ while providing information through ω_t. Rejected calls and failed transactions preserve the prior state and return failure feedback. The agent observes the policy, task request, tool schemas, and tool responses. Full database states are available to the execution and verification system.

We associate each action with its observed state difference,

$$\delta_t = \mathrm{Diff}(s_{t-1}, s_t), \qquad s_{t-1} \xrightarrow{a_t,\, \delta_t} s_t, \tag{2}$$

where Diff compares records by primary key and records field values before and after execution, including changes caused by insertion and deletion. The environment induces a directed state-transition graph: states are nodes, tool calls are edges, and each edge carries its state difference. One edge can contain several record or field changes. Reads and rolled-back calls have empty state differences and form self-loops, with their observations distinguishing information retrieval from failure. Generated tools define the transition rules, and execution materializes a trajectory through the graph. Each transition provides state evidence for task verification and reward computation.

2.2 Task Synthesis and Executable State Rubrics

For each environment, the generator constructs tasks together with executable state rubrics and reference tool sequences. A task q expresses the user's goal and constraints in natural language. Rubrics translate these requirements into conditions on business records, and a reference sequence supplies an executable route to the requested outcome. All three follow the environment specification and business rules. Task difficulty is controlled through subgoal composition, dependencies, search and comparison requirements, and distractors. The agent receives the request and environment interfaces, while reference solutions and acceptance checks are reserved for verification.

For a task with a database-state goal, an executable state rubric comprises core criteria

$$\mathcal{R}_q^{\text{core}} = (f_1, \ldots, f_M), \qquad f_j : \mathcal{S}_\Sigma \to \{0, 1\}, \quad j = 1, \ldots, M, \tag{3}$$

where $M \geq 1$ is the number of core criteria and $f_j(s) = 1$ means that state s satisfies the jth requirement. We omit the task subscript on f_j when q is fixed. Each criterion is implemented as a check using one of five declarative forms: record existence, record absence, a field-value condition, an aggregate constraint, or a read-only query assertion. They connect CRUD effects to business requirements. Creation can be checked through the existence of a qualifying record, updates through its properties, and deletion through the absence of matching records. Aggregate and query assertions express amounts, counts, and relationships across records. Bonus criteria encode optional preferences separately from the core criteria.

The cancellation example in Figure 1 requires an order to reach its cancelled state, a corresponding refund to satisfy the applicable amount constraint, and the reservation to be released. These requirements yield distinct checks on the affected records. A call may satisfy several checks, and one check may require several calls. Rubric synthesis follows the user's constraints: when several choices are valid, checks describe their shared requirements. For example, a booking can be accepted by its requested time, service type, and budget. A newly created record is identified through task-relevant properties. Free-form text is checked by exact match only when its wording is explicitly specified, and mandatory constraints belong among the core criteria.

Evaluating the core criteria at state s_t yields a satisfaction vector and the number of satisfied criteria:

$$\mathbf{b}_q(s_t) = \big(f_1(s_t), \ldots, f_M(s_t)\big), \qquad c_t = \sum_{j=1}^{M} f_j(s_t). \tag{4}$$

The core criteria are all satisfied when $c_T = M$, where T is the final tool-call index. Applying the same checks before and after an action yields

$$\Delta\mathbf{b}_{q,t} = \mathbf{b}_q(s_t) - \mathbf{b}_q(s_{t-1}). \tag{5}$$

Each component is $+1$ for a newly satisfied requirement, 0 for unchanged satisfaction, and -1 for a requirement that ceases to hold. The state difference δ_t records the database changes, and $\Delta\mathbf{b}_{q,t}$ measures their effect on rubric satisfaction. Checks execute against the full state, which supplies the unchanged records and relations needed to interpret those changes. Their outputs identify individual completed or violated requirements and yield the count increment $c_t - c_{t-1}$ used in Section 3.2. Queries and refusals additionally require assessment of the agent's answer or refusal, as described in Section 3.1.

2.3 Validation and Reward Auditing

Environment verification checks that the schema can be instantiated, tool parameters agree with executable operations, and contract references resolve to valid tables and fields. Dynamic checks execute tools with constructed arguments and compare the resulting state with the declared effects. They also test whether preconditions reject inadmissible calls. Failed candidates receive localized diagnostics for refinement within a fixed budget, after which unresolved candidates are discarded. The same verification routines are used by the generation process and by the final collection gate.

Task verification checks rubric references and expressions, then replays the reference tool sequence from s_0. The replay must execute successfully and satisfy every core criterion at its final state. For state-changing training tasks, each core criterion must also be unsatisfied initially, preventing

progress credit for conditions already true before interaction. Recording the initial state and the state after each successful reference write yields $\mathcal{C}_q = (s_0^\star, \ldots, s_L^\star)$, where $s_0^\star = s_0$ and L is the number of reference write calls. Full database snapshot hashes identify these states. Reads preserve persistent state and are omitted from this chain. The replay supplies a verified path through the environment's transition graph and a reference for the progress signal used during training.

Reward auditing examines whether executable checks faithfully express the request and business rules. It combines rule-based scans with inspection of flagged tasks and failed trajectories. The audit targets checks that fix one choice among several valid solutions, require unsupported exact matches to agent-authored text, or award credit for initially satisfied conditions. It also examines whether required facts are accessible through tools, target values follow from the request and available information, and the constraints can be jointly met. Findings guide revisions to the environment, task, or rubric before training. Accepted tasks retain executable checks and replay evidence, allowing subsequent rollouts to be evaluated in the same state representation used during synthesis.

3 Agent Training

We train the agent in two stages. Supervised fine-tuning first teaches the model to follow business policies, invoke environment-specific tools, and respond to execution feedback. Reinforcement learning then improves long-horizon decisions on more difficult tasks. This progression follows the supervised-initialization and RL structure used in prior tool-learning systems (Dong et al., 2025), while both stages in AReward operate on the same executable environments and task semantics.

3.1 Supervised Fine-Tuning

We construct supervised demonstrations by running DeepSeek V4 Pro as a teacher agent inside the synthesized environments. The teacher receives the environment policy, user request, and environment-scoped tool schemas. It interacts with a user simulator when clarification is needed and observes the actual results of every tool call. Reference solutions and executable rubrics are reserved for verification and are not included in the policy prompt.

We retain trajectories that pass task-specific verification. For modify and create tasks, the final database state must satisfy every core criterion. Query tasks must preserve the database and return an answer consistent with results obtained by replaying the reference read operations. Refusal tasks must preserve the state and comply with the environment policy. Accepted trajectories are serialized as multi-turn conversations containing assistant reasoning, tool calls, tool outputs, and user responses. Given a supervised set $\mathcal{D}_{\mathrm{SFT}}$, we optimize the standard next-token objective over the designated assistant targets:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\sum_{(x,y)\in\mathcal{D}_{\mathrm{SFT}}} \sum_{j\in\mathcal{M}(y)} \log \pi_\theta(y_j \mid x, y_{<j}), \tag{6}$$

where $\mathcal{M}(y)$ is the set of supervised token positions selected by the loss mask. This stage provides the policy used to initialize reinforcement learning. Appendix A.1 describes trajectory construction and validation.

3.2 Reinforcement Learning

In long-horizon tool use, sparse outcome rewards provide little guidance about which intermediate actions advance or reverse task progress. AReward addresses this problem with evidence already produced by its environments. Each CRUD action produces a concrete database state that can be checked against the task's executable rubrics. Figure 2 shows how these signals enter the policy update.

State-grounded reward. For task q, executing a rollout o yields $\tau = (s_0, a_1, s_1, \ldots, a_T, s_T)$, where a_t is a tool call and s_t is the database state after executing it. Replaying the reference solution yields a reference state chain $\mathcal{C}_q = (s_0^\star, \ldots, s_L^\star)$. Read operations do not create nodes in this chain because they leave the persistent state unchanged. A successful write adds the resulting state, represented by a hash of the full database snapshot.

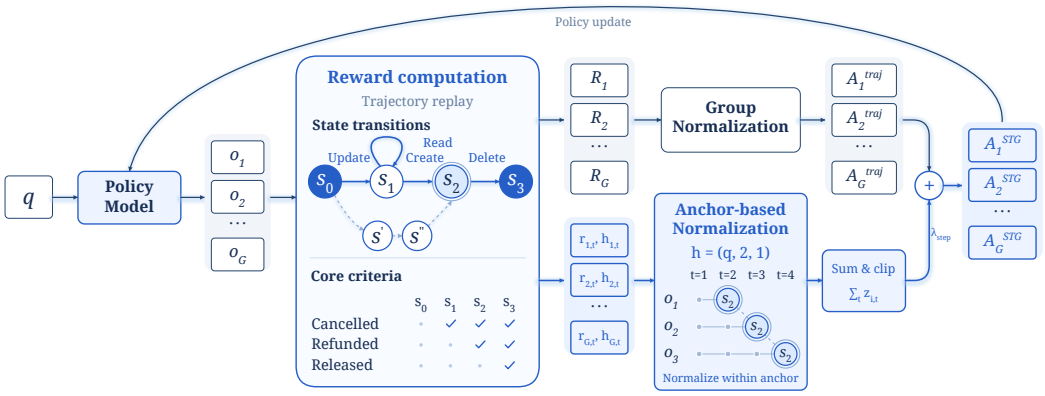


Figure 2: **STG-GRPO.** Replaying trajectories o_i yields returns R_i, step-level rewards $r_{i,t}$, and anchors $h_{i,t}$. The upper and lower branches compute trajectory-level and anchor-grouped contributions to A_i^{STG}, respectively (Eq. (9)). Highlighted nodes show matching anchors reached at different steps.

AReward assigns each action a reward from four forms of execution evidence:

$$r_t = \lambda_g r_t^{\mathrm{graph}} + \lambda_r r_t^{\mathrm{rubric}} + \lambda_e r_t^{\mathrm{exec}} + \lambda_o r_t^{\mathrm{outcome}}. \tag{7}$$

The graph-progress reward measures normalized progress along $\mathcal{C}_q$. Returning to an earlier reference state produces negative progress, while an off-chain state receives no graph-progress credit. The rubric-progress reward measures the change in the fraction of satisfied core criteria. It rewards any state that fulfills additional task requirements, including valid states that do not lie on the reference chain. The execution reward scores tool-call validity and execution success, and penalizes failed or repetitive operations. The outcome reward encodes binary task success at the final action, as determined by satisfaction of all core criteria. All four reward components are computed from tool execution. Appendix A.3 gives the exact weights and boundary cases.

STG-GRPO. Group Relative Policy Optimization (GRPO) compares several responses sampled for the same prompt using normalized trajectory returns (Shao et al., 2024). This comparison remains coarse when two long trajectories receive similar returns after reaching different intermediate states. We introduce **STG-GRPO**, which augments the trajectory-level advantage with comparisons aligned by progress along the reference state chain and rubric satisfaction.

After action a_t, we assign the step an anchor

$$h_t = \big(q,\, p_t,\, M - c_t\big), \tag{8}$$

where p_t is the latest matching position on the reference state chain (-1 for an off-chain state), M is the number of core criteria, and c_t is the number currently satisfied. The anchor summarizes the task, post-action position on the reference state chain, and number of unsatisfied core criteria. For rollout i in a same-prompt group, STG-GRPO computes

$$A_i^{\mathrm{STG}} = A_i^{\mathrm{traj}} + \lambda_{\mathrm{step}} \,\mathrm{clip}\left(\sum_t \mathrm{Norm}_{h_{i,t}}(r_{i,t}), -3, 3\right), \tag{9}$$

where A_i^{traj} is the standard group-normalized trajectory-level advantage. Norm_h standardizes step-level rewards among visits to the same anchor within the prompt group. This compares rewards for actions that reach the same post-action anchor, while retaining the global signal from eventual task completion.

The resulting trajectory-level advantage is broadcast over the valid response tokens. STG-GRPO uses step-level state evidence to refine trajectory comparison, while keeping optimization at the response level. During training, parallel agent workers sample multi-turn trajectories in isolated environments, and reward workers replay completed trajectories to recover state transitions, rubric satisfaction, and anchors. We train with four rollouts per task and move through staged task pools. Appendix A describes the full training system, optimization settings, staged curriculum, and long-tail task management.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We compare three checkpoints of Qwen3.6-27B: the base model, the SFT checkpoint (402), and the RL checkpoint (stage10-59). VitaBench (He et al., 2025) contains 400 tasks, with 100 each in Delivery, In-store, OTA, and Cross-domain. Our τ-bench evaluation combines τ^2-bench (Barres et al., 2025) with the official `banking_knowledge` domain from τ-Knowledge (Shi et al., 2026), distributed in the same repository. It covers 489 instances: Airline (50), Retail (114), Telecom (114), Telecom-workflow (114), and Banking (97). Telecom and Telecom-workflow use the same task set under different policies. Overall task success rates weight evaluation instances; score differences are in percentage points (pp).

For the τ-bench runs, decoding uses temperature 0, a generation cap of 24,576 tokens, enabled reasoning, and inline tool-call parsing. The user simulator is Qwen3.6-27B, with a 200-step interaction limit and one trial per instance. Banking uses BM25 retrieval and 262,144-token serving contexts; the other configurations use 65,536. SFT/RL simulator configurations match on Telecom and Telecom-workflow. Airline and Retail expose no user-side domain tools but use different recorded simulator variants. Daggers in Figure 3 mark these differences. Evaluation records lack the effective Airline/Retail prompts and the Banking grading revision, and do not establish protocol matching on VitaBench.

4.2 BENCHMARK RESULTS

Figure 3 reports aggregate and domain-level task success rates for all three checkpoints, with separate Base-to-SFT and SFT-to-RL transitions.

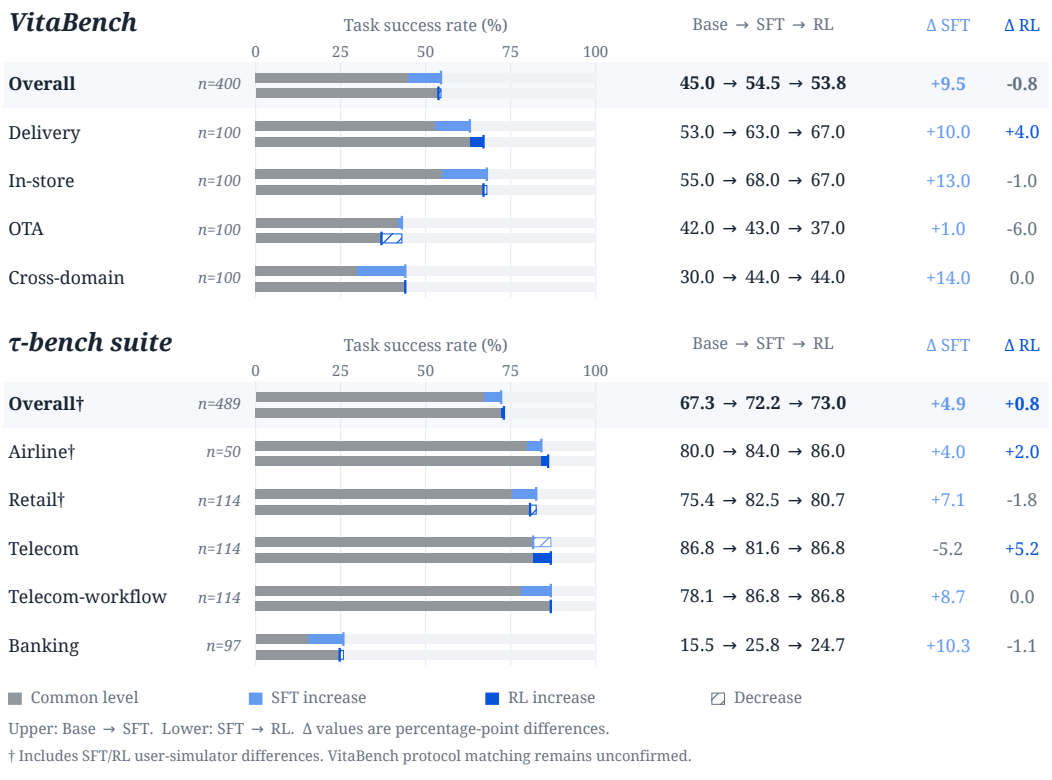


Figure 3: **Checkpoint comparison on VitaBench and the τ-bench suite.** Each row shows Base-to-SFT above SFT-to-RL. Gray marks the shared score, blue an increase, and hatching a decrease. ΔSFT is SFT minus Base; ΔRL is RL minus SFT. The τ-bench aggregate includes 489 instances with Banking from τ-Knowledge. Daggers mark simulator differences (Section 4.1).

VitaBench. The overall task success rates are 45.0% for Base, 54.5% for SFT, and 53.8% for RL, corresponding to 180, 218, and 215 successful tasks. SFT is 9.5 pp above Base, with higher success rates in every domain. RL remains 8.8 pp above Base but is 0.8 pp below SFT. Relative to SFT, its Delivery success rate increases by 4.0 pp, In-store decreases by 1.0 pp, OTA decreases by 6.0 pp, and Cross-domain remains at 44.0%.

τ-bench suite. The three checkpoints solve 329, 353, and 357 of 489 instances, giving overall task success rates of 67.3%, 72.2%, and 73.0%. The observed differences are +4.9 pp from Base to SFT and +0.8 pp from SFT to RL, calculated before rounding the displayed percentages. Within the two domains with matched SFT/RL simulator configurations, Telecom rises from 81.6% to 86.8% (+5.2 pp), while Telecom-workflow remains at 86.8%. Airline and Retail change by +2.0 and -1.8 pp, respectively. Banking contains 97 instances, with Base solving 15 (15.5%), SFT solving 25 (25.8%), and RL solving 24 (24.7%). In these domains and the aggregate, the effects of training and simulator configuration cannot be separated.

SFT-to-RL differences vary by domain: Delivery and Telecom improve, Cross-domain and Telecom-workflow remain stable, and In-store, OTA, and Banking decline. Component-level attribution requires matched-data, matched-budget ablations; uncertainty estimates require per-task outcomes or repeated runs.

4.3 Interaction Cost and Rubric Satisfaction

Table 1 reports mean agent turns and tool calls per task. A turn is one assistant message; calls are counted individually, including multiple calls in one message. User-simulator actions are excluded. Domain means weight domains equally; aggregate success rates weight evaluation instances.

Table 1: **Agent turns and tool calls per task.** RL denotes stage10-59. Four-domain means exclude Banking.

	Base		SFT-402		RL	
Domain	Turns	Calls	Turns	Calls	Turns	Calls
VitaBench						
Delivery	17.2	13.6	22.1	15.4	21.6	14.5
In-store	17.4	12.2	21.6	13.0	20.5	12.2
OTA	21.0	23.3	33.1	24.0	32.8	23.4
Cross-domain	23.1	19.6	29.9	21.1	28.8	19.5
Four-domain mean	19.67	17.16	26.68	18.36	25.92	17.41
τ-bench evaluation						
Airline	11.1	5.6	9.7	8.8	10.2	8.0
Retail	12.9	6.9	11.2	7.8	11.3	7.7
Telecom	14.4	4.5	14.2	7.1	13.8	5.9
Telecom-workflow	12.9	4.0	17.5	5.9	15.7	5.6
Four-domain mean	12.83	5.25	13.15	7.40	12.75	6.80
Banking	20.0	14.4	22.8	22.3	22.1	21.0

Interaction cost. On VitaBench, RL uses 2.8% fewer agent turns and 5.2% fewer tool calls than SFT, with the overall success rate 0.8 pp lower. Delivery improves by 4.0 pp while calls fall from 15.4 to 14.5; Cross-domain retains a 44.0% success rate with calls falling from 21.1 to 19.5. In-store and OTA reductions accompany lower success rates.

Across the four non-Banking τ^2 configurations, RL uses 3.0% fewer turns and 8.1% fewer calls than SFT. Telecom provides a joint improvement under matched simulator settings: the success rate rises from 81.6% to 86.8%, while calls fall from 7.1 to 5.9 (16.9%). Base reaches the same 86.8% success rate with 4.5 calls. In Banking, calls decrease from 22.3 to 21.0, alongside a success-rate decrease from 25.8% to 24.7%.

Supplementary checkpoint analysis: RL-50. On 100 VitaBench Cross-domain tasks, Base, SFT-402, and the earlier RL-50 checkpoint satisfy 1,190, 1,263, and 1,290 of 1,441 rubric criteria:

82.58%, 87.65%, and 89.52%. SFT and RL-50 both solve 44 tasks, yet RL-50 satisfies 27 more criteria (+1.87 pp). This criterion-weighted satisfaction rate captures partial task completion beyond binary task success. Counts of satisfied criteria for the main RL checkpoint, stage10-59, are unavailable.

4.4 Case Analysis

We examine how Base and RL turn the same inventory observations into different booking states. Figure 4 follows a VitaBench task requiring oxygen-equipped accommodation from May 1 to May 5, covering four nights without changing rooms.

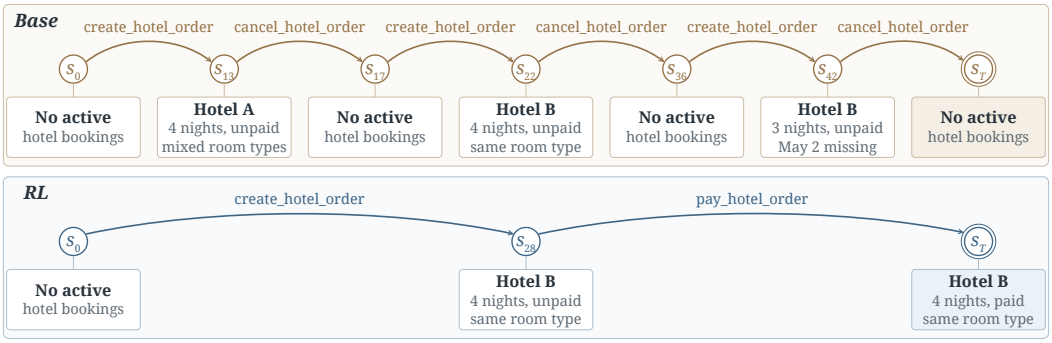


Figure 4: **Hotel booking states for VitaBench task 10807007.** Following Section 2, s_t denotes the state after tool call t and s_T the final state of each run. Boxes show active hotel bookings. Intermediate calls are omitted, and each arrow groups calls to the named tool. Empty-booking nodes may differ in other database fields.

Room availability. Hotel A lacks the selected room type on May 4. Hotel B has one room type available on all four dates, with daily inventory of 3, 1, 3, and 2. Both models receive identical records. Base books three nights in one room type and the final night in another. The user rejects this arrangement, and Base cancels all four orders. RL checks all four dates before selecting Hotel B, then creates and pays for four orders in the same room type. Its booking satisfies the benchmark's four-night, same-room-type condition.

Booking and payment. Base subsequently books Hotel B, but cancels after the user demands a physical-room guarantee. A retry fails for May 2, and the remaining bookings are cancelled. Base finishes with no active hotel booking after 35 write attempts and 16 cancellations across all domains. RL finishes with four paid nights after 12 write calls and no cancellations. Satisfied rubric criteria are 4/11 versus 11/11. Later user responses differ between runs.

The paths illustrate the distinction between creating records and retaining a completed subgoal. AReward's rubric-progress reward captures both: establishing required conditions earns progress, while invalidating them removes it.

5 Conclusion

We presented AReward for jointly synthesizing executable environments, tasks, and executable state rubrics, and STG-GRPO for using rubric satisfaction and progress along the reference state chain in agent training. The final RL checkpoint solves 215 of 400 VitaBench tasks, with Delivery and In-store each reaching 67.0% success. On Telecom, it improves over SFT by 5.2 percentage points while using 16.9% fewer tool calls under matched simulator settings. State-based case analysis reveals how agents establish and invalidate task conditions beyond binary task success. Future work will isolate the reward and optimization components under matched training budgets and evaluate generalization across more diverse environments.

AI USE STATEMENT

During environment and task construction, generative models supported the synthesis of database schemas, tool implementations, business policies, user requests, executable state rubrics, and reference solutions. A teacher model generated supervised demonstrations through tool interaction, and model-based user simulators supported trajectory collection and evaluation. During manuscript preparation, generative AI assisted with literature search and citation checks, drafting and revising text, translating case excerpts, interpreting recorded trajectories, and creating figure code. Generated environments and reference solutions undergo the execution and verification procedures described in the paper. The authors direct the revisions and remain responsible for the manuscript's claims, citations, results, and content.

ETHICS STATEMENT

This work studies agent training in synthesized transactional environments and established benchmark simulators. User interactions are simulated, and tool calls operate on sandbox records rather than live customer accounts or commercial services. The reported orders, payments, and reservations therefore do not initiate real transactions. Generated scenarios and simulated users may reflect biases in the source models, while executable rubrics assess only the conditions they specify. Benchmark success should not be treated as evidence of safe deployment in live services. The framework is intended for controlled research and evaluation; applications involving real users would require separate assessment of authorization, privacy, and unintended state changes.

REPRODUCIBILITY STATEMENT

Section 2 describes environment and task synthesis, executable state rubrics, reference solution replay, and reward auditing. Section 3 specifies supervised initialization and STG-GRPO, and Section 4.1 reports benchmark configurations and protocol differences. Appendix A details trajectory collection, isolated execution, reward computation, training settings, and the curriculum. Appendix B provides task examples with their acceptance checks, and Appendix C presents state graphs with source-indexed interaction excerpts. These descriptions connect the reported results to the task definitions, execution procedures, and evaluation settings. The anonymous code repository is `https://anonymous.4open.science/r/stg-anonymous-248B/`.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025. doi: 10.48550/arXiv.2506.07982.

Xin Cheng, Shuo He, Lang Feng, HaiYang Xu, Ming Yan, Lei Feng, and Bo An. Beyond Trajectory-Level Attribution: Graph-Based Credit Assignment for Agentic Reinforcement Learning. *arXiv preprint arXiv:2605.26684*, 2026. doi: 10.48550/arXiv.2605.26684.

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, Jiajie Jin, Yutao Zhu, Hanbin Wang, Fangyu Lei, Qinyu Luo, Mingyang Chen, Zehui Chen, Jiazhan Feng, Ji-Rong Wen, and Zhicheng Dou. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026. doi: 10.48550/arXiv.2604.18292.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-Group Policy Optimization for LLM Agent Training. *arXiv preprint arXiv:2505.10978*, 2025. doi: 10.48550/arXiv.2505.10978.

Shubham Gandhi, Saurabh Goyal, Kiran Kate, and Yara Rizk. DRACO: Fine-Grained Credit Assignment with Dynamic Rubrics for Long-Horizon Agent Training. *arXiv preprint arXiv:2609.04094*, 2026. doi: 10.48550/arXiv.2609.04094.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains. *arXiv preprint arXiv:2507.17746*, 2025. doi: 10.48550/arXiv.2507.17746.

Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. LLM-Rubric: A Multidimensional, Calibrated Approach to Automated Evaluation of Natural Language Texts. *arXiv preprint arXiv:2501.00274*, 2024. doi: 10.48550/arXiv.2501.00274.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. VitaBench: Benchmarking LLM Agents with Versatile Interactive Tasks in Real-world Applications. *arXiv preprint arXiv:2509.26490*, 2025. doi: 10.48550/arXiv.2509.26490.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, Xijun Gu, Peiyi Tu, Jiaxin Liu, Wenyu Chen, Yuzhuo Fu, Zhiting Fan, Yanmei Gu, Yuanyuan Wang, Zhengkai Yang, Jianguo Li, and Junbo Zhao. Reinforcement learning with rubric anchors. *arXiv preprint arXiv:2508.12790*, 2025. doi: 10.48550/arXiv.2508.12790.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. Simulating Environments with Reasoning Models for Agent Training. *arXiv preprint arXiv:2511.01824*, 2025. doi: 10.48550/arXiv.2511.01824.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. *arXiv preprint arXiv:2305.20050*, 2023. doi: 10.48550/arXiv.2305.20050.

Bowen Lv, Xiao Liu, Yanyu Ren, Hanyu Lai, Bohao Jing, Hanchen Zhang, Yanxiao Zhao, Shuntian Yao, Jie Tang, and Yuxiao Dong. SCALECUA: Scaling computer use agents with verifiable task synthesis and efficient online rl. *arXiv preprint arXiv:2607.11185*, 2026.

Anas Mahmoud, MohammadHossein Rezaei, Zihao Wang, Anisha Gunjal, Bing Liu, and Yunzhong He. Reward Hacking in Rubric-Based Reinforcement Learning. *arXiv preprint arXiv:2605.12474*, 2026. doi: 10.48550/arXiv.2605.12474.

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen-tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, Hannaneh Hajishirzi, and Pang Wei Koh. DR Tulu: Reinforcement learning with evolving rubrics for deep research. *arXiv preprint arXiv:2511.19399*, 2025. doi: 10.48550/arXiv.2511.19399.

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

William F. Shen, Xinchi Qiu, Chenxi Whitehouse, Lisa Alazraki, Shashwat Goel, Francesco Barbieri, Timon Willi, Akhil Mathur, and Ilias Leontiadis. Rethinking Rubric Generation for Improving LLM Judge and Reward Modeling for Open-ended Tasks. *arXiv preprint arXiv:2602.05125*, 2026. doi: 10.48550/arXiv.2602.05125.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024. doi: 10.1145/3689031.3696075.

Quan Shi, Alexandra Zytek, Pedram Razavi, Karthik Narasimhan, and Victor Barres. τ-Knowledge: Evaluating conversational agents over unstructured knowledge. *arXiv preprint arXiv:2603.04370*, 2026. doi: 10.48550/arXiv.2603.04370.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling Tool-Interactive Environments for LLM Agent via Programmatic Synthesis. *arXiv preprint arXiv:2601.05808*, 2026. doi: 10.48550/arXiv.2601.05808.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents. *arXiv preprint arXiv:2407.18901*, 2024. doi: 10.48550/arXiv.2407.18901.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning. *arXiv preprint arXiv:2504.20073*, 2025. doi: 10.48550/arXiv.2504.20073.

Yifan Wu, Yiran Peng, Yiyu Chen, Jianhao Ruan, Zijie Zhuang, Cheng Yang, Jiayi Zhang, Man Chen, Yenchi Tseng, Zhaoyang Yu, Liang Chen, Yuyao Zhai, Bang Liu, Chenglin Wu, and Yuyu Luo. AutoWebWorld: Synthesizing Infinite Verifiable Web Environments via Finite State Machines. *arXiv preprint arXiv:2602.14296*, 2026. doi: 10.48550/arXiv.2602.14296.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. *arXiv preprint arXiv:2406.12045*, 2024. doi: 10.48550/arXiv.2406.12045.

Huan Zhang, Mingju Chen, Dongxu Zhou, Can Lv, Heng Chang, Sen Cui, Faguo Wu, and Shiji Zhou. TRCA: Transition-wise Rubric Credit Assignment for Long-horizon LLM Agents. *arXiv preprint arXiv:2608.16156*, 2026. doi: 10.48550/arXiv.2608.16156.

Jiayi Zhang, Yiran Peng, Fanqi Kong, Cheng Yang, Yifan Wu, Zhaoyang Yu, Jinyu Xiang, Jianhao Ruan, Jinlin Wang, Maojia Song, HongZhang Liu, Xiangru Tang, Bang Liu, Chenglin Wu, and Yuyu Luo. AutoEnv: Automated Environments for Measuring Cross-Environment Agent Learning. *arXiv preprint arXiv:2511.19304*, 2025. doi: 10.48550/arXiv.2511.19304.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, and Steven Hoi. MAI-UI technical report: Real-world centric foundation gui agents. *arXiv preprint arXiv:2512.22047*, 2025.

Yuxin Zuo, Zikai Xiao, Li Sheng, Fei Huang, Jianhong Tu, Yuxuan Liu, Tianyi Tang, Xiaomeng Hu, Yang Su, Qingfeng Lan, Yantao Liu, Qin Zhu, Yinger Zhang, Bowen Yu, Haiquan Zhao, Haiyang Xu, Jianxin Yang, Jiayang Cheng, Junyang Wang, Lianghao Deng, Mingfeng Xue, Tianyi Bai, Yang Fan, Yubo Ma, Yucheng Li, Zeyu Cui, Zhihai Wang, Zhihui Xie, Zhuorui Ye, An Yang, Dayiheng Liu, Jingren Zhou, and Ning Ding. Qwen-AgentWorld: Language World Models for General Agents. *arXiv preprint arXiv:2606.24597*, 2026. doi: 10.48550/arXiv.2606.24597.

A AGENT TRAINING DETAILS

The two training stages share the same environment contracts, tool interfaces, and execution semantics. Supervised fine-tuning learns the interaction protocol from accepted teacher trajectories. Reinforcement learning then uses executable state evidence to improve decisions on harder tasks.

A.1 SUPERVISED FINE-TUNING DATA

Executable trajectory collection. For each environment, we construct a policy that describes available operations, required confirmations, and refusal conditions. It is generated from the environment definition and task descriptions without task-specific rubrics or reference solutions. DeepSeek V4 Pro serves as the teacher, interacting with the executable tools and an independent user simulator in an isolated database. Read calls return query results; write calls return state differences or precondition errors. We record every user, assistant, and tool turn, including assistant reasoning and tool arguments.

Task-specific acceptance. We select demonstrations according to task semantics. Modify and create trajectories pass when all core state rubrics hold in the final database. Query trajectories must preserve the database and answer consistently with replayed reference reads. Refusal trajectories must preserve the state and comply with the environment policy. Bonus rubrics do not determine acceptance. Failed trajectories are retained for diagnosis and excluded from SFT.

Serialization and split. Accepted trajectories become ShareGPT-style conversations. The system message contains the environment policy. Assistant turns preserve reasoning and Qwen3-formatted calls, while tool turns contain rows, state differences, or errors. Each sample carries its environment's OpenAI-compatible tool schemas. A render–parse round trip checks tool names and arguments. We split by environment identifier, keeping all tasks from one environment in the same partition, and train with the established last-round loss mask.

A.2 REINFORCEMENT LEARNING INFRASTRUCTURE

Asynchronous stateful rollouts. The training system is built on verl (Sheng et al., 2024) with a vLLM rollout engine. The rollout layer dispatches multiple agent loops asynchronously across inference and environment workers. Reward evaluation is parallelized over completed trajectories. This arrangement limits idle time during multi-turn environment interaction.

Each rollout receives an isolated in-memory SQLite instance initialized from the environment schema and seed records. Read tools execute read-only queries. Write tools check their parameters and preconditions, execute within a savepoint, and return the resulting state difference. Failed writes are rolled back. The policy receives only the user request and the tool interfaces available in that environment. Reference solutions and acceptance rubrics remain in training metadata and are not exposed in the prompt.

Tool schemas are constructed for each environment because the same tool name may take different parameters across environments. The resulting interface matches the executor assigned to that rollout, which validates each call against its tool definition.

Replay-based reward computation. Online tool execution returns observations but does not inject a shaped reward at each turn. After generation, a reward worker initializes a fresh copy of the same environment and replays the valid tool calls in order. It records the database snapshot, number of satisfied core rubrics, and execution status after every call. The resulting trace is the sole input to the reward in Eq. (7). Deterministic replay keeps rollout execution and reward evaluation under the same environment semantics.

Tool calls are parsed independently, so a malformed call contributes no transition while other valid calls are replayed. Illegal tools and failed executions remain visible to the execution reward. A reward-side exception returns a zero score for that sample.

Table 2: RL settings shared across curriculum stages, taken from the stage launch configurations.

Setting	Value
Policy initialization	Qwen3.6-27B with an SFT LoRA adapter
Training hardware	One node with 8 H200 GPUs
Rollout engine	vLLM, tensor parallel size 8
Rollout memory fraction	0.70, eager execution
Task batch / rollouts	16 tasks / 4 rollouts per task
Agent / reward workers	8 / 8
Maximum assistant turns	25
Prompt / response limits	8,192 / 16,384 tokens
Engine context limit	98,304 tokens
LoRA rank	64
PPO mini-batch	16
Per-GPU actor micro-batch	2
Per-GPU log-prob micro-batch	2
Checkpoint interval / retention	10 steps / 3 checkpoints

Table 2 lists the settings that are explicit in all four stage launch configurations. The first stage starts from the SFT adapter. Each later stage resumes from the checkpoint produced by the preceding stage, so the task distribution changes without resetting the policy.

A.3 Reward and STG-GRPO Implementation

Reference state chain. For each task, we replay the reference solution from the initial database. The chain contains the initial state and the state after every successful write. Reads are omitted because they do not change persistent state. Snapshot hashes map an executed state to its latest matching position in the chain. Let p_t be this position and let $\bar{p}_{t-1}$ be the last matched position. The graph term is

$$r_t^{\text{graph}} = \begin{cases} (p_t - \bar{p}_{t-1})/\max(L, 1), & p_t \geq 0, \\ 0, & p_t = -1, \end{cases} \tag{10}$$

where $p_t = -1$ denotes an off-chain state. After a matched step, $\bar{p}_t = p_t$; after an off-chain step, $\bar{p}_t = \bar{p}_{t-1}$. A later return to an earlier chain state can therefore receive a negative graph increment.

Executable rubric progress. Let c_t be the number of core rubrics satisfied after action t and let M be the total number of core rubrics. Training tasks pass an initial-state gate that requires every core rubric to be unsatisfied before execution. The rubric term is

$$r_t^{\text{rubric}} = (c_t - c_{t-1})/M, \qquad c_0 = 0. \tag{11}$$

The term is positive when an action establishes a required condition and negative when it invalidates one. Bonus rubrics are retained for task analysis but do not enter the RL reward.

Execution and outcome terms. A successful first use of a read tool receives 0.1, capped at ten distinct read-tool names per trajectory. An illegal tool receives -0.3, and a failed execution receives -0.2. A repeated run of successful calls to the same write tool receives -0.5 from the third call onward. The outcome term is 1 only at the last replayed action when every core rubric is satisfied. With the weights used in training,

$$r_t = 0.5 r_t^{\text{graph}} + 0.5 r_t^{\text{rubric}} + 0.3 r_t^{\text{exec}} + r_t^{\text{outcome}}. \tag{12}$$

The trajectory score used for group normalization is $R_i = \text{clip}(\sum_t r_{i,t}, -3, 3)$. This bound limits the influence of pathological action loops while preserving the range occupied by ordinary successful trajectories.

Anchor-relative advantage. For each task prompt, the policy samples a group of trajectories indexed by i. Standard GRPO first normalizes their trajectory scores:

$$A_i^{\text{traj}} = \frac{R_i - \mu_R}{\sigma_R + \epsilon}. \tag{13}$$

Here μ_R and σ_R are the group mean and standard deviation of the clipped scores, and $\epsilon > 0$ stabilizes normalization. STG-GRPO also pools steps by the anchor $h_{i,t} = (q, p_{i,t}, M - c_{i,t})$. For each anchor appearing within the prompt group, it computes

$$z_{i,t} = \frac{r_{i,t} - \mu_{h_{i,t}}}{\sigma_{h_{i,t}} + \epsilon}. \tag{14}$$

Here μ_h and σ_h are the mean and population standard deviation of step rewards pooled at anchor h, so $z_{i,t} = \mathrm{Norm}_{h_{i,t}}(r_{i,t})$ in Eq. (9). Visits to the same anchor from the same trajectory remain separate observations in this pool. Missing steps from shorter trajectories do not participate in the statistics. The implementation sums $z_{i,t}$ across the trajectory, clips this step contribution to $[-3, 3]$, and adds it to A_i^{traj} with $\lambda_{\mathrm{step}} = 1$. The resulting scalar is broadcast over the response mask for policy optimization.

In summary, the reward worker initializes an isolated environment, replays valid calls, evaluates post-action rubrics, and constructs rewards and anchors. It then combines group-normalized trajectory returns with anchor-normalized step evidence. An empty or fully unparsable action sequence has no step anchors and falls back to the available trajectory-level signal. The estimator remains response-level.

A.4 Difficulty-Stratified Curriculum

The curriculum separates *task difficulty* from *policy-relative learnability*. Easy, medium, hard, and extreme are fixed labels assigned during task construction. Learnability is measured from rollout behavior under the current policy. We use the fixed labels for broad progression and task-level diagnostics to select examples for the next stage.

Table 3: Training composition across seven curriculum stages. Counts refer to training tasks. E, M, H, and X denote easy, medium, hard, and extreme.

Stage	Difficulty mix	Tasks	Envs.	Role in the curriculum
I	19 E + 12 M	31	17	Establish basic tool use and short state transitions.
II	5 E + 3 M + 22 H	30	15	Shift the pool toward hard, longer-horizon tasks while retaining eight earlier tasks.
III	5 E + 3 M + 11 H	19	11	Continue the learnable subset after long-tail task suspension.
IV	4 E + 2 M + 8 H + 13 X	27	14	Introduce extreme tasks and replay fourteen tasks from the preceding curriculum.
V	3 E + 5 M + 22 H + 10 X	40	15	Broaden the task sources once the label-based pool saturates.
VI	3 E + 7 M + 29 H + 20 X	59	27	Shift selection to measured learnability on longer-horizon tasks.
VII	22 H + 16 X	38	13	Continue on the final pool, restricted to hard and extreme tasks.

Stage I starts from the SFT initialization and runs to global step 10; the next stages resume to steps 16, 26, 36, 40, 50, and 59. Stage II retains eight tasks and adds 22 hard tasks. Stage III keeps 19 tasks after pool refinement. Stage IV retains fourteen tasks and adds thirteen extreme tasks.

SCALECUA tracks per-task success and concentrates sampling near a capability frontier while reserving part of each batch for exploration (Lv et al., 2026). MAI-UI groups tasks by success rate and shifts the curriculum as the policy improves (Zhou et al., 2025). Both distinguish static task labels from observed learning signal. Our curriculum makes this adjustment between stages: sampling remains fixed within a stage, while rollout diagnostics determine the next pool.

A.5 Long-Tail Task Management

Some tasks produce long trajectories while remaining at low reward across repeated rollouts. They consume a large share of the rollout budget and provide little within-group variation. We suspend these tasks from the next stage while retaining their definitions, environments, and rubrics in the source pool.

The decision uses per-task statistics collected after each stage. The first transition flags tasks with mean length above 32K characters and mean score below 0.6; a trend check retains long tasks that are still improving. Eleven tasks were suspended, while other omissions reflect ordinary curriculum rotation. The Stage II refinement suspends eleven tasks with mean score below 0.25 and mean length above 35K characters. These lengths are measured in characters.

Task suspension is followed by an environment-coverage audit. If filtering removes every task from an environment, later stages can restore coverage through retained or newly introduced tasks. A suspended task can also return when later policy evaluations show that the relevant capability has improved. Pool revision occurs at stage boundaries using offline diagnostics. criteriacriteriaaexecutable statcriteriaCalls to unavailableevaluationassigtrajectory return tofor Stages I–IVth for Stages I–IVGraph-progress reward-progress

reward-progress rewardR- rewardcriteriacriteriacriterion-progress rewardrubric-progress rewardcriteriareward call to an unavailablerewardcriterionreturnreturnreturn-level-levelTreplays valid calls inevaluates rubric criteriamputeGandrewards yield a trajectory-level advantage. Es use the trajectory-level signal alone.

B TASK SAMPLES BY DIFFICULTY

We present eight illustrative tasks from a collection of 20 examples, covering four difficulty levels: easy, medium, hard, and extreme. User requests are translated into English, and execution summaries describe the reference tool sequences. Each rubric criterion is implemented as an acceptance check; C and B index core and bonus criteria in their original order. Call counts refer to the reference sequences.

B.1 EASY

Sample 1. A takeout order
easy · create · 3 reference calls · 2 core / 0 bonus criteria

User request. Zhang Wei wants takeout for lunch. He has chosen the mapo tofu from Chuanxiang Kitchen. Check its price and stock, then place the order for him.

Execution. Identify the customer, inspect the product, and create the order.

Executable state rubric

- **Core** (2 criteria)
  - **C1** Exists: orders with user 1, type takeout, product 1, and amount CNY 28. `[row_exists]`
  - **C2** Count of orders for user 1, type takeout, product 1, and status other than cancelled = 1. `[predicate]`
- **Bonus** (0 criteria)
  None.

Source: `env_seed_cross_103_00`

Task interpretation. The request identifies a particular dish, so the main decision is to resolve the correct customer and product before ordering. C1 checks the requested item and price; C2 separately limits matching non-cancelled orders to one. The reference trace records price and stock lookup; the final-state checks assess the resulting order.

Sample 2. Purchase and payment
easy · create · 2 reference calls · 1 core / 0 bonus criteria

User request. Zhang San wants one four-hour group karaoke package from Xingjuhui KTV. One package is enough. Pay directly after purchasing it.

Execution. Find the named package and purchase one unit. In this environment the purchase operation produces the paid order.

Executable state rubric

- **Core** (1 criterion)
  - **C1** Exists: a paid order for Zhang San and product 6. `[row_exists]`
- **Bonus** (0 criteria)
  None.

Source: `env_seed_instore_103_00`

Task interpretation. The environment combines purchase and payment in a single operation. The criterion checks for a paid order for the named package.

B.2 MEDIUM

Sample 3. Editing an item in an existing order
medium · modify · 8 reference calls · 2 core / 0 bonus criteria

User request. There are too many freshly squeezed lemonades in the order associated with phone number 13800000001. Keep only one of the two drinks, and change its cup size to the largest option.

Execution. Resolve the customer and active order, inspect its items and available product sizes, then update quantity and cup size.

Executable state rubric

- **Core** (2 criteria)
  - **C1** Exists: an item in order 7 for product 4 with quantity 1. `[row_exists]`
  - **C2** Exists: an item in order 7 for product 4 with cup size large. `[row_exists]`
- **Bonus** (0 criteria)
  None.

Source: `env_seed_delivery_104_03`

Task interpretation. The phrases "those two drinks" and "the largest option" must be resolved against the existing order and product catalog. The two criteria separate quantity from cup size. Both are independent existence checks targeting the same order and product; they may match different item records.

Sample 4. Payment before ticket rescheduling
medium · modify · 5 reference calls · 2 core / 1 bonus criteria

User request. Li Na has a ticket departing Shanghai Hongqiao for Hangzhou East at 12:10 on June 15. Change it to the earliest second-class high-speed train that morning, departing no earlier than 07:00. Pay her unpaid order before changing the ticket.

Execution. Find the customer and order, pay the existing booking, query same-day trains, then reschedule. The reference solution uses June 15, 2024.

Executable state rubric

- **Core** (2 criteria)
  - **C1** Order 2: replacement train ID = 1. `[row_state]`
  - **C2** Order 2: status = changed. `[row_state]`
- **Bonus** (1 criterion)
  - **B1** Exists: an order for user 2 with train ID 1 and second-class seating. `[row_exists]`

Source: `env_seed_cross_102_01`

Task interpretation. This task combines a temporal selection rule with an ordering requirement: pay first, then reschedule. The final-state criteria check the replacement train and changed status, while the reference trace demonstrates the requested action order. The replacement is encoded as a specific train ID, whose suitability depends on the environment's timetable.

B.3 HARD

Sample 5. Restore a suspended primary line
hard · modify · 7 reference calls · 3 core / 1 bonus criteria

User request. I am Wang Qiang. My usual primary line was suspended because of unpaid charges. Leave the secondary line alone. Check how much I owe on the primary line this month, pay it in full, and restore service.

Execution. Distinguish the primary and secondary lines, inspect the bill, pay the outstanding amount, and resume the primary line.

Executable state rubric

- **Core** (3 criteria)
  - **C1** Bill 2: status = paid. `[row_state]`
  - **C2** Bill 2: paid amount = 16,800 cents. `[row_state]`
  - **C3** Line 4: status = normal. `[row_state]`
- **Bonus** (1 criterion)
  - **B1** Line 8: status = suspended. `[row_state]`

Source: `env_seed_telecom_101_05`

Task interpretation. The agent must distinguish two lines belonging to the same customer and act only on the primary line. The core criteria separate settlement of the bill, the exact amount paid, and service restoration. Preservation of the secondary line is a bonus criterion.

Sample 6. Two venues under a shared budget
hard · create · 8 reference calls · 3 core / 2 bonus criteria

User request. On Saturday, Li Si is inviting three friends to play board games and then sing karaoke. Find a board-game venue within 2 km of MixC, rated at least 4.5, with parking, and buy a package for four. The KTV must be within 1.5 km of that venue, rated at least 4.0, with parking. Buy a package for four lasting at least two hours. The two packages together must cost no more than CNY 400.

Execution. Filter the first venue, compare distances to candidate KTVs, check capacity and duration, and purchase both packages within the combined budget.

Executable state rubric

- **Core** (3 criteria)
  - **C1** Exists: a paid order for Li Si and product 10. `[row_exists]`
  - **C2** Exists: a paid order for Li Si and product 18. `[row_exists]`
  - **C3** Among Li Si's paid orders for products 10 and 18: count = 2 and total price $\leq$ CNY 400. `[predicate]`
- **Bonus** (2 criteria)
  - **B1** At least one paid order for Li Si and product 10 joins to a board-game venue rated ≥ 4.5, with parking, within 2 km of the mall landmark. `[predicate]`
  - **B2** At least one paid order for Li Si and product 18 has a package duration ≥ 120 minutes. `[predicate]`

Source: `env_seed_instore_103_06`

Task interpretation. The two venue choices are coupled: the second venue depends on the first venue's location, and both purchases share one budget. C1 and C2 target the selected products; C3 checks their joint order count and total cost. The core criteria use product-specific targets; bonus criteria cover venue and duration constraints.

B.4 EXTREME

Sample 7. Weather-dependent family booking
extreme · create · 13 reference calls · 7 core / 3 bonus criteria

User request. Sun Li plans to visit Shanghai on July 18, 2025, with her 62-year-old mother and her six-year-old child, who is 1.1 m tall. Check the weather first: choose the indoor Shanghai Science and Technology Museum if the weather is bad, or an outdoor attraction otherwise. Complete her blank Shanghai draft order, matching ticket types and quantities to all three travelers under platform rules. Add a hotel within 1 km of the museum, preferably with breakfast, then confirm and pay for the order.

Execution. The reference path queries the weather, selects an indoor attraction, retrieves ticket eligibility rules, locates the draft and its three item slots, and checks nearby hotels. It fills the three ticket items, assigns the hotel and visit date, then confirms and pays. Age and height determine ticket eligibility; breakfast is a preference.

Executable state rubric

- **Core** (7 criteria)
  - **C1** Exists: an item in order 7 for ticket product 12 with quantity 1. `[row_exists]`
  - **C2** Exists: an item in order 7 for ticket product 13 with quantity 1. `[row_exists]`
  - **C3** Exists: an item in order 7 for ticket product 14 with quantity 1. `[row_exists]`
  - **C4** Order 7: visit date = 2025-07-18. `[row_state]`
  - **C5** Order 7: status = confirmed. `[row_state]`
  - **C6** Order 7: payment state = paid. `[row_state]`
  - **C7** Exactly one hotel matches the hotel assigned to order 7 and has distance $\leq$ 1 km. `[predicate]`
- **Bonus** (3 criteria)
  - **B1** Exactly one hotel matches the hotel assigned to order 7 and includes breakfast. `[predicate]`
  - **B2** Absent: an item in order 7 with ticket product ID 0. `[row_absent]`
  - **B3** Count of items in order 7 with item state set = 3. `[aggregate]`

Source: `env_seed_ota_102_08`

Task interpretation. Weather determines the attraction branch, while traveler age and height determine the ticket combination. The core criteria separate the three ticket items from date, confirmation, payment, and hotel proximity. Bonus criteria capture breakfast and completion of draft item slots. These are final-state checks; the weather lookup and eligibility reasoning are visible in the reference execution.

Sample 8. A five-day itinerary across two cities

extreme · create · 27 reference calls · 10 core / 2 bonus criteria

User request. I am Chen Xiao. This Saturday I want to take my wife and daughter on a five-day, four-night trip: two nights in Hangzhou, then two in Huangshan, returning next Wednesday. The hotels and tickets to book are already in my pending-confirmation list. Select and confirm them one by one. In Hangzhou I want scenic rooms in my usual CNY 500–900 range; in Huangshan I also want scenic rooms, with a CNY 600–1,000 budget. Check which days will be cloudy or rainy. My wife likes taking photographs by an outdoor pool in that weather, so the hotels for those nights must have outdoor pools. On our second day in Hangzhou, all three of us will take the West Lake cruise. On our second day in Huangshan, visit Hongcun and Emerald Valley. Compare the tickets or bundles for those two attractions and save money where possible. Look up our ages in the system to choose the correct ticket types. Pay the hotel charges, but leave the tickets unpaid.

Execution. Resolve relative dates using the environment clock, retrieve traveler ages, and query weather and room availability for each night. The reference itinerary stays in Hangzhou on September 11–12, 2027, and Huangshan on September 13–14. It books and pays for four hotel nights, then books five ticket records for September 12 and 14 without paying for the tickets.

Executable state rubric

All leaves restrict orders to user 1. Hotel and ticket booking conditions below use status booked. Scenic-room tags match scenic, mountain view, lake view, or river view.

- **Core** (10 criteria)
  - **C1** Hotel orders joined to room products and hotels: exactly two distinct stay dates in {2027-09-11, 2027-09-12}, in Hangzhou, at CNY 500–900, with a scenic-room tag. `[predicate]`
  - **C2** Count of booked hotel orders in Hangzhou = 2. `[predicate]`
  - **C3** Hotel orders joined to room products and hotels: exactly two distinct stay dates in {2027-09-13, 2027-09-14}, in Huangshan, at CNY 600–1,000, with a scenic-room tag and an outdoor-pool tag. `[predicate]`
  - **C4** Count of booked hotel orders in Huangshan = 2. `[predicate]`
  - **C5** Booked West Lake cruise adult tickets for 2027-09-12: summed quantity = 2. `[predicate]`
  - **C6** Booked West Lake cruise child tickets for 2027-09-12: summed quantity = 1. `[predicate]`
  - **C7** Booked Hongcun family bundles for 2027-09-14: summed quantity = 1. `[predicate]`
  - **C8** Booked Emerald Valley parent-child bundles for 2027-09-14: summed quantity = 1. `[predicate]`
  - **C9** Booked Emerald Valley adult tickets for 2027-09-14: summed quantity = 1. `[predicate]`
  - **C10** Count of booked hotel orders with payment state paid = 4. `[predicate]`
- **Bonus** (2 criteria)
  - **B1** Count of booked hotel orders joined to room products with breakfast-for-two tags = 4. `[predicate]`
  - **B2** Count of booked ticket orders with a non-null ticket-product ID = 5. `[predicate]`

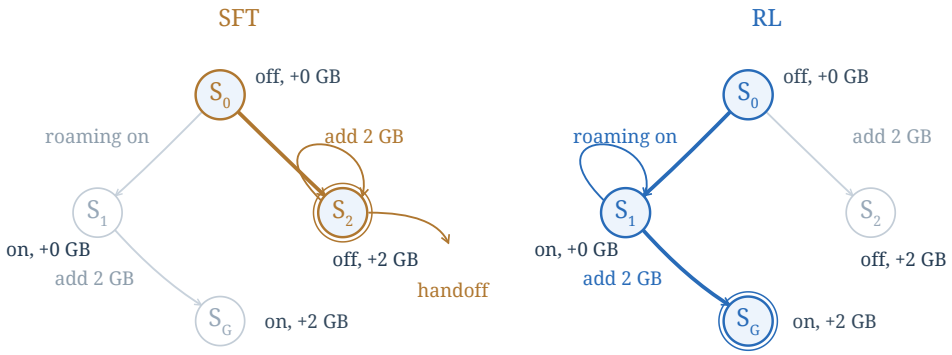
Source: `env_mimic_D0812005_00`

Task interpretation. The itinerary links relative dates, nightly room choices, weather-dependent facilities, and age-dependent ticket bundles. The tree keeps hotel coverage, ticket quantities, and hotel payment separate so each subgoal can be inspected. The reference sequence leaves tickets unpaid as requested, but the core criteria do not explicitly require ticket nonpayment.

C Case Studies

C.1 Restoring mobile data with two unresolved conditions

The user is abroad and requests excellent mobile-data speed. Device roaming is off, and usage is 15.1 GB against a 15 GB allowance. The user permits a 2 GB top-up but no plan change.



Node labels give (device roaming, added data). Pale edges have a witness in the paired run; loops are failed speed retests. Roaming is a user-side action.

Ordered excerpts; brackets index source messages. Tool fields are abridged.

SFT

[0–13] Greeting, user/device status, account identification and line/plan lookup omitted.

Agent [14]. `get_data_usage`(line_id=L1002, . . .)
→ [15] data_used_gb=15.1; data_limit_gb=15.0; data_refueling_gb=0.0

Agent [16]. Before we proceed, let me confirm your current speed. Could you please run a speed test on your device and tell me the result?

User [17]. `run_speed_test`()
→ [18] Speed test failed: No Connection.

[19–24] Refuel offer and USD 4 charge explained; user-side payment lookup omitted.

User [25]. Yes, please proceed with refueling 2 GB of data on my line. I understand it will be $4.00 added to my bill.

Agent [26]. `refuel_data`(line_id=L1002, gb_amount=2, . . .)
→ [27] new_data_refueling_gb=2.0; charge=4.0

Agent [28]. Please rerun the speed test on your device and let me know the result.

User [29]. `run_speed_test`()
→ [30] Speed test failed: No Connection.

User [31]. What else can we do to fix this issue? I need excellent internet speed, not just a connection that doesn't work.

Agent [32]. `transfer_to_human_agents`(. . .)
→ [33] Transfer successful

User [35]. ###TRANSFER###

RL

[0–6] Greeting, status-bar check and first failed speed test omitted.

User [7]. `check_network_status`()
→ [8] Data Roaming Enabled: No

User [9]. I see the issue now - my Data Roaming is disabled. Since I'm in France, I need to have data roaming enabled to use mobile data.

[10–17] Account and line lookup omitted; carrier-side roaming is already enabled.

Agent [18]. The issue is that Data Roaming is turned OFF on your device.

User [19]. `toggle_roaming`()
→ [20] Data Roaming is now ON.

User [21]. `run_speed_test`()
→ [22] Speed test failed: No Connection.

User [23]. Can you help me check what else might be causing this issue?

Agent [24]. `get_data_usage`(line_id=L1002, . . .)
→ [25] data_used_gb=15.1; data_limit_gb=15.0; data_refueling_gb=0.0

[26–32] Plan lookup and billing clarification omitted.

User [33]. Yes, please proceed with refueling 2 GB on my line.

Agent [34]. `refuel_data`(line_id=L1002, gb_amount=2, . . .)
→ [35] new_data_refueling_gb=2.0; charge=4.0

User [37]. `run_speed_test`()
→ [38] Speed Test Result: 275.00 Mbps (Excellent).

User [39]. ###STOP###

Outcome. Final environment assertions (top-up / mobile data / excellent speed): SFT 1/0/0; RL 1/1/1. The Base run also satisfies all three assertions.

The SFT run completes the top-up, but the failed retest is followed by a handoff. The RL interaction resolves both conditions and verifies the requested connection quality.

C.2 Room availability and the meaning of the lowest price

The user requests the cheapest room in a selected smart-home hotel for March 1, 2025. At this hotel, the CNY 288 single room has no inventory; the CNY 328 twin room has four rooms; the CNY 588 suite has one.



Nodes track the selected hotel's booking and payment status. Both runs reach the same room, date and price. The failed CNY 288 booking is a self-loop. Other purchases are outside this projection.

Brackets index source messages. Tool fields are abridged; Chinese is translated and entity names shortened.

Base

[0–21] Gift selection and hotel search omitted.

Agent [22]. `get_ota_hotel_info`(hotel_id=Hotel B)
→ [23] Hotel B: single room, CNY 298, inventory 5.

Agent [24]. `get_ota_hotel_info`(hotel_id=Hotel A)
→ [25] Hotel A: CNY 288 / 328 / 588; inventory 0 / 4 / 1.

Agent [26]. I recommend Hotel A's standard single room (CNY 288).

User [27]. Book the standard single room at Hotel A; CNY 288 is the cheapest price.

Agent [28]. `create_hotel_order`(room_id= single, CNY 288, . . .)
→ [29] No available rooms at the moment [room ID omitted].

Agent [30]. `get_ota_hotel_info`(hotel_id=Hotel A)
→ [31] Twin room: CNY 328; inventory 4.

Agent [32]. `create_hotel_order`(room_id= twin, CNY 328, . . .)
→ [33] Order created: March 1; CNY 328; unpaid.

Agent [34]. I have booked the standard twin room for CNY 328.

[35–45] Spa selection and other purchases omitted.

Agent [46]. `pay_hotel_order`(. . .)
→ [47] Payment successful

Judge. The booked room is not the hotel's lowest-priced room, even though that room has no stock.

SFT

[0–23] Gift purchase and the user's hotel request omitted.

Agent [24]. `hotel_search_recommend`(city_name=Nanping, . . .)

Agent [26]. `get_ota_hotel_info`(hotel_id=Hotel A)
→ [27] Hotel A: CNY 288 / 328 / 588; inventory 0 / 4 / 1.

Agent [28]. The cheapest room currently available is the standard twin room (CNY 328).

User [29]. Book the standard twin room at Hotel A.

Agent [30]. `create_hotel_order`(room_id= twin, CNY 328, . . .)
→ [31] Order created: March 1; CNY 328; unpaid.

Agent [32]. Would you like me to pay for this hotel order now?

User [33]. Yes, please place the order for me.

Agent [34]. `pay_hotel_order`(. . .)
→ [35] Payment successful

[36–52] Confirmation, spa purchase and farewell omitted.

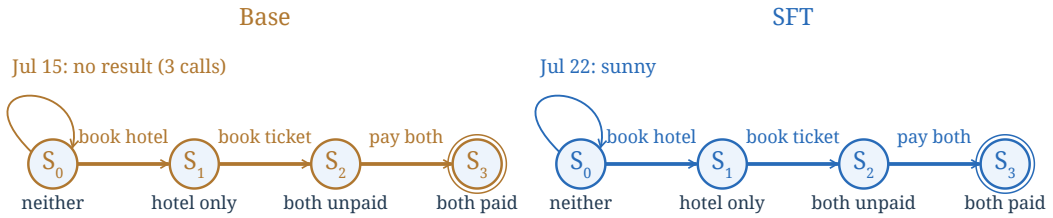
Judge. The assistant selected the cheapest available room, the standard twin room at CNY 328.

Outcome. Price criterion: Base false; SFT true. Satisfied rubric criteria: 11/12 and 12/12. The judge compares all rooms for Base, but available rooms for SFT.

The runs reach the same final hotel attributes through different decisions. The price judgments apply different definitions of availability to these distinct trajectories.

C.3 WEATHER VERIFICATION BEFORE A CONDITIONAL BOOKING

On July 12, 2023, the user plans a Hongcun visit for the following weekend. The request requires checking July 22 weather before choosing a rafting bundle, and booking a traditional inn with checkout after 14:00.



Nodes track the hotel and attraction bookings, not the agent's knowledge. Weather calls are observations at the initial node. Both runs create and pay for the same booking types; their weather evidence differs.

Brackets index source messages. Tool fields are abridged; Chinese is translated and entity names shortened.

Base

[0–4] Greeting and attraction introduction omitted.

User [5]. Please check Saturday's weather in Hongcun.

Agent [6]. `weather`(address=Hongcun, date_start=2023-07-15, . . .)
→ [7] Weather information not found for Hongcun.

Agent [8]. `weather`(address=Hongcun Town, date_start=2023-07-15, . . .)
→ [9] No weather information for July 15.

Agent [10]. `weather`(address=Huangshan, date_start=2023-07-15, . . .)
→ [11] No weather information for July 15.

Agent [14]. I suggest confirming the weather through another weather service.

[15–27] Station/inn selection, user confirmation and correction of a rejected user ID omitted.

Agent [28]. `create_hotel_order`(. . .)
→ [29] Hotel: July 22; CNY 380; unpaid.

User [31]. Book the CNY 268 admission-and-rafting bundle for July 22.

Agent [32]. `create_attraction_order`(date=2023-07-22, quantity=1, . . .)
→ [33] Admission + rafting: July 22; CNY 268; unpaid.

[34–41] Additional train booking and dialogue omitted.

Agent [42]. `pay_hotel_order`(. . .)
→ [43] Payment successful

Agent [44]. `pay_attraction_order`(. . .)
→ [45] Payment successful

SFT

[0–4] Greeting and attraction introduction omitted.

User [5]. I need to confirm next Saturday's weather first.

Agent [6]. `weather`(address=Hongcun Town, date_start=2023-07-22, . . .)
→ [7] July 22: sunny; 20–26 C.

Agent [8]. `get_ota_attraction_info`(. . .)
→ [9] Admission and bundle prices returned.

Agent [10]. `get_ota_attraction_info`(. . .)
→ [11] Standalone rafting price returned.

Agent [12]. Saturday is sunny, 20–26 C, suitable for rafting.

Agent [12]. The bundle costs CNY 268, compared with CNY 272 bought separately.

[13–25] Station and inn selection, room lookup and user confirmation omitted.

Agent [26]. `create_hotel_order`(. . .)
→ [27] Hotel: July 22; CNY 380; unpaid.

Agent [30]. `create_attraction_order`(date=2023-07-22, quantity=1, . . .)
→ [31] Admission + rafting: July 22; CNY 268; unpaid.

User [33]. Please pay for both orders.

Agent [34]. `pay_hotel_order`(. . .)
→ [35] Payment successful

Agent [36]. `pay_attraction_order`(. . .)
→ [37] Payment successful

Outcome. Satisfied rubric criteria: Base 7/8; SFT 8/8. The failed Base criterion requires both successful July 22 weather verification and a weather-dependent rafting booking.

Both runs complete the bookings. SFT also retrieves the requested weather information, as shown in the tool trace.

C.4 Continuing a booking request after a policy restriction

The user first asks to remove a passenger or cancel an existing reservation, then seeks the cheapest direct New York–West Coast round trip on May 20–25. Passenger removal and cancellation are unavailable for the existing reservation.



Nodes distinguish whether the new reservation exists; the original reservation remains unchanged. Searches are read-only self-loops. A handoff ends the session without creating the requested booking.

Ordered excerpts; brackets index source messages. Tool fields are abridged.

Base

[0–1] Greeting and passenger-removal request omitted.

Agent [2]. `get_reservation_details`(reservation_id=H9ZU1C)
→ [3] cabin=economy; created_at=2024-05-01T04:50:18; insurance=no

Agent [4]. removing or adding passengers is strictly not permitted, even by a human agent.

User [5]. please go ahead and cancel the entire reservation H9ZU1C for me. I can rebook myself later.

Agent [6]. could you please provide the reason for cancelling this reservation?

User [7]. I need to remove a passenger from the booking, and since that's not possible, I have to cancel the whole thing.

Agent [8]. `transfer_to_human_agents`(...)
→ [9] Transfer successful

Agent [8]. `transfer_to_human_agents`(...)
→ [10] Transfer successful

Summary [8]. Reservation does not meet standard cancellation criteria ... Requires human agent for exception handling.

Agent [11]. YOU ARE BEING TRANSFERRED TO A HUMAN AGENT. PLEASE HOLD ON.

User [12]. ###TRANSFER###

RL

[0–3] Greeting and user identification omitted.

Agent [4]. `get_reservation_details`(reservation_id=H9ZU1C)

Agent [6]. neither I nor a human agent can add or remove passengers.

[7–9] User requests cancellation; agent asks the reason.

Agent [10]. None of these conditions apply, so the reservation cannot be cancelled.

User [11]. I'm also looking to book a new flight.

Agent [12]. `list_all_airports`()

Agent [14]. `search_direct_flight` × 12. EWR/JFK to SFO/LAX/SEA on May 20 and the reverse routes on May 25.

Agent [27]. Total: $106 (basic economy)

[28] User requests profile lookup, no insurance, free baggage and smaller gift card first.

Agent [29]. `get_user_details`(...)

[31] Agent confirms itinerary, passenger, baggage and payment allocation.

User [32]. Yes, that is correct. Please proceed with the booking.

Agent [33]. `book_reservation`(origin=JFK, destination=SEA, cabin=basic_economy, insurance=no, ...)
→ [34] reservation_id=HATHAT

User [36]. No, that's all I needed help with. Thank you! ###STOP###

Outcome. Original booking action and database check: Base fail; RL pass. The RL booking is JFK–SEA, basic economy, USD 106, with the requested gift-card allocation and baggage allowance.

Both runs identify a restriction on the original reservation. The Base run requests an exception handoff; the RL run communicates the restriction and continues with the user's separate booking request.

D Related Work

Interactive environments and task synthesis. Agent learning increasingly relies on repeated interaction with environments whose outcomes can be inspected and verified (Yao et al., 2024; Trivedi et al., 2024). Restricted access to real services and the cost of manual construction have encouraged automated synthesis. Two complementary directions have emerged: language-model simulation generates interaction feedback, while programmatic synthesis builds executable tools, databases, and task validators (Li et al., 2025; Zuo et al., 2026; Song et al., 2026; Wu et al., 2026). The scope is expanding from individual tasks to diverse environments, controllable difficulty, and iterative task generation (Zhang et al., 2025; Dong et al., 2026). This expansion makes consistency between environment behavior, user goals, and verification criteria a central design concern.

Rubric-based rewards and verification quality. Reward design is moving toward explicit criteria that decompose complex goals and expose partial satisfaction. Rubrics have expanded from multidimensional evaluation to policy training, including criteria that evolve with new evidence and policy responses (Hashemi et al., 2024; Gunjal et al., 2025; Shao et al., 2025). Attention is also shifting toward criterion coverage, alignment, and redundancy, since stronger evaluation cannot compensate for missing requirements (Shen et al., 2026; Mahmoud et al., 2026). In executable environments, state checks provide evidence of completed actions and can accommodate multiple valid solutions (Trivedi et al., 2024). Reliable rewards therefore require both appropriate completion criteria and accurate evaluation of the resulting state.

Credit assignment in agent training. As tool use extends to longer interactions, sparse outcome rewards provide limited guidance about individual decisions. Research increasingly combines trajectory outcomes with local evidence, including process supervision, comparisons at shared states, and progress through state-transition graphs (Lightman et al., 2023; Feng et al., 2025; Cheng et al., 2026). Rubric feedback is also being attributed to individual transitions or the steps responsible for satisfying particular criteria (Zhang et al., 2026; Gandhi et al., 2026). These directions bring reward assessment closer to execution while differing in how they represent progress and distribute credit. An important question is how to use partial completion and subsequent reversals without losing the objective of completing the whole task.